João Rafael Moraes Nicola

A formal analysis of Identity and Sortality in the Unified Foundational Ontology (UFO)

Vitória, ES, Brazil

2021

João Rafael Moraes Nicola

A formal analysis of Identity and Sortality in the Unified Foundational Ontology (UFO)

PhD Thesis submited to the Informatics Graduate Program of the Federal University of Espírito Santo as a requirement for obtaining a PhD degree in Computer Science, under the supervision of Prof. Dr. Giancarlo Guizzardi.

Universidade Federal do Espírito Santo – UFES

Departamento de Informática

Programa de Pós-Graduação em Informática

Supervisor: Prof. Dr. Giancarlo Guizzardi

Vitória, ES, Brazil

2021

Nicola, João Rafael Moraes, 1978-
N634f A formal analysis of Identity and Sortality in the Unified Foundational Ontology (UFO) / João Rafael Moraes Nicola. - 2021.
410 f. : il.

Orientador: Giancarlo Guizzardi.
Tese (Doutorado em Informática) - Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Ontologia Aplicada. 2. Modelagem Conceitual. 3. Engenharia de Software. 4. Sistemas de Informação. I. Guizzardi, Giancarlo. II. Universidade Federal do Espírito Santo. Centro Tecnológico. III. Título.

CDU: 004

João Rafael Moraes Nicola

A formal analysis of Identity and Sortality in the Unified Foundational Ontology (UFO)

PhD Thesis submited to the Informatics Graduate Program of the Federal University of Espírito Santo as a requirement for obtaining a PhD degree in Computer Science, under the supervision of Prof. Dr. Giancarlo Guizzardi.

Approved. Vitória, ES, Brazil, April, 20th, 2021:

Prof. Dr. Giancarlo Guizzardi

Prof. Dr. Nicola Guarino

Prof. Dr. Walter Alexandre Carnielli

Prof. Dr. João Paulo A. Almeida

Prof. Dr. Daniele Porello

Vitória, ES, Brazil
2021

To my dearest, always by my side.

To my children, that shall face the challenges of the tomorrow.

To my god, who brought us together.

Abstract

The Unified Foundational Ontology (UFO) is a conceptual framework grounded on principles derived from Ontology discipline of Philosophy, with applications in the field of Software Engineering, specially in Conceptual Modeling, as the semantic reference for the OntoUML modeling language. Among the concepts described in the UFO fragment of endurants (UFO-A), the concept of sortality plays a central role in the classification of UFO substantial universals. However, this concept, and the related concepts of identity and individuality currently lack a systematic formal characterization, hindering their application in the analysis of substantial universals. This research enriches the Unified Foundational Ontology (UFO) literature with a formal specification for a fragment of UFO-A that allows the characterization of these concepts. This specification is presented in Isabelle/HOL, a logical formalism that allows a machine-assisted verification. We construct a formal framework, based on this specification and on categoric-theoretic concepts through which we propose formal definitions for the concepts of individuality and identity, and, from these definitions, we propose a formal characterization of the concept of sortality. Illustrations and the application of the proposed definitions on the domain of conceptual modeling are also presented.

Keywords: applied ontology, conceptual modeling, identity, individuality, sortality.

Resumo

A Ontologia de Fundamentação Unificada (UFO) é um arcabouço conceitual firmado em princípios derivados da disciplina de Ontologia da Filosofia, com aplicações no campo de Engenharia de Software, especificamente em Modelagem Conceitual, servindo como fundamento semântico para a linguagem de modelagem OntoUML. Entre os conceitos descritos no fragmento de endurantes da UFO (UFO-A), o conceito de sortalidade desempenha um papel central na classificação dos universais de substâncias. Contudo, esse conceito e os conceitos conexos de identidade e individualidade carecem atualmente de uma formalização sistemática, o que dificulta suas aplicações na análise e modelagem dos universais de substância. Esta pesquisa enriquece a literatura da UFO com uma especificação formal para um fragmento de UFO-A que permite a caracterização destes conceitos. Esta especificação é apresentada em Isabelle/HOL, um formalismo lógico que permite uma verificação auxiliada por máquina. Construímos um arcabouço formal com base nesta especificação e em conceitos da Teoria de Categorias, através dos quais propomos definições formais para os conceitos de individualidade e identidade, e, a partir destas, propomos também uma caracterização formal do conceito de sortalidade. São também apresentados exemplos das aplicações das definições propostas na atividade de modelagem conceitual.

Palavras-chave: ontologia aplicada, modelagem conceitual, identidade, individualidade, sortalidade.

Contents

1 INTRODUCTION . . . 15
1.1 Problem, Motivations . . . 16
1.2 Research Hypothesis . . . 17
1.3 Research Objectives . . . 18
1.4 Justification . . . 18
1.5 Research Scope . . . 19
1.6 Methodological Aspects . . . 19
1.7 Brief Report . . . 20
1.8 Thesis Structure . . . 21

2 PRELIMINARY REVIEW . . . 23
2.1 Ontology and its meanings . . . 23
2.2 Axiomatic Ontological Theories . . . 25
2.3 Proof Assistant Systems . . . 27
2.3.1 Isabelle . . . 28
2.4 Conceptual Modeling and OntoUML . . . 30
2.5 Category Theory . . . 30

3 SIMPLIFIED UFO THEORY . . . 31
3.1 Introduction . . . 31
3.2 Particulars and Possible Worlds . . . 31
3.2.1 Modes of Existence . . . 35
3.3 Substantials, Moments and Inherence . . . 35
3.3.1 No Ultimate Bearer Anomaly . . . 41
3.3.2 Cyclic Inherence Anomaly . . . 41
3.3.3 Infinite Inherence Chain Anomaly . . . 42
3.3.4 Finite-Chain Inherence . . . 48
3.4 Quality Spaces and Qualia . . . 51
3.5 Qualified Particulars . . . 53
3.6 Towardness Relation . . . 54
3.6.1 Towardness theory . . . 56
3.7 Theory of Particulars . . . 57
3.8 Universals and Instantiation . . . 57
3.9 Final Considerations . . . 59

4 PARTICULAR STRUCTURES AND MORPHISMS . . . 61

4.1 Introduction **61**
4.2 Particular Structures **62**
4.3 Particular Structures Homomorphisms **62**
4.3.1 Definitions 63
4.3.2 Axiomatization 64
4.3.3 The Category of Particular Structures 67
4.3.4 Classes of Particular Structure Morphisms 68
4.4 Structural Properties of Particulars **74**
4.4.1 Isomorphically Unique Particulars 74
4.4.2 Permutability 75
4.4.3 Collapsability 75
4.4.4 Permutability and Isomorphical Uniqueness 76
4.5 Twin Spheres Examples **76**
4.6 Final Considerations **80**

5 INDIVIDUALITY AND IDENTIFIABILITY **83**
5.1 Introduction **83**
5.2 Individuality **84**
5.3 Identity Through Permutability **85**
5.3.1 Isomorphically-Invariant Identity Predicates 85
5.3.2 Identifiable Particulars 87
5.3.3 Identifiability, Isomorphical Uniqueness and Permutability 87
5.3.4 Correspondence Function 88
5.4 Identity Through Identity Anchors **89**
5.4.1 Anchoring 89
5.5 Identity Characterization Equivalences **91**
5.6 Anchoring Classes and Identification Dependence **91**
5.6.1 Anchoring Classes 91
5.6.2 Anchoring Sets 93
5.6.3 Identity-Related Dependency Relations 94
5.7 Final Considerations **95**

6 SORTALITY **97**
6.1 Introduction **97**
6.2 Sortals: a change in perspective **98**
6.3 Finite Particular Structures **103**
6.4 Isomorphism-invariant Instantiation **103**
6.5 Universal Trimming Operation **105**
6.5.1 Detailing moments 107
6.6 Formal Characterization of Sortality **109**

6.7 Final Considerations . . . 111

7 APPLICATIONS . . . 113
7.1 Introduction . . . 113
7.2 Falsifiability of Identifiability, Individuality and Sortality . . . 113
7.2.1 Individuality and Identity . . . 114
7.2.2 Sortality . . . 115
7.3 UFO in Isabelle/HOL as a Formal Framework . . . 117
7.4 Final Considerations . . . 119

8 CONCLUSION . . . 121
8.1 Future Works . . . 122

Bibliography . . . 125

APPENDIX 127

APPENDIX A – OPENING THE ISABELLE/HOL SOURCES . . . 129

APPENDIX B – UFO PARTICULARS AND UNIVERSALS . . . 131
B.1 Particulars and Possible Worlds . . . 131
B.1.1 Modes of Existence . . . 135
B.2 Substantials, Moments and Inherence . . . 135
B.2.1 No Ultimate Bearer Anomaly . . . 147
B.2.2 Cyclic Inherence Anomaly . . . 150
B.2.3 Infinite Inherence Chain Anomaly . . . 153
B.2.4 Finite-Chain Inherence . . . 167
B.3 Quality Spaces and Qualia . . . 187
B.4 Qualified Particulars . . . 190
B.5 Towardness Relation . . . 194
B.5.1 Towardness theory . . . 194
B.6 Theory of Particulars . . . 196
B.7 Universals and Instantiation . . . 197

APPENDIX C – UFO PARTICULAR STRUCTURES AND MORPHISMS . . . 205
C.1 Particular Structures . . . 205
C.2 Particular Structures Homomorphisms . . . 209
C.2.1 Definitions . . . 209
C.2.2 Axiomatization . . . 210

C.2.3 The Category of Particular Structures . . . 215
C.2.4 Classes of Particular Structure Morphisms . . . 217
C.2.5 Morphism image structure . . . 234
C.3 Properties of UFO Particular Structures and Morphisms . . . 255
C.3.1 Sub-Structures . . . 255
C.3.2 Inverse Images . . . 279
C.4 Structural Properties of UFO Particulars . . . 298
C.4.1 Isomorphically Unique Particulars . . . 298
C.4.2 Permutability . . . 299
C.4.3 Collapsability . . . 300
C.4.4 Permutability and Isomorphical Uniqueness . . . 301
C.5 Examples of UFO Particular Structures and Morphisms . . . 304
C.5.1 Single Non-Empty World Structures . . . 304
C.6 Twin Spheres Examples . . . 307

APPENDIX D – INDIVIDUALITY AND IDENTIFIABILITY . . . 319
D.1 Individuality . . . 319
D.2 Identity Through Permutability . . . 319
D.2.1 Isomorphically-Invariant Identity Predicates . . . 319
D.2.2 Identifiable Particulars . . . 329
D.2.3 Identifiability, Isomorphical Uniqueness and Permutability . . . 329
D.3 Identity Characterization Equivalences . . . 333
D.3.1 Correspondence Function . . . 334
D.4 Anchoring and Identification Dependence . . . 342
D.5 Identity Through Identity Anchors . . . 342
D.5.1 Anchoring . . . 342
D.6 Anchoring Classes and Identification Dependence . . . 350
D.6.1 Anchoring Classes . . . 350
D.6.2 Anchoring Sets . . . 356
D.6.3 Identity-Related Dependency Relations . . . 364

APPENDIX E – SORTALITY . . . 367
E.1 Finite Particular Structures . . . 367
E.2 Isomorphism-invariant Instantiation . . . 369
E.3 Instantiation in Finite Particular Structures . . . 374
E.4 Universal Trimming Operation . . . 379
E.4.1 Detailing moments . . . 379
E.5 Formal Characterization of Sortality . . . 398

APPENDIX F – AUXILIARY THEORIES . . . 401

1 Introduction

The design and development of an information system (IS) is an activity that bridges the subjective world of human concepts and the formal, objective world of computer programs.

Although the end result of this activity may be a computer program, which at first glance is as nothing more than a formal description of a computable function, the value of an IS rests in the fact its elements hold meanings, for its users, that transcend a mere formal interpretation in terms of an abstract computation.

When a programmer writes an implementation of an algorithm (e.g. a sorting algorithm), or a program fragment that describes an abstract data structure, he or she does not attach any particular meaning to the symbolic data that is manipulated by the algorithm or that is stored in the data structure. Whether or not an algorithm is correct or a data structure implementation is better suited than another is a judgement made solely on *formal* and *abstract* properties of these implementations. For example, the fact that the data stored in a database table refers to cars, food, human beings or fictional characters hold no bearing in the implementation of the algorithms and data structures that underlie the database system.

In contrast, when a user operates an IS designed to keep and process data about vehicles, he or she may reasonably expect that the presence entry `(plate=ZAP-1234, owner=John Doe)` in a database table named `VehicleOwner` signifies that there exists a real vehicle, whose licence plate is *ZAP-1234*, that is owner by a person named *John Doe*.

Thus the correctness and, ultimately, the value of an IS from the persepective of its users, depend on how faithfully the IS represents in its database, reports, forms, etc., the entities whose information it is expected to capture, store, process and present, and whose existence lie beyond the system itself.

In other words, an IS has two main parts: its implementation, encoded as a formal and abstract sequence of symbols, and the conceptual beliefs carried by its users, which determine the interpretation of the program data and behaviour in terms of entities under the intended representation domain of the IS. This last part is called the IS's ontological commitment, i.e. the set of concepts and relationships that form the beliefs regarding the system's domain. In this context, the term ontology designates a "formal, explicit specification" of the "shared conceptualization" that comprises the IS's ontological commitment (GRUBER, 1993b).

For that reason, the IS development process has to take into account *what* are the

entities upon whose representations the system is built. Furthermore, it also necessary to consider *how* the IS's users *conceptualize* these entities.

In this context, this thesis has the goal to enrich the UFO ontology literature with a formal specification for UFO-A (UFO fragment focused on endurants) that includes a characterization of the concepts of identity, individuality and sortality, which presently, though being fundamental to the theory of UFO endurants, lack systematic analysis and formal definitions. The formal specification is presented in Isabelle/HOL, a logical formalism that allows a machine-assisted verification.

The subsequent sections of this chapter detail the problems, hypothesis and goals that direct the research described in this thesis.

1.1 Problem, Motivations

Although the Unified Foundational Ontology (UFO) has been described in its original work (GUIZZARDI, 2005) and in other subsequent works produced at the NEMO (The Ontology & Conceptual Modeling Research Group) research group [1] using a mix of informal and formal notations, there has not yet been a systematic specification of the formal properties of the ontology under a logical formalism that allows an machine-assisted verification of its formal properties.

The lack of such a specification means that the verification of the consistency of the UFO ontology and of its expected properties must be done manually, introducing a potential source of problems due to human interpretation errors or omissions.

An initial formal analysis of the UFO ontology shows that the concept of Sortality of UFO universals is not defined precisely in the original UFO work, since it relies on the concept supplying an identity principle for its instances to be well defined. However, although this expression seems to carry an easy intuitive meaning, it has no direct encoding in the formal theory. The lack of this formal definition means that the concept of sortality, which is a central concept in the UFO ontology of universals and also a relevant element in the interpretation of the OntoUML (GUIZZARDI, 2005) conceptual modeling language, has no clear precise way of been asserted. And thus it introduces an important aspect of subjectivity in the interpretation of OntoUML models and any other theory that relies on this concept.

The third issue is that even tough the UFO presumes that particulars in the theory possess identity and individuality, this concepts are not clearly defined in UFO itself. The lack of a clear definition leads one to the difficulties in framing the meaning of the concept of sortality and it does not give us a clear methodology to access weather or not

[1] The articles are available at <https://nemo.inf.ufes.br/publicacoes/peer-reviewed/>, accessed in 2020-11-24.

the particulars we are dealing with when we are building models of the theory actually posses identity and individuality. Again, the apparent intuitiveness of this concepts hide a complex formal characterization.

An usual definition of a *principle of identity* relies on the existence of a predicate that is capable of expression the conditions that rule the identity relation on a set of individuals. The existence of a predicate, on the other hand, implies the existence (or choice) of a formal language in which it is expressed. The introduction of these concepts at a such fundamental level of the foundational ontology can be considered, at the minimum, inadequate, since it reduces the definition of a fundamental concept regarding endurants (identity, sortality) to concepts that are considered much more specific, i.e. linguistic elements such as predicates or formal languages. Ideally, concepts should either be intuitive enough not to require a formal definition or they should be defined in terms of concepts that are more fundamental.

Regarding the notion of individuality, even though it is mentioned in UFO as a property of its particulars, it is nevertheless left undefined either informally or formally.

In conclusion, the lack of a formal specification in UFO has hidden some gaps in its definition, specifically on the notions of sortality, identity and individuality, which when formally clarified may raise the usefulness of the theory and the precision of its applications.

1.2 Research Hypothesis

The first hypothesis is that it is possible to define formally the concept of individuality for particulars in UFO.

The second hypothesis is that it is possible to define the concept of identity without requiring the existence of some predicate or the choice of some formal language, as explained in the previous section.

The third hypothesis is that a clear definition of the concept of identity leads also to a clear definition of the concept of sortality. The lesser hypothesis is that the clear definition of the concept of sortality may give us a precise strategy for the evaluation of this property in the applications of the UFO, specifically in the verification of OntoUML models.

The fourth hypothesis is the use of a higher order logic instead of a classical first order logic in the analysis and specification of the UFO concepts give us solution possibilities that are not easily translatable to first order logic.

The fifth and last hypothesis is that the formalization of the UFO Ontology in a logical formalism that allows the assisted proof construction and verification using the

computer software allows the precise framing of the problems identified in the previous section.

1.3 Research Objectives

The goal of this research is to achieve a formal analysis and specification of the concepts of identity, individuality and sortality in the context of the UFO-A Ontology of Endurants. More specifically, we aim to achieve this goal by pursuing the following objectives:

1. to identify a fragment of the UFO-A Ontology that is sufficient for the analysis of the concepts of identity, individuality and sortality, and to formally specify this fragment in the Isabelle/HOL logical formalism;
2. to construct a formal framework in Isabelle/HOL that allows the analysis and definition of the concepts of identity, individuality and sortality of UFO particulars;
3. to propose within this formal framework a formal definition for the concepts of individuality and identity, where the definition of the latter does not require the existence of identifying predicates and of a formal language in which these are expressed;
4. to propose a formal definition of the concept of sortality based on the proposed identity definition;
5. to illustrate the application of the proposed definitions on the domain of conceptual modeling.

1.4 Justification

The achievement of the goals stated in the previous section may bring several benefits to the UFO ontology itself and also to the practice of conceptual modeling, more specifically, the conceptual modeling based on the UFO.

It also provides a useful formal framework in which future works about the UFO Ontology which requires some formal treatment could be based on. More specifically, the existence of a formal specification of UFO in a formalism that allows proof assisted construction such as the Isabelle/HOL formalism serves as a basis for any work that requires a formal treatment of the UFO concepts. Even thought our goal is not to provide a complete formal specification of the UFO Ontology, it nevertheless shows the steps that are required to complete this task.

Also the existence of a formal framework that allows the analysis and definition of the concepts of identity, individuality and sortality can also serve as a basis to further define and propose formal concepts based on this notions.

The existence of a formal definition for the concept of identity that does not require the existence of some language, will allow the verification of the UFO sortality and identity even of individuality to be introduced in the existing tools for model checking of conceptual models, such as the OLED (OntoUML Lightweight Editor) tool (GUERSON et al., 2015).

An objective methodology for the evaluation of whether or not particulars have identity and individuality, and whether concepts are sortal or not, can potentially clarify the use of these concepts in the practice of conceptual modeling and remove a source of subjectivity in the interpretation of the UFO Ontology.

Besides the benefits to the research on UFO, the concepts that are the focus of this work, identity and individuality, may also prove useful in areas that go beyond the scope of Information Systems.

1.5 Research Scope

Even thought the inicial scope of this research was to formally specify the whole UFO-A ontology, as explained in section 1.7, the prompt identification of the gaps in the definitions of the concepts of identity, individuality and sortality has given a motivation to restrict the formalization work to a fragment of the UFO Ontology that is sufficient for the analysis of these concepts. Among the notable omissions from the formalization presented in this work are the concepts of parthood, of relators and of events, which are actually part of the UFO-B Ontology. The notion of quality moments is also simplified somewhat. And even though the notion of relators do not appear in this formalization there is an alternative simpler, less expressive, but sufficient element that serve to represent material relations between endurants which is called towardness relation in this work. The diferences from the original work however are just a simplification that allows us to provide a formal analysis that is feasible in the scope of a phd thesis.

1.6 Methodological Aspects

Taking in consideration the goal of proposing a formal specification for a fragment of the UFO Ontology and to provide a formal analysis and specification of the concepts of identity, individuality and sortality in a way that can be verified by software and also considering the lack of alternative definitions of the concepts of identity and individuality in the applied ontology literature, we pursue this research with the following methodological principles:

1. the specification shall be done exclusively in the Isabelle/HOL logical formalism;

2. the specification is restricted to the concepts presented in the original presentation of the UFO Ontology, excluding posterior works that extend or revise this original concepts;

3. formal tools that might be useful for the formal treatment of the concepts of identity shall be looked up in the formal logic literature;

4. given the centralness of the formal specification in this work, the construction of the informal text that composes the chapters that describe the formal theory and the formal results shall be written embedded in the logical theories itself, so as to cement the idea that a research description can be directly tied up to a formally verified specification.

1.7 Brief Report

The initial goal of the author's research work as a phd student was to formally specify the whole UFO-A Ontology in the Isabelle/HOL formalism.

This idea came up during a conversation with Giancarlo Guizzardi, the author of the original UFO work and the author's present phd advisor. However early in the process of formalizing the UFO concepts a few issues were identified by the author regarding the notions of sortality. The effort to provide a formal specification for this notion, which is a notion very important for the applications of UFO, lead to the identification of the lack of a clear definition for the notions of identity and subsequently also of individuality.

At this point the author paused his formalization of the notion of sortality and while focusing his formalization effort on orthogonal concept of parthood, the author also tried to find the solution that could fullfill the gap identified in the original theory regarding those problematic notions.

At some point, the author came up with a novel idea for treating these concepts that did not present the issues identified initially. After explaining this solution to his phd advisor, he was told that these results were by themselves sufficient as a research work in the scope of a phd thesis. From this point on, the focus of the research was narrowed to these concepts and, after some initial prototyping in Isabelle/HOL, an article that described the general idea behind this novel approach was produced and published in (NICOLA, 2018).

However, the use of a category-theoretic approach for analysing the problem was still too abstract to be applied in the context of the UFO. Thus, the development of a more concrete approach, which considered the particularities of the UFO ontology, lead to

the construction of the formalization of the UFO fragment presented in this work and the formal definition of the concepts of identity, individuality and sortality in this context.

1.8 Thesis Structure

This work starts with a preliminary review of the concepts and of the context in which this thesis is inserted, described in chapter 2 . Chapters 3 to 6 describe the formal theory of the fragment of the UFO the analysis and definitions of the concepts of identity, individuality and sortality. More specifically, chapter 3 describes the formalization of the UFO fragment, while chapter 4 propose a formal framework based on the category theory which allows the formal treatment of the concepts of identity, individuality and sortality, while the chapter 5 focuses on the concepts of individuality and identifiability and, making use of the framework proposed in chapter 4, provide a definition for this notions. Finally, on chapter 6 , we provide a formal definition for the notion of sortality based on the definitions of individuality and identifiability proposed in chapter 5. The chapter 7 illustrates the usage of the concepts that were introduced in the previous chapters in the context of the conceptual modeling and of the formal research in the UFO Ontology.

Finally we present a conclusion of this thesis and a discussion regarding future works on chapter 8.

2 Preliminary Review

2.1 Ontology and its meanings

The word "ontology" appears in this work with various denotations. Although this situation may confuse the reader, it reflects the situation in the field under which this work is positioned, which, by the way, is called "Applied Ontology". Before we delve into a deeper discussion of the various concepts underpinning the meanings of that term, let's first briefly summarize them and establish some conventions that shall be useful to avoid confusion in its use.

Some of the meanings attributed to the term "ontology" are described in the Merrian-Webster Online dictionary [1]:

> "1: a branch of metaphysics concerned with the nature and relations of being.
> 2 : a particular theory about the nature of being or the kinds of things that have existence."

According to the dictionary, "ontology" is a word that may denote a branch of Metaphysics, itself a branch of Philosophy, or a particular theory, representing a particular belief about what existence is or which kind of things exist. In the former case, we refer to it using a proper proper uncountable noun form, as in "Ontology is the study of existence and modes of existence". In the later case we use it as common countable noun, as in "an ontology of universals". The term "Ontology" also appears in Philosophy qualified as "Formal Ontology" denoting the study of the subjects of Ontology using the tools of Formal Logic, which provides the means for the construction of formal theories regarding topics such as the *nature of objects*, *events*, *parts and wholes*, *relations*, *properties*, etc., which can be called "formal ontologies" or formal theories about general aspects of being and existence.

The term "ontology" also appears in the field of Information Systems or Software engineering. Gruber defines ontology as "an explicit specification of a conceptualisation" (GRUBER, 1993a). In this sense, an ontology is an artifact whose purpose if to "describe *ontological commitments* for a set of agents, so that they can communicate about a domain of discourse without necesarily operating on a global shared theory". The basic premise is that even though two or more agents may need to communicate to achive some common purpose, their *conceptualisations* or *ontological commitments* may differ in such a way that

[1] "ontology." Merriam-Webster.com. 2020. https://www.merriam-webster.com (3 Feb 2020).

the effectiveness of their communication is impaired. The term "conceptulization" denotes a system of concepts that represent an abstraction of a portion or aspect of reality. An ontology is an *explicit* speficiation because it describes all the concepts and assumptions that comprise the conceptualisation, avoiding "hidden" or implicit assumptions. If we assume that an intelligent agent's perception of reality is mediated by abstractions in the agent's mind, or concepts, we can say the conceptualisation through which the agent perceives the world is the agent's "ontological commitment".

Borst goes a little further and defines ontology as a "formal specification of a shared conceptualisation" (BORST; BORST, 1997). Here the emphasis is made on the *formal* presentation of the specification, which implies the use of formal specification language, with a formal syntax and semantics, and on the *shared* characteristic of the conceptualisation, meaning that it represents an collective agreement on the meaning of the specified concepts. The requirement of a formal specification language is due to the intention of communicating, in a clear and precise way, the contents of the conceptualisation. Note that the use of a formal language does not imply that an ontology is a perfect specification of conceptualisation. Instead, it approximantes, within the limits of the specification language, the intended intepretations (or models) of the ontological commitment implied by the later.

Studer *et al* both Gruber's and Borst's definitions, considering an ontology "a formal, explicit specification of a shared conceptualisation". They explain that *conceptualisation* "refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon", that "explicit" means that"the type of concepts used, and the constraints on their use are explicitly defined", that "formal" means that "the ontology should be machine readable" and that "shared" implies that an ontology "captures consensual knowledge".

The concern with the formal specification of a concept can also be applied to the concept of ontology itself. In (GUARINO, 1998), Guarino describes a formal characterization of the concept of ontology and other related concepts, such as conceptualization, ontological commitment and intended models. This formal theory clarifies, among other things, the relationship between an ontology and *language* and the various degrees of precision with which a "good" or "bad" ontology specifies its intended models. The approach used by Guarino, that makes use of a possible worlds structure to characterize concepts, can also be seen in the Unified Foundational Ontology (GUIZZARDI, 2005), the focus of the present work.

2.2 Axiomatic Ontological Theories

There are many ways to describe a conceptual system. The most common choice is simply to explain concepts using natural language, an approach that leverages the large vocabulary, and corresponding shared concepts, shared by those that are proficient in a natural language, such as English. Using a natural language also has the benefit of not requiring that the involved parties learn an artificial language whose usage scope may be limited.

However, natural languages have several drawbacks as a conceptual system specification language. The most important one is that natural languages have complex syntactical rules and have imprecise and ambiguous semantics. Consequently, conceptual system specifications written in a natural language may fail to communicate the precise meaning that was intended by its author.

By using a formal language, with formal and precise syntax and semantics, the kinds of errors that may occur in the interpretation of a specification can be reduced and limited to the interpretation of its free symbols, which may be a small subset of all symbols described in it. We call such descriptions a formal specification or, if the formal language is a formal logic, we call it an axiomatic ontological theory.

In this thesis we present a large axiomatic ontological theory, written in the Isabelle/HOL language, which describes a large collection of concepts, including a subset of the concepts that form the UFO foundational ontology. Before the presentation of the theory itself, which is done in the next chapters, we provide here a simple example of a specification of a toy ontological theory, to illustrate the process by which an axiomatic ontological theory is constructed.

The idea is to describe the concept of an *Individual* and of an *Universal*, along with the *instantiation* and *subsumption* relations, whose meaning can be summarized as:

1. There exists at least one individual and one universal;
2. No individual is an universal and vice-versa;
3. Instantiation is a relation between individuals and universals;
4. Every individual instantiate at least one universal and every universal is instantiated by at least one individual;
5. Subsumption is a relation between universals;
6. An universal subsumes another just in case all individuals that instantiate the former also instantiate the later.

$$\Sigma \equiv \{\textbf{individual}, \textbf{universal}, \textbf{instantiation}, \textbf{subsumption}\}$$

$$arity(x) \equiv \begin{cases} 1 & \text{, if } x \text{ is } \textbf{individual} \text{ or } \textbf{universal}\text{, or} \\ 2 & \text{, if } x \text{ is } \textbf{instantiation} \text{ or } \textbf{subsumption}. \end{cases}$$

Figure 1 – First-Order Signature of Toy Ontology Specification

Given this description written in natural language, let's proceed by writing an specification in first order logic.

First Order Logic Specification

We start the process by identifying the concepts that are to be specified and constructing a signature, consisting in a set of formal constants with associated arities, and associating these constants with the concepts we want to specify:

- *Individual* concept: represented by the monadic symbol **individual**, i.e. the expression **individual**(x) is interpreted as x is an *Individual*;
- *Universal* concept: represented by the monadic symbol **universal**, i.e. the expression **universal**(x) is interpreted as x is an Universal;
- Instantiation relation: represented by the diadic symbol **instantiates**, i.e. the expression **instantiates**(x, y) is interpreted as x *instantiates* y;
- Subsumption relation: represented by the diadic symbol **subsumes**, i.e. the expression **subsumes**(x, y) is interpreted as x *subsumes* y.

Formally, the signature of the first-order theory consists on the set Σ of symbols and the function $arity : \Sigma \mapsto \mathbb{N}$ depicted in Figure 1.

Now we specify the meaning of these constants by writing formal axioms, in order to convey the intended meaning of their associated concepts, as depicted in Figure 2.

At this point, two questions might be raised: (1) is this theory consistent, i.e. can we present at least one example that satisfies it; and (2) is this theory minimal, i.e. are all axioms necessary or is there a proper subset of those that imply all the others. The first question is important because the consistency of a first-order theory guarantees that the theory makes (at least one) sense, ruling out the existence of logical contradictions in its axioms. The second question is also important because if we compare two theories where one is a subset of the other but which have the same models, then the first one can be considered preferable, due to being a more concise specification of the same concepts.

Axiom Name	Axiom
FO1	$(\exists x.\, \mathbf{individual}(x)) \wedge (\exists x.\, \mathbf{universal}(x))$
FO2	$\forall x.\, \neg\, (\mathbf{individual}(x) \;\wedge\; \mathbf{universal}(x))$
FO3	$\forall x.\forall y.\, \mathbf{instantiates}(x, y) \longrightarrow \mathbf{individual}(x) \;\wedge\; \mathbf{universal}(y)$
FO4.1	$\forall x.\mathbf{individual}(x) \longrightarrow \exists y.\mathbf{instantiates}(x, y)$
FO4.2	$\forall x.\mathbf{universal}(x) \longrightarrow \exists y.\mathbf{instantiates}(y, x)$
FO5	$\forall x.\forall y.\, \mathbf{subsumes}(x, y) \longrightarrow \mathbf{universal}(x) \;\wedge\; \mathbf{universal}(y)$
FO6	$\forall x.\forall y.\, \mathbf{subsumes}(x, y) \longleftrightarrow (\forall z.\, \mathbf{instantiates}(z, x) \longrightarrow \mathbf{instantiates}(z, y))$

Figure 2 – Axiomatization of toy ontology in first order logic.

We can show the consistency of the theory by providing a simple model $M = (D,\, R,\, I)$ where $D = \{John,\, Person\}$ is the domain of interpretation; $R = \{R_1, R_2, R_3, R_4\}$ is a set of relations, where $R_1 = \{John\}$, $R_2 = \{Person\}$, $R_3 = \{(John, Person)\}$ and $R_4 = \{(Person, Person)\}$; and $I \,:\, \Sigma \mapsto D \cup R$ is the interpretation function, where $I(\mathbf{individual}) = R_1$, $I(\mathbf{universal}) = R_2$, $I(\mathbf{instantiates}) = R_3$ and $I(\mathbf{subsumes}) = R_4$. We leave to the reader the verification that this models satisfies the axioms depicted in Figure 2. Regarding the minimality of the theory, we can show that the axiom FO5 follows from FO1, FO4.1, FO4.2, and FO6, and, thus, that the theory can be safely reduced by removing it, leading to a reduction in the complexity of the later.

As we can see, the tools provided by Formal Logic are an import asset in the design of axiomatic ontologies. As we shall see in the next section, the process of cosntructing theses theories and applying logical methods in their analysis can be made easier through the use of proof assistant systems.

2.3 Proof Assistant Systems

Proof assistant systems are software systems developed for assisting in the definition and proof of mathematical logic theories.

The history of machine assisted proofs started in 1967 with prof. N. G. de Brujin's Automath project (BRUIJN, 1983). The goal of the project was to design a formal language in which mathematical theories could be expressed and have their correctness verified by a computer. In 1968, an Automath language was proposed, AUT-68, and was followed by several variations, among which is the AUT-QE extension, which received most of the research attention in the period.

Several automated proof checking systems were proposed during the course of the project. These systems could check whether an Automath theory written in a computer text file was syntactically valid, with respect to the Automath language syntax, and whether the proofs, which were also terms of the language, were correct.

Among the motivations listed by de Brujin (BRUIJN, 1991) for using systems such as Automath, we can highlight the following: (1) protection against human oversight in long chains of formal arguments, particularly in cases where the formal proof is too long or too complex for human being verification in a limited time, which include computer systems formal correctness proofs; (2) to "lighten the burden of referees of mathematical papers", by providing a means for the automatic verification of formal theories published in a paper, as long as the author provides a version written in a computer verifiable language, allowing the referee to concentrate on the actual results of the paper; (3) to assist in the *understanding* of mathematics, i.e. an automated proof checker plays a role of a baseline student: if a teacher cannot convince a machine that an argument is valid, expecting the students to understand it might be an illusion; and (4) to help us achieve an absolute rigor in the formalization of theories and arguments.

The systems produced in the context of the Automath projects are considered proof-checkers but not *proof-assistants*. The distinction lies in the role played by the system in the construction of formal proofs: while, in the Automath case, the user has to write the proofs in a detailed form, without any assistance from the system, in the case of a proof-assistant system, the system itself generates the proof, or parts of the proof, with the user's guidance, which usually takes the form of scripts that refer to proof tactics.

Proof-assistants should also not be mistaken with theorem provers: in the later case, the system takes a theorem and tries to find a complete proof for it, failing otherwise, while in the former case, the user is still in control of the proof construction process. Some proof-assistants, however, make use of theorem provers to help search for proofs of intermediary goals to facilitate the proof construction.

There are many approaches in the design of proof assistant languages: the type-theoretical approach, taken by proof-assistants that were inspired in Automath, such as Nuprl (CONSTABLE et al., 1986) and Coq (BERTOT et al., 2004), is based on the Curry-Howard correspondence, i.e. theorems are represented by *types* in a type system and proofs are encoded as functions that satisfies those types; in the LCF (*Logic* of *Computable Functions*) approach, taken by the HOL (GORDON; MELHAM, 1993), HOL-Light (HARRISON, 2016) and Isabelle (NIPKOW; KLEIN, 2014) proof assistants, among others, represents theorems as values of an abstract data type in the language, whose values can only be constructed by using abstract operations that represent valid inference rules, i.e. theorems are values (not types) and proofs are "by construction".

2.3.1 Isabelle

The formal theories presented in this thesis were developed using the Isabelle proof-assistant, a framework that follows the LCF approach. Isabelle supports the construction and verification of theories written in various formal languages built upon the Isabelle/Pure

core logic, of which the most mature and most popular is the Isabelle/HOL language.

Isabelle/HOL is a higher-order logic based on typed lambda calculus with parametric polymorphism. As a functional language, some expressions in Isabelle/HOL denote functions, or higher order functions. In particular, logical propositions are represented by functions that have the *bool* type, while first-order n-ary predicates are represented by functions whose types such as $T_1 \to \cdots \to T_n \to \mathbf{bool}$, where T_i, for $1 \leq i \leq n$ is the type of the i-th predicate's parameter.

The availability of higher-order predicates allow the direct representation of expressions which, in a first-order context, would require the use of meta-linguistic constructions, such as logical schemes. In particular, all the logical connectives and quantifiers are represented as typed values. For example, the universal quantifier $\forall$ is represented as a higher-order predicate with type $(\alpha \to bool) \to bool$, i.e. it takes a monadic predicate whose parameter has type α and results in a proposition (a *bool* value). Bounded quantifiers, such as $\exists x \in X.P$, are also easily representable in HOL, e.g. the bounded existential has type $\alpha\ set \to (\alpha \to bool) \to bool$: given an α-set, the domain of quantification, and a monadic predicate whose parameter has type α, the bounded quantifier results in a predicate.

The flexibility of the Isabelle/HOL's type system is complemented by the flexibility provided by possibility of using mix-fix notation in its syntax, a feature that enables the construction of specialized syntax constructions that approximate their conventional usage. In particular, first-order logic expressions look very similar in Isabelle/HOL, with slight difference in how function application is expressed, since Isabelle/HOL's follows the lambda-calculus convention where the application of a function f to parameters $p_1 \cdots p_n$ if written as "$f\ p_1\ p_2\ \ldots\ p_n$", instead of "$f(p_1, p_2, \ldots, p_n)$".

An Isabelle/HOL theory is encoded in theory file, which includes a name that is used to identify and refer to the theory, a list of *imported* theories, whose definitions and proofs are included in the actual context, and a sequence of Isabelle/HOL statements, which may introduce free symbols, axioms, defined symbols, inductive and co-inductive data types, recursive and co-recursive definitions, and lemmas, among others.

Some Isabelle/HOL statements create proof obligations, which have to be satisfied by the inclusion of a respective proof. The obvious case is the lemma declaration statement, which has to be followed by the proof of the same. Other statements also require proofs, such as recursive function definitions, which require proof of totality and well-foundness. However, in many cases a proof obligation can be discharged using automated proof methods, freeing the theory author from having to tediously prove simpler logical propositions.

For an in-depth introduction to the Isabelle/HOL language and the Isabelle/Isar proof language, the reader is referred to [?].

2.4 Conceptual Modeling and OntoUML

The two core concepts in Computer Science are the computer and the computer program, both of which are studied through several abstract models that represent computations in different but ultimately equivalent perspectives, such as recursive functions, post systems, lambda calculus and Turing machines. In all these models computer programs are considered as no more than an abstract symbolic function that, given some input in the form of symbols, generate its output again in symbolic form.

Nevertheless, computers and computer programs are the main components of an information system, an artifact that whose purpose is to help human beings in the storage, processing and retrieval of information regarding entities that may be very concrete, such as real persons and real objects.

The fact that information systems can be used effectively for such purposes indicates that they are somewhat more than just computer programs, since they also carry a series of assumptions regarding the relation of its abstract symbols to the entities that it refers to and that exist outside the abstract models of computation themselves.

2.5 Category Theory

This work uses concepts of the theory of categories (MACLANE, 2013) and, more specifically, of the theory of concrete categories (ADAMEK; HERRLICH; STRECKER, 2009).

3 Simplified UFO Theory

3.1 Introduction

This chapter presents a formal specification in Isabelle/HOL of a fragment of the UFO-A ontology. The chapter is structured as a series of sections that introduce the sub-theories that comprise the whole theory of the UFO-A fragment, that are presented in a order that respect the dependency between the theories, i.e. the theories at some section do not depend on theories presented on later sections.

Following this order, the sections (and theories) are presented as follows: first, we present the formal characterization of the structure of possible worlds that is used in UFO in the section 3.2, where each possible world is taken to be the set of particulars that are considered to exist in it, and we also mention the distinction between endurant particulars and event particulars, though we skip the formal characterization of the notion of Events due to our focus on the endurant fragment of UFO (UFO-A); we then introduce the relation of inherence in section 3.3, dividing the set of endurants into the set of substantials and the set of moments; afterwards, we introduce the notion of quality spaces in section 3.4 and use it to define the notion of qualified moments in section 3.5, that is a simpler variation of the notion of Qualities presented in the original work; we then present the notion of *towardness* in section 3.6 as a basic representation of binary material relations that, though being strictly less expressive than the notion of Relators presented in the original work, is nevertheless enough to express relational distinctions between particulars that is sufficient for the analysis of the relational aspects of identity; in section 3.7 we bring together the theories proposed in the previous sections into a final theory of UFO particulars; finally, in section 3.8 we present a minimal theory of instantiation and of UFO universals[1].

3.2 Particulars and Possible Worlds [2]

The most basic notion in UFO, and in the UFO derived ontology described in this thesis, is the concept of a *particular* as an entity, or being, that enjoy the property of existing, either in actuality or, at least, in potential.

Given a domain of discourse that delineates a scope particulars, any maximal configuration of its elements that is considered possible, in the sense that it is conceivable that the reality they represent could possibly obtain, is called a *possible world*.

[1]The full theory of UFO universals based on the framework presented in chapter 4 is described in section 6.4

[2]The full Isabelle/HOL code for this section can be found in section B.1.

As was explained in chapter 2, the representation of an ontological commitment requires a suitable representation of the possible configurations of the reality towards which the commitment is made. Thus, the notion of a possible world and of the delineation of possibilities given by the set of all possible worlds is the basis of the ontological theory provided in this chapter.

This section presents the structure of particulars and possible worlds that is used throughout the thesis. The structure is initially presented in its most basic form, lacking elements that can be used to characterize the particulars and describe their inter-relationships. These elements shall be introduced gradually on the next sections.

The only information given by this structure is the composition of the set of particulars, and their potential co-existence, i.e. two or more particulars are deemed possibly co-existants just in case there is at least one possible world which contain all of them. As such, the representation of a possible world is simply a set of particulars:

type_synonym *'p world =* ‹*'p set*›

The theory of possible worlds described in this section has the following signature:

Type	*Denotation*
'p	*particulars*
'p world	*possible worlds (sets of particulars)*
'p world set	*sets of possible worlds*

Symbol	*Type*	*Denotation*
$\mathcal{W}$	*'p set set*	*set of possible worlds*

The set of *particulars*, written as $\mathcal{P}$, consists in the set of all particulars that compose all possible worlds:

definition $\mathcal{P}$ *::* ‹*'p set*›
  where ‹$\mathcal{P} \equiv \bigcup \mathcal{W}$›

Given two particulars, *x* and *y*, we say that *x* is *existentially dependent upon* *y*, written as *ed x y*, whenever the existence of the former implies the existence of the later:

definition
  ed :: ‹*'p* $\Rightarrow$ *'p* $\Rightarrow$ *bool*›
 where
  ‹*ed x y* $\equiv$ $x \in \mathcal{P} \wedge y \in \mathcal{P} \wedge$
    $(\forall w \in \mathcal{W}.\ x \in w \longrightarrow y \in w)$›

Given two particulars, *x* and *y*, we say that *x* and *y* are *existentially interdependent*, written as *interdep x y*, if they are existentially dependent upon each other:

definition
```
interdep :: ‹'p ⇒ 'p ⇒ bool›
```
where
```
‹interdep x y ≡ ed x y ∧ ed y x›
```

Given two particulars, `x` and `y`, we say that `x` and `y` are *existentially independent*, written as `indep x y`, if neither the existence of `x` nor the existence of `y` imply the existence of the other:

definition
```
indep :: ‹'p ⇒ 'p ⇒ bool›
```
where
```
‹indep x y ≡ x ∈ 𝒫 ∧ y ∈ 𝒫
            ∧ ¬ ed x y ∧ ¬ ed y x›
```

The set of possible worlds is constrained by the following axioms:

Axiom 1 (`at_least_one_possible_world`)**.** *Since at least one possible configuration is deemed possible, e.g. that of actuality, at least one possible world must exist, even if its empty, i.e. even if no particulars exist in actuality:*

$$\mathcal{W} \neq \emptyset$$

Axiom 2 (`particulars_do_not_exist_in_some_world`)**.** *No particular is deemed necessary, or, in other words, every particular's existence is* contigent*. This restriction was added to exclude entities that could be considered to exist by definition, such as natural numbers, logical formulas, etc., since such entities are better described by specific axiomatic theories instead of by an ontological theory that carry the added structure of possible existence and non-existence:*

$$x \in \mathcal{P} \Longrightarrow \exists\, w{\in}\mathcal{W}.\ x \notin w$$

Axiom 3 (`injection_to_ZF_exist`)**.** *The domain from which particulars are drawn is not "too big", in the sense that it can be represented by [glos:Zermelo-Fraenkel] sets through at least one injective function. This constraint is required to circumvent some limitations of Isabelle/HOL type system regarding proofs that exhibit type polymorphism:*

$$\exists\, f :: 'p \Rightarrow ZF.\ inj\ f$$

The full Isabelle/HOL source for this theory describes various lemmas that can be derived from these axioms and definitions. We include here some of the most relevant, omitting the respective proofs, which can be found in **??**.

The following lemmas follow exclusively from the definition of the respective relations:

Lemma 1 (`ed_refl`, `ed_trans`). *Existential dependency is a [glos:pre-order], i.e it is a reflexive and transitive relation:*

$$\texttt{x} \in \mathcal{P} \Longrightarrow \texttt{ed x x}$$
$$[\![\texttt{ed x y; ed y z}]\!] \Longrightarrow \texttt{ed x z}$$

Lemma 2 (`interdep_refl`, `interdep_symm`, `interdep_trans`). *Existential interdependency is an [glos:equivalence-relation], i.e. it is reflexive, symmetric and transitive:*

$$\texttt{x} \in \mathcal{P} \Longrightarrow \texttt{interdep x x}$$
$$\texttt{interdep x y} \Longrightarrow \texttt{interdep y x}$$
$$[\![\texttt{interdep x y; interdep y z}]\!] \Longrightarrow \texttt{interdep x z}$$

Lemma 3 (`indep_sym`). *Existential independency is a symmetric relation:*

$$\texttt{indep x y} \Longrightarrow \texttt{indep y x}$$

From the axioms, we can derive the following lemmas:

Lemma 4 (`non_empty_particulars_at_least_two_worlds`). *If there is at least one particular, then there must be at least two distinct possible worlds:*

$$\mathcal{P} \neq \emptyset \Longrightarrow \exists\, \texttt{w}_1 \in \mathcal{W}.\ \exists\, \texttt{w}_2 \in \mathcal{W}.\ \texttt{w}_1 \neq \texttt{w}_2$$

Lemma 5 (`no_empty_worlds_imp_two_particulars`). *If no empty possible worlds exist, i.e. it isn't possible for a reality void of existing things to obtain, then there are at least two distinct particulars:*

$$\emptyset \notin \mathcal{W} \Longrightarrow \exists\, \texttt{x}_1 \in \mathcal{P}.\ \exists\, \texttt{x}_2 \in \mathcal{P}.\ \texttt{x}_1 \neq \texttt{x}_2$$

3.2.1 Modes of Existence

The first distinction that UFO makes between particulars is one based on their [glos: modes of existence]: a particular is either an *endurant*, i.e. a being that exists as a complete entity in every instant of time in which it is considered present, or an *event*, also called a *perdurant*, i.e. a being that is observable as a whole only when the whole time interval of its existence is considered.

Since this thesis focuses on endurants, we exclude the set of events from the set of particulars, by considering the later a synonym for the set of endurants:

```
abbreviation (in possible_worlds_sig)
  endurants (‹ℰ›) where ‹ℰ ≡ 𝒫›
```

In the following sections, the structure of possible worlds is enriched with elements that allow a minimal representation of qualities of particulars and of relations between them.

3.3 Substantials, Moments and Inherence [3]

In this section, we present a theory about UFO's *inherence* relation and the two categories of UFO endurants that are distinguished through it: *Substantials* and *Moments*. We present several candidate theories for the inherence relation, beginning with one that is based on the axioms from the UFO's original work (GUIZZARDI, 2005). We identify some undesirable properties, or anomalies, that are derivable from the original axiomatization and, for each anomaly, we propose additional axioms that exclude it. Afterwards, we prove that those additional axioms and a few of the original ones are logically equivalent to the requirement that the inherence relation must be a noetherian relation. We end this section by proposing the theory of inherence which shall be used in this thesis.

The basic signature of the inherence relation theories presented in this section extends the signature of the `possible_worlds` theory with a single symbol that denotes the inherence relation:

Symbol	*Type*	*Denotation*
(◁)	'p ⇒ 'p ⇒ bool	inherence relation (inheresIn), where x ◁ y denotes that x inheres in y

[3] The full Isabelle/HOL code for this section can be found in section B.2.

The inherence relation is the link between an endurant and its particularized properties. For example, when we say that "John's eye is blue", it is represented here as the existence of two endurants, "John's eye" and "John's eye being blue", where the later, a particularized property, is said to "inhere" in the former. Conversely, the former is said to "bear" the later, or to be the later's *bearer*.

In other words, any endurant may *bear* particularized properties that characterize it, being themselves also endurants. Those endurants that represent particularized properties are called *moments*, while those that do not are called *substantials*.

The inherence relation is restricted by the following axioms that, when added to the `possible_worlds` theory, form what we call the `inherence_base` theory, a minimal set of axioms for all the inherence theories presented in this section:

Axiom 4 (`inherence_scope`[4]). *The inherence relation is restricted to* endurants *only:*

$$x \triangleleft y \Longrightarrow x \in \mathcal{E} \wedge y \in \mathcal{E}$$

Axiom 5 (`inherence_imp_ed`[5]). *The existence of a moment must, of course, imply the existence of its bearer:*

$$x \triangleleft y \Longrightarrow ed\ x\ y$$

Axiom 6 (`moment_non_migration`[6]). *Moments inhere in a single bearer:*

$$[\![x \triangleleft y;\ x \triangleleft z]\!] \Longrightarrow y = z$$

Using the inherence relation we can define the sets of moments and substantials, that are represented, respectively, by the symbols $\mathcal{M}$[7] and $\mathcal{S}$[8]:

definition $\mathcal{M}$ **where** ‹$\mathcal{M} \equiv \{\ x\ .\ \exists y.\ x \triangleleft y\ \}$›

definition $\mathcal{S}$ **where** ‹$\mathcal{S} \equiv \mathcal{E}$ - $\mathcal{M}$›

The *bearer* of a moment, i.e. the endurant it inheres in, is given by the function β (`bearer`), defined as:

[4](GUIZZARDI, 2005), pg. 213, num. 3.
[5](GUIZZARDI, 2005), pg. 213, num. 4.
[6](GUIZZARDI, 2005), pg. 214, num. 9.
[7](GUIZZARDI, 2005), pg. 214, num. 8.
[8](GUIZZARDI, 2005), pg. 215, num. 11.

definition *bearer* :: ‹'p ⇒ 'p› (‹β›)
  where ‹β m ≡ THE x. m ◁ x›

Since the transitive closure of the inherence relation shall play a role in some proofs, we introduce here a special syntax that denotes it:

abbreviation
  inheres_in_trancl :: ‹'p ⇒ 'p ⇒ bool›
  (**infix** ‹◁◁› 75)
    where ‹x ◁◁ y ≡ $(◁)^{++}$ x y›

Similarly, we also represent the n-th transitive iteration of the inherence relation using a special syntax:

abbreviation
  inheres_in_by :: ‹'p ⇒ nat ⇒ 'p ⇒ bool›
  (‹_ ◁◁- _› [74,1,74] 75)
    where ‹x $◁◁^{n}$ y ≡ ((◁)^^n) x y›

The n-th transitive iteration is defined for every natural number *n*. In special, the 0-th iteration is simply the identity relation. Other examples include the 1-th iteration, which is simply the inherence relation itself, and the 2-th iteration, $x \lhd\lhd^{2} y$, which is equivalent to $\exists z.\ x \lhd z \wedge z \lhd y$. The *n-th bearer* of a moment is defined as the endurant to which is linked by a n-th iteration of the inherence relation:

definition
  nth_bearer :: ‹'p ⇒ nat ⇒ 'p› (‹#β›)
    where ‹#β m n ≡ THE x. m $◁◁^{n}$ x›

In the context of a *inherence_base* theory, the following lemmas hold:

The bearer function is well defined for moments:

lemma *bearer_eqI:*
  assumes ‹x ◁ y›
  shows ‹β x = y›

Every moment inhere in the endurant given by the bearer function:

lemma *bearerI:*
  assumes ‹x ∈ $\mathcal{M}$›
  shows ‹x ◁ β x›

Nth-bearers, when they exist, are also unique:

lemma *nth_inherence_unique_cond:*
assumes ‹$x \lhd\lhd^n y$› ‹$x \lhd\lhd^n z$›
shows ‹$y = z$›

Under the same condition, nth-bearers coincide, if they exist, with the nth-iteration of the bearer function:

lemma *nth_bearer_eq_nth_iteration_of_bearer:*
assumes
nth_bearer_exists: ‹$\exists y.\ x \lhd\lhd^n y$›
shows ‹$\#\beta\ x\ n = (\beta\ \hat{}\hat{}\ n)\ x$›

There are no separate bearer chains, i.e. if y_1 and y_2 are indirect bearers of the same moment x, then one of the must be an indirect bearer of the other, depending on how many inherence "steps" they are from x:

lemma *inherence_mid_point[intro]:*
‹$y_1 \lhd\lhd^{(n-m)} y_2$›
if *assms:* ‹$x \lhd\lhd^m y_1$› ‹$x \lhd\lhd^n y_2$› ‹$m \leq n$›

From the idea that moment may inhere in other moments, which themselves can be other moments, comes the notion of the *order of a moment*: a first-order moment is a moment that inheres in a substantial, a second-order moment is a moment that inheres in a first-order moment, etc.

Since a moment has an order of, at minimum, one, we can add substantials as a special case, saying that they have order 0.

Endurants can form chains linked by the inherence relation. For a moment m, a substantial that limits its inherence chain, if it exists, is called an ultimate bearer of m.

Formally, we define these notions as:

If x is a moment, then an endurant y is considered an *ultimate bearer* of x if and only if x is either equal to y or x inheres transitively in y, and y is a substantial:

definition
is_an_ultimate_bearer_of
:: ‹$'p \Rightarrow 'p \Rightarrow bool$›
where
‹$is_an_ultimate_bearer_of\ x\ y \longleftrightarrow$
$(\lhd)^{**}\ y\ x \wedge x \in \mathcal{S}$›

If x inheres in y by n steps, i.e. $x \lhd\lhd^n y$, and y is an ultimate bearer of x, then we say that x has order n. Otherwise, if x is a substantial, we say that x has order 0:

definition
  has_order
  :: ‹*'p* ⇒ *nat* ⇒ *bool*›
  where
    ‹*has_order x n* ⟷
      (∃*y*. *x* $\lhd\lhd^n$ *y* ∧
          is_an_ultimate_bearer_of y x)›

In the context of a *inherence_base* theory, we can prove the following lemmas:

Lemma 6 (*ultimate_bearer_unique*)**.** *Ultimate bearers, whenever they exist, are unique:*

⟦*x* ∈ $\mathcal{E}$; *is_an_ultimate_bearer_of y x*; *is_an_ultimate_bearer_of z x*⟧ ⟹ *y* = *z*

Lemma 7 ($thm_nameultimate_bearer_unique_order$)**.** *If an endurant has an ultimate bearer, then it has a unique order:*

⟦*x* ∈ $\mathcal{E}$; *is_an_ultimate_bearer_of y x*⟧ ⟹ ∃!*n*. *has_order x n*

It is reasonable to expect, in UFO's conceptualization of inherence, that every moment has a determinate order and a unique ultimate bearer. A moment cannot be simultaneously, a first-order moment and a second-order moment. Nor should it have more than one ultimate bearer, since a moment represents the particularization of a property of *one* individual. This uniqueness is guaranteed as long as every moment have an ultimate bearer, as shown in lemmas *ultimate_bearer_unique* and *ultimate_bearer_unique_order*.

In order to show that UFO's axiomatization of inherence guarantee the well-definedness of these concepts, it is necessary and sufficient, then, to show that every moment has an ultimate every ultimate bearer It is sufficient to prove, then, that every moment has an ultimate bearer

To complete the axiomatization of the inherence relation with respect to the original theory presented in (GUIZZARDI, 2005), we need to add a few more axioms to the *inherence_base* theory, composing what we call the *inherence_original* theory:

Axiom 7 (*inherence_irrefl*[9])**.** *The inherence relation is an irreflexive relation:*

[9](GUIZZARDI, 2005), pg. 213, num. 5.

$\neg\ x \triangleleft x$

Axiom 8 (`inherence_asymm`[10]). *The inherence relation is an asymmetric relation:*

$x \triangleleft y \Longrightarrow \neg\ y \triangleleft x$

Axiom 9 (`inherence_intransitive`[11]). *The inherence relation is intransitive:*

$[\![x \triangleleft y;\ y \triangleleft z]\!] \Longrightarrow \neg\ x \triangleleft z$

It is reasonable to expect that any chain $m_1 \triangleleft m_2 \triangleleft m_3 \triangleleft \cdots$ of moments will end up in a substantial, an ultimate bearer for the moments $m_1, m_2, \ldots$. Furthermore, only one such substantial should exist for each moment, since a moment expresses, directly, or indirectly, a property of a single substantial. Under these assumptions, it is possible to refer to *the ultimate bearer of* a moment in a determinate way.

Similarly, a moment should have a determinate *order*. Is it a first-order moment, a property of a substantial, or a second-order moment, a property of a property of a substantial, or a third-order moment, etc. If a moment has only one order, then it makes sense to speak of *the order of* a moment.

Formally, we can define these concepts through the following functions:

```
definition order :: ‹'p ⇒ nat›
  where ‹order m ≡ THE n. has_order m n›

definition
    ultimateBearer
    :: ‹'p ⇒ 'p› (‹!β›)
  where
    ‹!β m ≡ THE x. is_an_ultimate_bearer_of x m›
```

However, even though the existence of a unique ultimate bearer and of a definite order for each moment are expected properties of the theory, they do not necessarily hold in the axiomatization provided in the original UFO work.

[10] (GUIZZARDI, 2005), pg. 213, num. 6.
[11] (GUIZZARDI, 2005), pg. 214, num. 7.

3.3.1 No Ultimate Bearer Anomaly

As explained in the previous section, in any reasonable formal theory of UFO moments and substantials, it is expected that every moment has an ultimate bearer. The existence of a moment for which no ultimate bearer exists can be considered an anomaly. This anomaly can be represented formally through the following logical proposition:

definition
‹*has_moment_without_ultimate_bearer* $\longleftrightarrow$
(∃ *m* ∈ $\mathcal{M}$. ∀ *y*. ¬ *is_an_ultimate_bearer_of y m*)›

A theory in which this proposition holds is said to be a theory that has the "no ultimate bearer" anomaly. Considering the axioms introduced in the *inherence_base* locale as a minimum set of axioms for an inherence theory, the following lemma describes a way to prove that the theory has this property:

lemma (**in** *inherence_base*)
has_moment_without_ultimate_bearerI[intro]:
assumes ‹*m* ∈ $\mathcal{M}$› ‹$\bigwedge n.\ \exists x.\ m \lhd\lhd^{n}\ x$›
shows ‹*has_moment_without_ultimate_bearer*›

3.3.2 Cyclic Inherence Anomaly

Another situation we do not expect to occur in a reasonable theory of UFO moments and substantials is one in which a moment inheres directly, or indirectly, into itself. Such situation would be as nonsensical as saying that there is property which is a property of itself, or a set that is member of itself.

To define this anomaly formally, we first define the notion of a *cyclic relation*, were a relation here is taken to be a binary predicate. A binary predicate *R* is (or represents) a *cyclic relation* if and only if there is some element that is related to itself by the transitive closure of *R*. Formally:

definition ‹*cyclic R* ≡ ∃ *x*. R^{++} *x x*›

It is easy to see that if the inherence relation is cyclic, then the theory has the "no ultimate bearer" anomaly, since any element related to itself by the transitive closure of the inherence relation would have at least one n-th level bearer for each natural number $n > 0$, as the following lemma proves:

lemma (**in** *inherence_base*)
cyclic_inherence_has_moment_witout_ultimate_bearer[intro!,simp]:

```
‹cyclic (◁) ⟹ has_moment_without_ultimate_bearer›
```

The axioms described in the *inherence_original* and *inherence_base* locales, which corresponds to the axioms introduced in the original UFO theory, do not exclude models in which the inherence is cyclic, as the following lemma shows:

```
lemma inherence_original_allows_cycles:
  ‹∃ (𝒲 :: nat set set) inheresIn.
     inherence_original 𝒲 inheresIn ∧
     cyclic inheresIn›
```

As a consequence, the "ultimate bearer" anomaly is present in the original theory:

```
lemma inherence_original_has_ultimate_bearer_problem:
  ‹∃ (𝒲 :: nat set set) inheresIn.
     inherence_original 𝒲 inheresIn ∧
     inherence_sig.has_moment_without_ultimate_bearer
          𝒲 inheresIn›
```

We can eliminate cycles from the theory explicitly by including the following axiom:

Axiom 10 (*no_inherence_cycles*)**.** *No endurant must inhere, directly or indirectly, into itself:*

```
¬ x ◁◁ x
```

We call *inherence_V2* the locale consisting on the axioms of the original theory (*inherence_original*) plus this axiom.

3.3.3 Infinite Inherence Chain Anomaly

Another peculiar situation that may appear in an inherence theory is one in which there is an infinite ascending chain of inhering endurants, e.g. there is an infinite sequence $\{e_1, \ldots\}$ of endurants where, for every natural number $i > 0$, e_i inheres in e_{i+1}. Formally:

```
definition (in inherence_sig)
    inf_inh_asc_chain :: ‹'p set ⇒ 'p ⇒ bool›
  where
    ‹inf_inh_asc_chain X x ≡
      infinite X ∧ (∀y. y ∈ X ⟷ (◁)** x y)›
```

The existence of an infinite ascending inherence chain is represented by the following logical proposition:

definition *(in inherence_sig)*
  ‹*has_inf_inh_asc_chain* ≡
    ∃ *X x. inf_inh_asc_chain X x*›

We use the notation Δ^{β} *m s* to represent the order of a moment *m* (if it is unique) with respect to *s*, where *s* is one of its indirect bearers. Formally:

definition *(in inherence_sig)*
  bearer_order :: ‹*'p* ⇒ *'p* ⇒ *nat*› (‹Δ^{β}›)
  where ‹Δ^{β} *m s* ≡ *THE n. m* $\lhd\lhd^{n}$ *s*›

We use the notation *β* m* to represent the set of direct or indirect bearers of the moment *m*:

definition *(in inherence_sig)*
  bearers :: ‹*'p* ⇒ *'p set*› (‹*β**›)
  where ‹*β* m* ≡ { *s . m* ◁◁ *s* }›

The following lemma shows that, under the axioms of the *inherence_V2* locale, if a moment *x* inheres directly or indirectly into some *y*, it does so by a unique number of inherence steps:

lemma *(in inherence_V2) no_cycles:*
  assumes ‹*x* $\lhd\lhd^{n}$ *y*› ‹*x* $\lhd\lhd^{n'}$ *y*›
  shows ‹*n = n'*›

Consequently, as the following lemma proves, in the context of the *inherence_V2* locale, for any direct or indirect bearer of a moment *m* there is a definite number of steps by which *m* inheres in *s*:

lemma *(in inherence_V2) bearer_order_ex1:*
  assumes ‹*s* ∈ *β* m*›
  shows ‹∃ *!n. m* $\lhd\lhd^{n}$ *s*›

Thus, in *inherence_V2*, the notion of the order of a moment with respect to of its direct or indirect bearers is well defined an presents the following properties:

lemma *bearer_order_prop:*
  assumes ‹*s* ∈ *β* m*›
  shows ‹*m* $\lhd\lhd^{\Delta^{\beta}\ m\ s}$ *s*›

```
lemma bearer_order_eq:
  assumes ‹m ◁◁^n s›
  shows ‹Δ^β m s = n›
```

The expression Δ^β `x :: 'a ⇒ nat` is a function that associates direct or indirect bearers of `x` to its order. In `inherence_V2`, this function is injective, i.e. no distinct direct or indirect bearers of some moment `x` have the same order:

```
lemma bearer_order_inj: ‹inj_on (Δ^β x) (β* x)›
```

Leaving the revised locale `inherence_V2` and the notion of the order of a moment aside for a moment, we can show that in any theory that includes the basic axioms of the `inherence_base` locale, the existence of an infinite ascending inherence chain implies the presence of the "ultimate bearer" anomaly:

```
lemma (in inherence_base)
  infinite_inherence_chain_imp_has_moment_without_ultimate_bearer:
  assumes ‹has_inf_inh_asc_chain›
  shows ‹has_moment_without_ultimate_bearer›
```

By assumption there is an infinite ascending inherence chain `X` starting on some `x`:

```
  obtain X x where A: ‹inf_inh_asc_chain X x›
    using assms
    by (auto simp: has_inf_inh_asc_chain_def)
  obtain ‹infinite X›
         ‹∀y. y ∈ X ⟷ (◁)** x y›
    using A inf_inh_asc_chain_def
          that
    by simp
  then have B: ‹y ∈ X ⟷ (◁)** x y› for y
    by blast
  then have x_in[simp]: ‹x ∈ X› by simp
  have ‹X = {x} ∪ β* x›
```

As a consequence, the set of direct or indirect bearers of `x` is infinite:

```
  then have ‹infinite (β* x)›
    using ‹infinite X› by blast
```

Thus, the set of direct or indirect bearers of `x` forms a sequence indexed by the bearer order:

```
have bearers_eq:
  ‹β* x = { #β x i | i . 0 < i ∧ i ≤ n }›
  if ‹x ◁◁^n y› ‹y ∈ 𝒮› for n y
```

We can also infer that this set is finite if one of the (direct or indirect) bearers of `x` is a substantial:

```
have finite_bearers: ‹finite (β* x)›
  if as: ‹x ◁◁ y› ‹y ∈ 𝒮› for y
proof -
  obtain n where ‹x ◁◁^n y›
    using as(1) tranclp_power
    by metis
  then have
    ‹β* x = { #β x i | i . 0 < i ∧ i ≤ n }›
    using as bearers_eq by simp
  then show ‹?thesis› by simp
qed
```

But, since the set of direct or indirect bearers of `x` is infinite, then all of its members must be moments, and since `x` inheres directly one at least one of those, `x` must also be a moment:

```
have ‹x ∈ ℳ›
proof -
  have ‹X - {x} ≠ ∅›
    using ‹infinite X›
    by (metis finite.emptyI infinite_remove)
  then obtain y where ‹y ∈ X - {x}›
    by blast
  then obtain ‹y ∈ X› ‹y ≠ x›
    by blast
  then have ‹x ◁◁ y›
    using ‹y ∈ X›
    by (metis B rtranclpD)
  then show ‹x ∈ ℳ›
    using trans_inheres_in_scope
    by blast
qed
```

Finally, we can prove that under the assumptions of this lemma, there is at least one moment (`x`) for which no ultimate bearer exists:

```
show ‹has_moment_without_ultimate_bearer›
```

```
  proof
    (simp add:
        has_moment_without_ultimate_bearer_def
        is_an_ultimate_bearer_of_def ;
        intro bexI[of _ ‹x›] ‹x ∈ 𝓜› allI impI notI)
    fix y
    assume ‹(◁)** x y› ‹y ∈ 𝓢›
    then consider ‹x = y› | ‹x ◁◁ y›
      by (meson rtranclpD)
    then consider ‹y ∈ 𝓜› | ‹x ◁◁ y› ‹y ∈ 𝓢›
      using trans_inheres_in_scope ‹x ∈ 𝓜›
      by blast
    then have ‹y ∈ 𝓜›
    proof (cases ; simp)
      assume ‹x ◁◁ y› ‹y ∈ 𝓢›
      then have ‹finite (β* x)›
        using finite_bearers by simp
      then have ‹False›
        using ‹infinite (β* x)› by simp
      then show ‹y ∈ 𝓜› by simp
    qed
    then show False
      using ‹y ∈ 𝓢› 𝓢_def by simp
  qed
qed
```

Using the previous lemmas, we can prove that the revised locale *inherence_V2* allows infinite ascending inherence chains to occur:

```
lemma inherence_V2_allows_infinite_inherence_chain:
  ‹∃ (𝓦 :: nat set set) inheresIn.
        inherence_V2 𝓦 inheresIn
      ∧ inherence_sig.has_inf_inh_asc_chain inheresIn›
```

Thus, the revised locale *inherence_V2* still has the "no ultimate bearer" anomaly, despite excluding cycles in the inherence relation:

```
lemma inherence_V2_has_ultimate_bearer_problem:
  ‹∃ (𝓦 :: nat set set) inheresIn.
        inherence_V2 𝓦 inheresIn
      ∧ inherence_sig.has_moment_without_ultimate_bearer
        𝓦 inheresIn›
```

Again, we can revise the *inherence_V2* locale by adding a axiom that excludes the existence of infinite ascending inherence chains, producing the locale *inherence_V3*:

Axiom 11 (*no_infinite_inherence_chains*).

¬ *inf_inh_asc_chain X x*

With the exclusion of infinite ascending inherence chains, the "no ultimate bearer" anomaly ceases to be present, as shown in the following lemma:

lemma *no_ultimate_bearer_problem:*
‹¬ *has_moment_without_ultimate_bearer*›

In fact, under the axioms of the *inherence_base* locale, the non-existence of a moment without an ultimate bearer is logically equivalent to the inherence being acyclic and not existing infinite ascending inherence chains:

lemma (**in** *inherence_base*) *no_ultimate_bearer_conditions:*
‹¬ *has_moment_without_ultimate_bearer* ⟷
¬ *cyclic* (◁) ∧ ¬ *has_inf_inh_asc_chain*›
(**is** ‹¬ *?A* ⟷ ¬ *?B* ∧ ¬ *?C*›)

As the following lemmas prove, under the axioms of the *inherence_base* locale, the non-existence of moments without ultimate bearers is also logically equivalent to the inherence relation being *noetherian*, i.e. if the converse of the inherence relation is well-founded. Before we present the proof, we need to define to set of suborders of an endurant. For any endurant *x*, the set of suborders of *x*, written as *suborders x*, is a set of natural numbers such that any member *n* is the order of *x* with respect to one of its direct or indirect bearers. Formally:

definition *suborders* :: ‹'p ⇒ nat set›
where
‹*suborders x* ≡ { *n* | *n y* . *x* $◁◁^n$ *y* }›

In the context of the *inherence_base* locale, for any endurant *x*, *x* has an ultimate bearer just in case its set of suborders is finite:

lemma *has_ultimate_bearer_iff_suborders_finite:*
assumes ‹*x* ∈ $\mathcal{M}$›
shows ‹(∃*y*. *is_an_ultimate_bearer_of y x*) ⟷
finite (*suborders x*)›

Thus, in the same context, the non-existence of moments without ultimate bearers is logically equivalent to the well-foundness (*wfP*) of the converse of the inherence relation ($(\triangleleft)^{-1-1}$), or, in other words, to the inherence relation being noetherian:

lemma *no_ultimate_bearer_problem_eq_noetherian:*
shows ‹$\neg$ *has_moment_without_ultimate_bearer* $\longleftrightarrow$
wfP $(\triangleleft)^{-1-1}$›

3.3.4 Finite-Chain Inherence

Since under the axioms of the *inherence_base* locale, non-existence of the "no ultimate bearer" anomaly is equivalent to the inherence relation being noetherian, let's consider an inherence theory, called *noetherian_inherence* that simply adds the following axiom to *inherence_base*:

Axiom 12 (*inherence_is_noetherian*)**.** *The inherence relation should be noetherian, which means that its converse should be well-founded:*

$$wfP\ (\triangleleft)^{-1-1}$$

This means that for any non-empty set Q *of particulars there is at least one member* z *of* Q *such that if* z *inheres in some endurant* y *then* y *is not in* Q*:*

$$x \in Q \Longrightarrow \exists z{\in}Q.\ \forall y.\ z \triangleleft y \longrightarrow y \notin Q$$

Under this axiom, the inherence relation is acyclic:

lemma *inherence_is_acyclic[intro!]:*
‹*acyclicP* $(\triangleleft)$›

This axiom also implies the axioms stated in the original theory, such as:

- Inherence is irreflexive (axiom 5 on page 213, (GUIZZARDI, 2005)):

  lemma *inherence_is_irrefl:* ‹$\neg$ $x \triangleleft x$›

- Inherence is asymmetric (axiom 6 on page 213, (GUIZZARDI, 2005)):

  lemma *inherence_is_asymm:*

  ‹$x \triangleleft y \Longrightarrow \neg\ y \triangleleft x$›

- Inherence is intransitive (axiom 7 on page 214, (GUIZZARDI, 2005)):

  lemma *inherence_intransitive_V1:*

  ‹$[\![\ x \triangleleft y\ ;\ y \triangleleft z\ ]\!] \Longrightarrow \neg\ x \triangleleft z$›

Under this axiom, all the expected properties of the inherence relation hold:

- Every moment has an unique order with respect to each of its bearers:

- Every moment has an ultimate bearer:

- The order of a moment is well defined:

  lemma *order_eq_simp:*

  assumes ‹$x \in \mathcal{E}$›

  shows ‹$order\ x = n \longleftrightarrow (\exists y.\ y \in \mathcal{S} \wedge x \triangleleft\triangleleft^{n}\ y)$›

- The ultimate bearer of an endurant is well defined:

  lemma *ultimate_bearer_eq_simp:*

  assumes ‹$x \in \mathcal{E}$›

  shows ‹$!\beta\ x = y \longleftrightarrow y \in \mathcal{S} \wedge (\triangleleft)^{**}\ x\ y$›

- The ultimate bearer is a substantial:

  lemma *ultimate_bearer_substantial[simp]:*

  ‹$x \in \mathcal{E} \Longrightarrow !\beta\ x \in \mathcal{S}$›

- If a moment inhere directly or indirectly into another endurant, then they both have the same ultimate bearer:

  lemma *ultimate_bearer_eqI1:*

  ‹!β x = !β y› **if** as: ‹x ◁◁ y›

All the axioms regarding the inherence relation in the original theory, stated in the *inherence_original* locale, are derivable from the basic inherence axioms (*inherence_base* locale) plus the noetherian axiom:

sublocale
noetherian_inherence ⊆ *inherence_original*

The axioms of the revised inherence theory *inherence_V2*, that explicitly excludes cycles from the *inherence_original* theory, are also derivable from the noetherian axiom:

sublocale *noetherian_inherence* ⊆ *inherence_V2*

Furthermore, the axioms of the revised inherence theory *inherence_V3*, that explicitly excludes infinite ascending inherence chains from the *inherence_V2* theory, are also derivable from the noetherian axiom:

sublocale
noetherian_inherence ⊆ *inherence_V3*

In fact, the theory *inherence_V3* is logically equivalent to the *noetherian_inherence* theory:

lemma *inherence_V3_and_noetherian_inherence_are_eq:*
‹*inherence_V3* $\mathcal{W}$ *inheresIn* ⟷
noetherian_inherence $\mathcal{W}$ *inheresIn*›
(**is** *"?G1 ⟷ ?G2"*)

Since the noetherian axiom

inherence_is_notherian

is equivalent to the axioms of the *inherence_V3* minus those of of the *inherence_V3* (*inherence_irrefl*, *inherence_asymm*, *inherence_intransitive*, *no_inherence_cycles* and *no_infinite_inherence_chains*), we replace these 5 axioms with the former in the inherence theory that shall be used in this thesis.

However, we add a further axiom to further restrict the inherence relation in such a way that no infinite descending chains occur.

Such a chain would represent a non-ending causal chain of explanation for why a substantial has some property P represented by a certain moment m that is part of the chain. It is as if no final answer could be given as to what grounds the fact that the substantial has the property P: it is P because it bears a moment m_1; m_1 is what it is because it bears moment m_2; m_2 is what it is because it bears moment m_3 and so forth.

To avoid this situation, we require that the inherence relation be not only noetherian but also well-founded. The resulting locale, *inherence*, adds the following axiom to the *noetherian_inherence* locale:

Axiom 13 (*inherence_is_wf*)**.** *The inherence relation must be well-founded, or, in other words, for any non-empty set* Q *of particulars, there must be a member* z *of* Q *such that, for any* y*, if* y *inheres in* z *then* y *is not on* Q*:*

$$wfP\ (\triangleleft),\ or$$

$$\exists x.\ x \in Q \Longrightarrow \exists z{\in}Q.\ \forall y.\ y \triangleleft z \longrightarrow y \notin Q$$

The consequence of the inherence relation being both well-founded and noetherian is that all of its inherence chains are finite:

lemma *inherence_chains_are_finite:*
  assumes *A:* ‹$X \subseteq \mathcal{E}$› ‹$X \neq \emptyset$›
    ‹$\bigwedge x\ y.\ [\![\ x \in X\ ;\ y \in X\ ]\!] \Longrightarrow$
      $x \lhd\lhd y \vee y \lhd\lhd x \vee x = y$›
  shows ‹*finite X*›

3.4 Quality Spaces and Qualia [12]

In the section 3.3, the concept of a *moment* was introduced as a particularization of a property: a moment represents a particular property of a particular object.

In this section, we explain the notion of *quality spaces* and of *quality moments* and how they are used as an element for characterizing properties. The theory presented in this section is a simplification of the original presentation made in (GUIZZARDI, 2005), achieved by omitting the concepts of complex qualities and complex quality spaces.

[12] The full Isabelle/HOL code for this section can be found in section B.3.

The elements introduced so far in the theory enable us to represent substantials, moments and the inherence relationship that relate them. These elements allow us to represent the fact that a substantial has some properties and to infer that the particularized properties of a substantial are distinct from those of another substantial.

However, we they are not enough to determine, for each moment `m`, (1) what type of property does `m` represent, e.g. color, height, mood and (2) what degree, magnitude or value of that type of property that `m` represents.

The notions introduced in this section, *Quality Spaces*, *Qualia*, and the *quale association relation*, enable the representation of these details.

A Quality Space is a set whose elements are called *qualia*, in the plural, or *quale*, in the singular. We represent the type of qualia elements through the type variable `'q`:

type_synonym `'q quality_space = ‹'q set›`

Given a family of sets $\mathcal{QS}$ `:: 'a set set`, we call $\mathcal{QS}$ a *family of quality spaces* just in case the following axioms are valid with respect to $\mathcal{QS}$:

Axiom 14 (`no_empty_quality_space`)**.** *There are no empty sets in $\mathcal{QS}$, i.e., all quality spaces are non-empty:*

$$\emptyset \notin \mathcal{QS}$$

Axiom 15 (`quality_spaces_are_disjoint`)**.** *Quality spaces of $\mathcal{QS}$ are disjoint. In other words, any quale is a member of a single quality space in $\mathcal{QS}$:*

$$[\![Q_1 \in \mathcal{QS};\ Q_2 \in \mathcal{QS};\ Q_1 \cap Q_2 \neq \emptyset]\!] \Longrightarrow Q_1 = Q_2$$

We denote by $\mathcal{Q}$ the set of all qualia, defined as:

definition `qualia (‹`$\mathcal{Q}$`›)`
  where `‹`$\mathcal{Q} \equiv \bigcup\ \mathcal{QS}$`›`

The `qspace` function associates an element of type `'q` to a quality space, if is a quale, or to the empty set, if it is not:

```
definition qspace :: ‹'q ⇒ 'q set› where
  ‹qspace q ≡
      if q ∈ 𝒬 then THE Q. Q ∈ 𝒬𝒮 ∧ q ∈ Q
                else ∅›
```

3.5 Qualified Particulars [13]

Some of the properties an object might have can be properties that are associated with measurements, degrees, forms, and other abstract forms.

For example the property "John's car having a certain reddish color" is associated with a certain color measurement which correspond to the exact tone of color it represents.

Another example is "John's height being 1.80m in length". In this case, the property is associated with the length measurement "1.80 m".

As was explained in section 3.4, we represent such abstract spaces using the concept of *quality spaces*, axiomatized in the `quality_space` locale. In this section, we introduce the notion of qualified moments as those particulars which are associated with elements of a quality space.

To determine whose moments are qualified moments, we extend the signatures `inherence_sig` and `quality_space_sig` with the symbol `(⇝) :: 'p ⇒ 'q ⇒ bool`, calling the resulting signature `qualified_particulars_sig`.

The proposition `assoc_quale m q`, written as `m ⇝ q`, is true just in case `m` is a moment, `q` is a quale and `m` is associated to `q`.

We call the the set of qualified particulars ($\mathcal{P}_q$) the set of moments associated to some quale:

definition `qualifiedParticulars` (‹$\mathcal{P}_q$›) **where**
‹$\mathcal{P}_q$ ≡ { x | x q . x ⇝ q }›

The quale associated to a qualified particular is given by the `qOf` function, where `qOf m` is equal to the associated quale, if `m` is a qualified particular. If `m` is not a qualified particulars, then the value of `qOf` is undefined:

definition `quale_of` (‹`qOf`›) **where** ‹`qOf x ≡ THE q. x ⇝ q`›

Similarly, the quality space indirectly associated with a qualified particular is given by the `qsOf` function, which associates a qualified particular to the quality space which its associated quale is a point of. If `x` is not a qualified particular, then `qsOf x` is the empty set:

definition `qsOf` **where**
‹`qsOf x ≡`
`if x ∈` $\mathcal{P}_q$
`then THE Q. Q ∈` $\mathcal{QS}$ `∧ (∀q. x ⇝ q ⟶ q ∈ Q)`
`else ∅`›

[13] The full Isabelle/HOL code for this section can be found in section B.4.

The `assoc_quale` predicate is constrained by the following axioms:

Axiom 16 (`assoc_quale_scope`). *The* `assoc_quale` *function associates moments to quales:*

$$x \rightsquigarrow q \implies x \in \mathcal{M} \wedge q \in \mathcal{Q}$$

Axiom 17 (`assoc_quale_unique`). *The* `assoc_quale` *function associates a moment to at most one quale:*

$$[\![x \rightsquigarrow q_1;\ x \rightsquigarrow q_2]\!] \implies q_1 = q_2$$

Axiom 18 (`quality_moment_unique_by_quality_space`). *In every possible world* `w`*, for every endurant* `x`*, there can by only one moment inhering in* `x` *for each quality space:*

$$\begin{aligned}&[\![w \in \mathcal{W}; y_1 \in w; y_2 \in w\\ &; y_1 \lhd x; y_2 \lhd x; y_1 \rightsquigarrow q_1\\ &; y_2 \rightsquigarrow q_2; q_1 \in Q; q_2 \in Q\\ &; Q \in \mathcal{QS}]\!] \implies y_1 = y_2\end{aligned}$$

Axiom 19 (`every_quality_space_is_used`). *Every quality space in the family of quality spaces must be associated to at least one moment:*

$$Q \in \mathcal{QS} \implies \exists x.\ \exists q{\in}Q.\ x \rightsquigarrow q$$

Axiom 20 (`quale_determines_moment`). *Given a bearer* `x` *and a quale* `q`*, there can be only one moment inhering in* `x` *associated with* `q`*:*

$$[\![y_1 \lhd x;\ y_2 \lhd x;\ y_1 \rightsquigarrow q;\ y_2 \rightsquigarrow q]\!] \implies y_1 = y_2$$

3.6 Towardness Relation [14]

In the UFO theory of endurants, substantials are the bearers of properties. As was explained in Section sec:qualified-particulars, the intrinsic properties of a substantial are grounded in the existence of moments which inhere in it and which are associated with elements from a space of qualia.

Similarly, *material relational properties* are also grounded in the existence of certain moments. More specifically, there are two kinds of moments that are present whenever a

[14] The full Isabelle/HOL code for this section can be found in section B.5.

material relation exists between two or more substantials: *externally dependent moments*, whose existence depends not only on its bearer, but also on another substantial, distinct from its bearer; and *relators*, which are complex moments formed by the mereological sum of externally dependent moments that share a common *founding event*.

In the present work, for the sake of simplifying the theoretical treatment of the notion of identity, an effort was made to avoid the introduction of a primitive parthood relation. However, as explained in the previous paragraph, the notion of parthood is required for introducing the UFO's notion of a *relator*.

The adopted alternative was to simplify the characterization of relational reality in a way that avoids using parthood relations while, at the same time, preserving a minimally useful capability of representing relational reality. To achieve that, we introduce the relation of *towardness* between a moment and a substantial distinct from its bearer.

Given a moment m inherent to a substantial s and another substantial s^*, distinct from s, if the towardness relation holds between m and s^* then:

- m represents a relational property of s;
- this relational property holds between s and s^*;
- m represents the relational property in the direction $s \rightarrow s^*$.

Such moments are called *directed moments*, since they are, in a sense, *pointed towards* another substantial then their own bearers.

The fact that m represents the particularization of a specific relation between s and s^* implies that m is existentially dependent upon both, i.e. that m is an externally dependent moment.

Given some substantials s and s^*, some examples of what a moment m that inheres in s and is directed to, or stands in a *towardness* relation with s^* could represent are:

- s being spatially apart from s^*;
- s being at specific distance (e.g. 10 km) from s^*;
- supposing s and s^* are human beings, s being interested in s^*;
- s being the father of s^*;

At first glance, the notion of directed moments seems very similar to the notion of externally dependent moments used in the original theory. However, the directionality of externally dependent moments, in the original theory, is not represented directly, but

can be recovered by the existential dependencies of the moment. However, the towardness relation may be stronger than external existential dependency, because a moment directed towards a certain substantial will be existentially dependent also on any other substantials its target is dependent upon.

By representing this directionality explicitly, we give it a status similar to what the inherence relation represents. In the same way, for example, that a representation of a particularized property should contain the explicit representation of the link between this property and the bearer of the property, i.e. the inherence relation, the representation of a relational property should also contain the explicit representation for the link between the property and the target of the relation it represents.

For the reasons explained before, we do not introduce the notion of a relator as a fusion of externally dependent moments or, in this case, of directed moments. Due to this decision we lose the ability to represent material relations that involve more than two relata, or that are composed by multiple individual relational properties, as an individual.

3.6.1 Towardness theory

Formally, we presume the existence of a relation called `towards` that holds between particulars, where we use the notation $x \longrightarrow y$ to state that the relation holds between particulars x and y, or that x is *directed towards* y. The towardness relation satisfies the following axioms:

Axiom 21 (`towardness_scope`). *The towardness relation holds between moments and substantials:*

$$x \longrightarrow y \Longrightarrow x \in \mathcal{M} \wedge y \in \mathcal{S}$$

Axiom 22 (`towardness_imp_ed`). *A moment being directed towards a substantial implies that the former is existentially dependent upon the later:*

$$x \longrightarrow y \Longrightarrow ed\ x\ y$$

Axiom 23 (`towardness_diff_ultimate_bearers`). *The substantial a directed moment is directed towards must be distinct from the substantial that is its ultimate bearer:*

$$x \longrightarrow y \Longrightarrow !\beta\ x \neq y$$

Axiom 24 (`towardness_single`). *Towardness is unique, i.e. a moment can only be directed towards a single substantial:*

$$[\![x \longrightarrow y_1;\ x \longrightarrow y_2]\!] \Longrightarrow y_1 = y_2$$

We use the notation $\mathcal{M}_{\rightarrow}$ to represent the set of directed moments:

definition *directed_moments* (‹$\mathcal{M}_{\rightarrow}$›) **where**
‹$\mathcal{M}_{\rightarrow} \equiv \{ m \,.\, \exists x.\ m \longrightarrow x \}$›

3.7 Theory of Particulars [15]

The simplified theory of UFO particulars presented in this chapter has its signature, given by the locale *ufo_particular_theory_sig*, as the merge of the *inherence_sig*, *qualified_particulars_sig*, and *towardness_sig* signature locales.

The axioms of the theory *ufo_particular_theory* consist in those of the theory *qualified_particulars* plus those of the theory *towardness*, with the addition of a single axiom meant to simplify the theory further to facilitate the logical analysis that shall be done in the next chapters:

Axiom 25 (*qualified_particulars_are_not_bearers*)**.** *Qualified moments cannot bear other moments:*

$$x \in \mathcal{P}_q \Longrightarrow \neg\ y \lhd x$$

3.8 Universals and Instantiation [16]

This section presents a brief theory of Universals and Instantiation based on the original theory presented in (GUIZZARDI, 2005).

An universal, in UFO, is the object in the theory that represents a concept. For example, the concepts *Car*, *Human Being*, *Red Colored* which could stand for, respectively, cars, human beings, and things that have a red color, are represent in UFO as Universals. A particular is called an *instance of* any universal whenever it satisfies the conditions that characterize the later. For example, one of the instances of the *Human Being* universal is this thesis author himself.

The instantiation relation between particulars and universals is represented formally through the *iof* :: *'p* ⇒ *'p set* ⇒ *'u* ⇒ *bool* symbol. The expression $m ::_w U$ means that in the possible world *w*, the particular *m* instantes *U*.

As the signature of the *iof* symbol hints, instantiation is a contigent relationship, i.e. a particular that instantiates an universal *U* in a possible world (e.g. in the actual

[15] The full Isabelle/HOL code for this section can be found in section B.6.
[16] The full Isabelle/HOL code for this section can be found in section B.7.

world) may not instantiate it in another possible world. For example, in one possible world, the author of this thesis may instantiate the *Anxious* and *PhD Candidate* universals, but in another possible world, the author might instante the *Happy* and *PhD* universals instead.

The locale *instantiation_sig* consists in the signature of the simplified particulars theory, given by the locale *ufo_particular_theory_sig*, extended with the symbol *iof :: 'p ⇒ 'p set ⇒ 'u ⇒ bool*.

Given an interpretation for the *iof* function, we can define the following concepts:

The set of universals are exactly those entities that are instantiated by something in some possible world:

definition ‹$\mathcal{U} \equiv \{ u \mid x\ w\ u\ .\ x ::_w u\}$›

This means that we do not accept impossible universals, i.e. those that cannot possibly have an instance.

We say that an universal u_1 *subsumes* an universal u_2 (written as $u_1 \sqsubseteq u_2$) just in case in every possible world, every instance of u_1 is also an instance of u_2:

definition *subsumes* **(infix** ‹$\sqsubseteq$› *75)* **where**
‹$u_1 \sqsubseteq u_2 \longleftrightarrow u_1 \in \mathcal{U} \wedge (\forall x\ w.\ x ::_w u_1 \longrightarrow x ::_w u_2)$›

For example, the universal *Dog* subsumes the universal *Animal* which instantes the universal *Living Being*.

We say that an universal *U* is characterized by another universal *u* just in case, in every possible world *w*, every instance of *U* bears a moments that instantiates the universal *u*.

definition *char_by* **where**
‹$char_by\ U\ u \longleftrightarrow U \in \mathcal{U} \wedge (\forall x\ w.\ x ::_w U \longrightarrow (\exists y.\ y \lhd x \wedge y ::_w u))$›

The set of instance of an universal *U* is given by the function *Insts* and consists in all instances that *U* has in all possible worlds:

definition *Insts* :: ‹*'u ⇒ 'p set*› **where**
‹$Insts\ U \equiv \{ x \mid x\ w\ .\ x ::_w U \}$›

The set of *substantial universals* consists in all universals that have some substantial as an instance:

definition ‹$\mathcal{U}_{\mathcal{S}} \equiv \{ u \mid x\ w\ u\ .\ x ::_w u \wedge x \in \mathcal{S}\}$›

Likewise, the set of *moment universals* consists in all universals that have some moment as an instance:

definition ‹$\mathcal{U}_{\mathcal{M}} \equiv \{ u \mid x\ w\ u\ .\ x ::_w u \wedge x \in \mathcal{M}\}$›

An universal u is considered *rigid* if and only if every instance of u always instantes it in all possible worlds in which they exist:

definition *rigid* :: ‹'u ⇒ bool› **where**
‹*rigid* $u \equiv u \in \mathcal{U} \wedge (\forall x\ w_1\ w_2.\ x ::_{w_1} u \wedge w_2 \in \mathcal{W} \wedge x \in w_2 \longrightarrow x ::_{w_2} u)$›

The basic theory of instantiation, described in the locale *instantiation_thy*, extends the *ufo_particular_theory* locale with the *instantiation_sig* signature and with the following axiom:

Axiom 26 (*iof_imp_existence*)**.** *If a particular* x *instantiates some universal in a possible world* w*, then then* x *must exist in* w*:*

$$x ::_w U \Longrightarrow w \in \mathcal{W} \wedge x \in w$$

The theory of UFO universals further extend the theory of instantiation with the following axioms:

Axiom 27 (*substantial_moment_univs_separate*)**.** *Universals are either Substantial Universals or Moment Universals, i.e., instantiation does not cross categories. In other words, an universal can either have only moments or substantials as instances:*

$$\mathcal{U}_{\mathcal{S}} \cap \mathcal{U}_{\mathcal{M}} = \emptyset$$

Axiom 28 (*moment_universals_are_rigid*)**.** *Moment universals are rigid:*

$$u \in \mathcal{U}_{\mathcal{M}} \Longrightarrow \textit{rigid}\ u$$

3.9 Final Considerations

In this chapter we presented a formal specification of a fragment of the UFO-A ontology using the Isabelle/HOL formalism. The presentation consists in commented parts of the complete Isabelle/HOL theory that is included in **??**.

The UFO-A fragment was chosen as to keep a sufficient collection of concepts and formal relations that enable the analysis of the notions of identity, individuality and sortality, meaning that there should be a minimal formal apparatus for representing intrinsic and extrinsic properties of particulars.

In this selection, we skipped the characterization of the Event concept, since our focus is restricted to the interpretation of the notions of individuality, identity and sortality

in the context of UFO Endurants. Furthermore, the higher status given to the Endurant concept versus the Event concept in UFO (an endurantist ontology) indicates that the definition of fundamental properties of endurants in UFO should not rely on the Event concept and, thus, that the later is not necessary for the analysis of those properties.

We also simplified the notion of Quality Spaces and of Qualities (here called Qualified Moments), when compared to the original presentation, by omitting the concept of Complex Qualities and Complex Quality Spaces.

Although this omission means that the product structure of some properties (such as Color) cannot be directly represented, the possibility of representing more than one intrinsic property through multiple moments is enough to express a plurality of intrinsic properties of an endurant.

Similarly, the omission of the Relator concept means that we lack a direct representation for complex material relations. However, since our focus is on the concept of identity and individuality, that are applicable to Substantials, and in the concept of Sortality, applicable to Substantial Universals, we can omit the Relator concept if we include some element that is capable of representing basic material relations (binary relations). This simplification was done through the introduction of the *towardness* relation.

We then brought all these formal specifications together in a final theory of UFO particulars and also presented a minimal theory of UFO universals and instantiation relations.

Finally, in section 3.3 we identified a logical gap in the original specification of the inherence relation that allows some undesirable interpretations of the inherence relation. We also identified the precise conditions under which this issue appears and propose a necessary and sufficient condition that fills this gap and that also replaces several of the original axioms.

4 Particular Structures and Morphisms

4.1 Introduction

Before we analyze the notion of identity *per se*, we should consider the question of whether or not a notion of identity can be defined in our theory for a certain configuration of particulars. This analysis transcends the level of the original theory in the sense that it requires answering questions at the meta-logical level.

Questions such as "Is identity definable?" or "Can I express a rigid definite description that identifies particular P in all possible worlds?" do not only depend on the language used to express these definitions, but also in the configuration of things that is being considered.

By relying on a syntactical and semantical analysis of the definability of identity in a particular formalism, undesirable elements are introduced in the analysis and definition of what does "a particular having (or not) an identity" stands for, i.e. a choice of proposition (a linguistic element) and choice of a formal language in which this proposition is expressed (again a linguistic element).

However, one of the hypothesis under which this work was undertaken is that it is possible to avoid the reliance on linguistic elements for the analysis and definition of the concept of identity by considering it under a *structural* perspective.

This chapter proposes a framework that allows a structural analysis of properties of concepts of the UFO fragment theory proposed in chapter 3. This approach brings the models of the theory of UFO particulars described in section 3.7 into the object level in the form of Isabelle/HOL structures (records) and proposes a category in which the objects are the UFO particular theory models (particular structures) and the morphisms are meaning preserving mappings between particulars of those models.

Besides defining the elements that compose the category, we also define several derived concepts that are useful for the analysis of the concept of identity, individuality and sortality.

The main sections of this chapter are structured as follows: we begin, in section 4.2, with the definition of an Isabelle/HOL *record* structure that represents a model of the UFO theory of particulars described in chapter 3, called an UFO particular structure; we then define, in section 4.3, a notion of *morphisms between UFO particular structures* that capture the notion of a meaning-preserving mapping from an UFO particular structure to another; in section 4.4, we present several structural properties of particulars that

are defined with respect to a UFO particular structure and to its morphisms and prove their logical relationships; finally, in section 4.5, we present some examples of particular structures and particular structure morphisms, proving the consistency of the theory.

4.2 Particular Structures [1]

In this chapter, we present a category of UFO particular structures, whose objects are Isabelle/HOL records that encode a model of the UFO theory of particulars presented in chapter 3.

The record that encodes a particular structure is defined as follows:

```
record ('p,'q) particular_struct =
  ps_quality_spaces :: ‹'q set set›
  ps_worlds :: ‹'p set set›
  ps_inheres_in :: ‹'p ⇒ 'p ⇒ bool›
  ps_assoc_quale :: ‹'p ⇒ 'q ⇒ bool›
  ps_towards :: ‹'p ⇒ 'p ⇒ bool›
```

The six fields of the particular structure record represent the formal elements of the UFO particular theory:

- `ps_worlds`: The set of possible worlds ($\mathcal{W}$)
- `ps_inheres_in`: The inherence relationship ($\triangleleft$)
- `ps_quality_spaces`: The set of quality spaces ($\mathcal{QS}$)
- `ps_assoc_quale`: The relationship of quale association ($\rightsquigarrow$)
- `ps_towards`: The towardness relation ($\longrightarrow$)

4.3 Particular Structures Homomorphisms [2]

In this section we show how certain functions between sets of particulars of two particular structures give rise to a notion of homorphisms between these structures. We call such functions *particular structure homomorphisms* or simply *particular structure morphisms*.

[1] The full Isabelle/HOL code for this section can be found in section C.1.
[2] The full Isabelle/HOL code for this section can be found in section C.2.

4.3.1 Definitions

Let's start with two particular structures, Γ_1`::'a` and Γ_2`::'a`, where `'p`$_1$ and `'p`$_2$ denote two types of particulars and `'q` denotes a type of qualia. We call Γ_1 the *source structure* and Γ_2 the *target structure*. In this context, for the purpose of simplifying the theories presented in this section, we define the following abbreviations:

The infix symbols `(`$\lhd_s$`)` and `(`$\lhd_t$`)` denote, respectively, the *inherence* relations of the source and target structures:

abbreviation *src_inheres_in* **(infix** ‹$\lhd_s$› *75)* **where**
‹*x* $\lhd_s$ *y* ≡ *src.inheresIn x y*›

abbreviation *tgt_inheres_in* **(infix** ‹$\lhd_t$› *75)* **where**
‹*x* $\lhd_t$ *y* ≡ *tgt.inheresIn x y*›

The infix symbols `(`$\rightsquigarrow_s$`)` and `(`$\rightsquigarrow_t$`)` denote, respectively, the *quale association* relations of the source and target structures:

abbreviation *src_assoc_quale* **(infix** ‹$\rightsquigarrow_s$› *75)* **where**
‹*x* $\rightsquigarrow_s$ *y* ≡ *src.assoc_quale x y*›

abbreviation *tgt_assoc_quale* **(infix** ‹$\rightsquigarrow_t$› *75)* **where**
‹*x* $\rightsquigarrow_t$ *y* ≡ *tgt.assoc_quale x y*›

The infix symbols `(`$\longrightarrow_s$`)` and `(`$\longrightarrow_t$`)` denote, respectively, the *towards* relations of the source and target structures:

abbreviation *src_towards* **(infix** ‹$\longrightarrow_s$› *75)* **where**
‹*x* $\longrightarrow_s$ *y* ≡ *src.towards x y*›

abbreviation *tgt_towards* **(infix** ‹$\longrightarrow_t$› *75)* **where**
‹*x* $\longrightarrow_t$ *y* ≡ *tgt.towards x y*›

The symbols $\mathcal{Q}_s$ and $\mathcal{Q}_t$ denote, respectively, the sets of qualia of the source and target structures:

abbreviation ‹$\mathcal{Q}_s$ ≡ *src.qualia*›

abbreviation ‹$\mathcal{Q}_t$ ≡ *tgt.qualia*›

For any symbol `S` defined in the context of a particular structure or in the context of a particular theory, i.e. those defined in the *particular_struct_sig* or in the *ufo_particular_theory_sig* signatures, the corresponding symbols of the Γ_1 and Γ_2 structures can be referred to by prefixing with *src.* or *tgt.*. For example, the set of possible

worlds of Γ_1 and Γ_2 can be referred to, respectively, using the expressions `src.`$\mathcal{W}$ and `tgt.`$\mathcal{W}$.

In this context, let φ denote a function from particulars of type `'p`$_1$ to particulars of type `'p`$_2$. We call the signature composed of Γ_1, Γ_2 and φ a *particular structure morphism* signature, referred to in the formal theory as `particular_struct_morphism_sig`. In the context of this signature we define the following symbols:

The symbol $\mathcal{P}_{img}$ denotes the *image* of the set of particulars of the source structure over the morphism function:

definition `morph_image (`‹$\mathcal{P}_{img}$›`)` **where**
‹$\mathcal{P}_{img} \equiv \varphi$ ` ‘ src.`$\mathcal{P}$›

The symbol φ^{-1} denotes a function from the type of particulars `'p`$_2$ to the type of particulars `'p`$_1$ such that it *inverts* the morphism function φ with regards to the set of particulars of the source structure:

definition `inv_morph (`‹φ^{-1}›`)` **where**
‹φ^{-1} `= inv_into src.`$\mathcal{P}$ φ›

4.3.2 Axiomatization

Thus, to define a notion of morphism based on the function φ, we need to define under what conditions can such a function be judged as a mapping that preserves the structure of the Γ_1 and Γ_2 structures. As the choice of conditions is somewhat arbitrary, we need some guiding principles to driver their specification. In this context, we call `(`φ`,`Γ_1 `,`Γ_2`)` a particular structure pre-morphism (formally a `pre_particular_struct_morphism`) just in case the following conditions are met:

Axiom 29. `quality_space_subset` *Every quality space present in the source structure is also present in the target structure. Formally:* `src.`$\mathcal{QS} \subseteq$ `tgt.`$\mathcal{QS}$.

Axiom 30. `morph_preserves_particulars` *The morphism φ preserves* particularity*, i.e. any particular of the source structure is mapped to a particular in the target structure:*

$$x \in \texttt{src.endurants} \Longrightarrow \varphi\ x \in \texttt{tgt.endurants}$$

Axiom 31. `morph_reflects_inherence` *The morphism φ does not* add *or* change *inherence relations between target particulars that are φ-images of source particulars. In other words, for any two particulars of the source structure, their corresponding image stand in a* inherence *relation if and only if they also stand in a inherence relation in the source structure:*

$$[\![x \in \texttt{src.endurants};\ y \in \texttt{src.endurants}]\!] \Longrightarrow \varphi\ x \lhd_t \varphi\ y = x \lhd_s y$$

Axiom 32. `morph_does_not_add_bearers` *The morphism φ does not add bearers to substantials of the source structure. Equivalently, the image of a particular of the source structure has a bearer if and only if that bearer is also the image of some source particular. This together with the* `morph_reflects_inherence` *axiom is equivalent to the first statement. Formally:*

$$[\![x \in \mathtt{src.endurants};\ \varphi\ x \lhd_t z]\!] \Longrightarrow \exists y \in \mathtt{src.endurants}.\ z = \varphi\ y$$

Axiom 33. `morph_reflects_towardness` *The morphism φ does not* add *or* change *"towards" relations between target particulars that are φ-images of source particulars. In other words, for any two particulars of the source structure, their corresponding image stand in a* towards *relation if and only if they stand in a towards relation in the source structure:*

$$[\![x \in \mathtt{src.endurants};\ y \in \mathtt{src.endurants}]\!] \Longrightarrow \varphi\ x \longrightarrow_t \varphi\ y = x \longrightarrow_s y$$

Axiom 34. `morph_does_not_add_towards` *The morphism φ does not add a* towards *relation to images of source particulars. In other words, a target moment that is the image of a source moment is directed "towards" another target moment if and only if it is the image of some moment that was directed to some other source moment:*

$$[\![x \in \mathtt{src.endurants};\ \varphi\ x \longrightarrow_t z]\!] \Longrightarrow \exists y \in \mathtt{src.endurants}.\ z = \varphi\ y$$

Axiom 35. `morph_reflects_quale_assoc` *The morphism φ preserves quale association of source particulars. If a source particular is associated with some quale then its image is also associated with the same quale, and vice-versa:*

$$x \in \mathtt{src.endurants} \Longrightarrow x \rightsquigarrow_s q = \varphi\ x \rightsquigarrow_t q$$

Besides the preservation conditions detailed in these axioms, the following preservation lemmas are derivable from the same axioms:

Lemma 8. `morph_preserves_substantials` *For any particular* `x` *of Γ_1 the following statements are logically equivalent:*

- `x` *is a substantial (in Γ_1);*
- φ `x` *is a substantial (in Γ_2).*

Lemma 9. `morph_preserves_moments_simp` *For any particular* `x` *of Γ_1 the following statements are logically equivalent:*

- `x` *is a moment (in Γ_1);*

- φ `x` *is a moment (in* Γ_2*).*

Lemma 10. `morph_reflects_bearers` *For any moment* `x` *of* Γ_1*, the image of the bearer, in* Γ_1*, of* `x` *is the bearer, in* Γ_2*, of the image of* `x`*, i.e.*

$$x \in src.\mathcal{M} \Longrightarrow tgt.bearer\ (\varphi\ x) = \varphi\ (src.bearer\ x)$$

Lemma 11. `morph_reflects_ultimate_bearers` *For any particular* `x` *of* Γ_1*, the image of the ultimate bearer, in* Γ_1*, of* `x` *is the ultimate bearer, in* Γ_2*, of the image of* `x`*, i.e.*

$$x \in src.endurants \Longrightarrow tgt.ultimateBearer\ (\varphi\ x) = \varphi\ (src.ultimateBearer\ x)$$

So far, the axioms presented in `pre_particular_struct_morphism` only describe restrictions on the elements of particular structures that characterize relationships of particular structure particulars. No restrictions were imposed on the sets of possible world sets of the source and target structure.

Before we add such axioms, we need to determine what structural properties should be preserved by φ when we say that "φ preserves possible world structures". Basically, we want to preserve the properties that are characterized by the set of possible worlds. Namely, we want to preserve the existential dependency and existential independence of particulars. In other words, if a particular is existentially dependent upon another in the source structure, than the image of the first should also be existentially dependent upon the image of the later. Similarly, the image of particulars that are existentially independent in the source structure should also be independent in the target structure.

We frame this condition using the notion of correspondence between worlds. Given a world w_s of the source structure and a world w_t of the target structure, we say that w_s and w_t correspond, written as $w_s \Leftrightarrow w_t$, if the image of w_s under φ corresponds to the largest subset of w_t that is the image of a subset of the set of particulars of the source structure. Phrased in another way, w_s and w_t correspond just in case, for every source particular x, x is in w_s if and only if x's image under φ is in w_t. Formally, we have:

definition *world_corresp* **(infix** ‹⇔› *75)* **where**

$$\langle w_s \Leftrightarrow w_t \equiv w_s \in src.\mathcal{W} \wedge w_t \in tgt.\mathcal{W} \wedge (\forall x \in src.\mathcal{P}.\ (x \in w_s \longleftrightarrow \varphi\ x \in w_t))\rangle$$

We can now complete the formal definition of a particular structure morphism by adding the following axioms to the previously introduced axioms, which state that every possible world in the source structure must have at least one corresponding possible world in the target structure, and vice-versa:

Axiom 36. `morph_worlds_correspond_src_tgt`

$$w_s \in src.\mathcal{W} \Longrightarrow \exists\, w_t.\ w_s \Leftrightarrow w_t$$

These axioms guarantee that existential dependency and existential independence are preserved by φ, as per the following lemmas:

Lemma 12. `morph_reflects_ed_simp` *Given a particular structure morphism φ from Γ_1 to Γ_2, and any Γ_1 particulars* `x` *and* `y`*, the following statements are logically equivalent:*

- `x` *and* `y` *are existentially dependent (in Γ_1)*
- φ `x` *and* φ `y` *are existentially dependent (in Γ_2)*

Lemma 13. `morph_reflects_src_indep_simp` *Given a particular structure morphism φ from Γ_1 to Γ_2, and any Γ_1 particulars* `x` *and* `y`*, the following statements are logically equivalent:*

- `x` *and* `y` *are existentially independent (in Γ_1)*
- φ `x` *and* φ `y` *are existentially independent (in Γ_2)*

4.3.3 The Category of Particular Structures

One of the properties of particular struct morphisms is that they are closed under composition:

Lemma 14. `particular_struct_morphism_comp` *For any morphisms φ_{12}*`::'p`$_1$ $\Rightarrow$ `'p`$_2$*, from Γ_1 to Γ_2, and φ_{23}*`::'p`$_2$ $\Rightarrow$ `'p`$_3$*, from Γ_2 to Γ_3, the functional composition of φ_{12} and φ_{23} is a morphism from Γ_1 to Γ_3.*

Furthermore, the identity function serves as an identity morphism for every particular structure:

Lemma 15. `id_is_a_morphism` *For any Γ*`::('p, 'q) particular_struct`*, the identity function on type* `'p` *is a morphism of Γ:*

`particular_struct_morphism` Γ Γ `id`

Since the class of particular structure morphisms is a subclass of the class functions that is closed under composition, they are also associative, i.e.

$(\varphi_{34} \circ \varphi_{23}) \circ \varphi_{12} = \varphi_{34} \circ (\varphi_{23} \circ \varphi_{12})$

By the same token, identity functions ($\lambda x::'t.\ x$)), being particular structure morphisms for any particular structure whose particular type is *'t*, are left and right identities for the composition of morphisms:

$id \circ \varphi = \varphi$

$\varphi \circ id = \varphi$

Thus, the class of particular structures and particular structure morphisms can be considered a *category*, i.e. for any type *'q* of qualia we have a category that:

- the objects are the particular structures whose particular type is *'p* for any *'p*;
- the set of morphisms between Γ_1*::('p*$_1$*, 'q) particular_struct* and Γ_2*::('p*$_2$*, 'q) particular_struct* is the set of functions of the form φ*::'p*$_1 \Rightarrow$ *'p*$_2$ that satisfy the conditions for a particular structure morphism, i.e. that satisfy *particular_struct_morphism* $\Gamma_1\ \Gamma_2\ \varphi$;
- for any pair φ_1*::'p*$_1 \Rightarrow$ *'p*$_2$ and φ_2*::'p*$_2 \Rightarrow$ *'p*$_3$ of morphisms between particular structures Γ_1, Γ_2 and Γ_3, their composition is given by function composition;
- for any Γ*::('p, 'q) particular_struct*, the identity function $\lambda x::'p.\ x$ is the identity for Γ;
- composition of particular structure morphisms is associative;
- composition of particular structure morphisms satisfy left and right unit laws, i.e. $id \circ \varphi = \varphi$ and $\varphi \circ id = \varphi$.

4.3.4 Classes of Particular Structure Morphisms

By using the concepts of injective, surjective and bijective functions, and by borrowing the notions of *endomorphism* and of *permutations*, we can classify the class of particular structure morphisms into corresponding subclasses that are relevant for the theories developed later in this thesis.

These classes are characterized by the following axioms:

A particular structure morphism φ from Γ_1 to Γ_2 is an injective morphism if and only if it satisfies the following axiom:

Axiom 37. `morph_is_injective` *the morphism is injective on the set of particulars of* Γ_1*, i.e.*

$$[\![\varphi\ x = \varphi\ y;\ x \in \mathit{src.endurants};\ y \in \mathit{src.endurants}]\!] \Longrightarrow x = y$$

A particular structure morphism φ from Γ_1 to Γ_2 is an surjective morphism if and only if it satisfies the following axiom:

Axiom 38. `morph_is_surjective` *the morphism is surjective from the set of particulars of* Γ_1*, to the set of particulars of* Γ_2*, i.e.*

$$\varphi\ `\ \mathit{src.endurants} = \mathit{tgt.endurants}$$

A particular structure morphism φ from Γ_1 to Γ_2 is an called an isomorphism (`particular_struct_bijection`) just in case it is both an injective morphism and a surjective morphism.

A particular structure morphism φ from Γ_1 to Γ_2 is called an endomorphism (`particular_struct_endomorphism_sig`) if and only if $\Gamma_1 = \Gamma_2$.

abbreviation *endurants* (‹$\mathcal{P}$›)
where ‹$\mathcal{P} \equiv$ *src.endurants*›

Since, in the context of an endomorphism, the source structure is equal to the target structure, we can used the symbols that are using in the context of a particular structure without ambiguity, e.g. $\mathcal{P}$, $(\Leftrightarrow)$, $(\triangleleft)$, $(\longrightarrow)$, $(\rightsquigarrow)$, $\mathcal{S}$ and $\mathcal{M}$.

A morphism φ is called a *permutation morphism* just in case it is both an endomorphism and also a bijective morphism.

The following sets represent the collections of morphisms that fall into each morphism class:

definition *morphs* (‹*Morphs*_,_› *[999,999] 1000)* **where**
‹$\mathit{Morphs}_{\Gamma,\Gamma'} \equiv \{\varphi\ .\ \mathit{particular_struct_morphism}\ \Gamma\ \Gamma'\ \varphi\ \}$›

definition *injectives* (‹*InjMorphs*_,_› *[999,999] 1000)* **where**
‹$\mathit{InjMorphs}_{\Gamma,\Gamma'} \equiv \{\varphi\ .\ \mathit{particular_struct_injection}\ \Gamma\ \Gamma'\ \varphi\ \}$›

definition *surjectives* (‹*SurjMorphs*_,_› *[999,999] 1000)* **where**
‹$\mathit{SurjMorphs}_{\Gamma,\Gamma'} \equiv \{\varphi\ .\ \mathit{particular_struct_surjection}\ \Gamma\ \Gamma'\ \varphi\ \}$›

definition *bijectives* (‹*BijMorphs_,_*› *[999,999] 1000)* **where**
‹$BijMorphs_{\Gamma,\Gamma'}$ ≡ {φ . *particular_struct_bijection* Γ Γ' φ }›

definition *endomorphisms* (‹*EndoMorphs_*› *[999] 1000)* **where**
‹$EndoMorphs_{\Gamma}$ ≡ {φ . *particular_struct_endomorphism* Γ φ }›

definition *permutations* (‹*Perms_*› *[999] 1000)* **where**
‹$Perms_{\Gamma}$ ≡ {φ . *particular_struct_permutation* Γ φ }›

For each morphism class, we can derive a useful set of lemmas. We highlight some of those here, but the reader is referred to the full Isabelle/HOL code for the complete collection:

The main feature of the class of injective morphisms is the existence of an inverse for the morphism function. The morphism and its inverse determine a bijection between the set of particulars of the source structure and its image on the set of particulars of the target structure. The properties regarding the injective morphism and its inverse are shown in the following lemmas.

Lemma 16. *inv_morph_morph* *For any injective morphism function, its inverse into the set of particulars of the source structure is well defined and works as a left-inverse (or a retraction):*

$x \in$ *src.endurants* $\Longrightarrow \varphi^{-1}$ (φ *x*) = *x*

Lemma 17. *morph_bij_on_img* *As an injective function, the morphism is a bijection to its image:*

bij_betw φ *src.endurants* $\mathcal{P}_{img}$

Lemma 18. *phi_inv_img* *Since φ is injective, its inverse is a surjection from φ's image to the source set of particulars:*

φ^{-1} ‘ $\mathcal{P}_{img}$ = *src.endurants*

Lemma 19. *phi_inv_inj_on_I_img* *The inverse is injective on the image of the morphism φ:*

inj_on φ^{-1} $\mathcal{P}_{img}$

Lemma 20. `phi_inv_bij_on_src_I` *Since it is both injective on the source set of particulars* `src.`$\mathcal{P}$ *and surjective onto the image of* φ *(*φ `‘` `src.`$\mathcal{P}$*), the morphism* φ *is a bijection between* `src.`$\mathcal{P}$ *and* φ `‘` `src.`$\mathcal{P}$*:*

$$\texttt{bij_betw}\ \varphi^{-1}\ \mathcal{P}_{img}\ \texttt{src.endurants}$$

Lemma 21. `morph_inv_morph_img` *Since it's surjective on* φ*'s image, the inverse morphism is a right-inverse for the* φ*'s image:*

$$x \in \mathcal{P}_{img} \Longrightarrow \varphi\ (\varphi^{-1}\ x) = x$$

Lemma 22. `inv_inheres_in_reflects_on_image` *The inverse of an injective morphism function reflects the inherence relation, i.e. for any* `x` *and* `y` *in the image of* φ*,* `x` *inheres in* `y` *if and only if their respective inverses also stand in the same relation:*

$$[\![x \in \mathcal{P}_{img};\ y \in \mathcal{P}_{img}]\!] \Longrightarrow x \lhd_t y = \varphi^{-1}\ x \lhd_s \varphi^{-1}\ y$$

Lemma 23. `inv_towardness_reflects_on_image` *Similarly, the inverse of an injective morphism also reflects the towardness relation, i.e. for any* `x` *and* `y` *in the image of* φ*,* `x` *os directed towards* `y` *if and only if their respective inverses also stand in the same relation:*

$$[\![x \in \mathcal{P}_{img};\ y \in \mathcal{P}_{img}]\!] \Longrightarrow x \longrightarrow_t y = \varphi^{-1}\ x \longrightarrow_s \varphi^{-1}\ y$$

Lemma 24. `world_preserve_inv_img` *Given an injective morphism, it is possible to determine some of its source possible worlds from its target possible worlds, using the following lemma:*

$$w_t \in \texttt{tgt.}\mathcal{W} \Longrightarrow \varphi^{-1}\ \texttt{‘}\ (w_t \cap \mathcal{P}_{img}) \in \texttt{src.}\mathcal{W}$$

A morphism function that is a surjection onto the set of particulars of its target structure is characterized by the fact that its image corresponds to the set of particulars of the target structure and by the fact that the morphism function has a right-inverse function, as per the following lemmas:

Lemma 25. `I_img_eq_tgt_I` *The image of surjective morphism function is equivalent to the set of particulars if the target structure:*

$$\mathcal{P}_{img} = \texttt{tgt.endurants}$$

Lemma 26. `inv_morph_morph` *For any surjective morphism, there is at least one function that serves as its right-inverse:*

$$x \in tgt.endurants \Longrightarrow \varphi\ (\varphi^{-1}\ x) = x$$

Lemma 27. `world_preserve_img` *Given a surjective morphism, it is possible to determine some of its target possible worlds from its source possible worlds, using the following lemma:*

$$w_s \in src.\mathcal{W} \Longrightarrow \varphi\ `\ w_s \in tgt.\mathcal{W}$$

Lemma 28. `tgt_quality_spaces_eq_src` *In a surjective morphism, the set of quality spaces of the source and target structures are equivalent:*

$$tgt.\mathcal{QS} = src.\mathcal{QS}$$

Lemma 29. `morph_bijective`*:w Since the morphism function is both injective and surjective, it is also bijective:*

$$bij_betw\ \varphi\ src.endurants\ tgt.endurants$$

Lemma 30. `inv_morph_bijective` *The inverse of a bijection is also a bijection:*

$$bij_betw\ \varphi^{-1}\ tgt.endurants\ src.endurants$$

The bijective nature of the morphism function entails the following reflection lemmas, which mirror the reflection axioms defined at `pre_particular_struct_morphism`:

Lemma 31. `inv_inheres_in_reflects` *The morphism function inverse reflects inherence related endurants from the target structure back into the source structure:*

$$[\![x \in tgt.endurants;\ y \in tgt.endurants]\!] \Longrightarrow \varphi^{-1}\ x \lhd_s \varphi^{-1}\ y = x \lhd_t y$$

Lemma 32. `inv_towardness_reflects` *Similarly, the morphism function inverse reflects towardness relata from the target structure back into the source structure:*

$$[\![x \in tgt.endurants;\ y \in tgt.endurants]\!] \Longrightarrow \varphi^{-1}\ x \longrightarrow_s \varphi^{-1}\ y = x \longrightarrow_t y$$

Lemma 33. `inv_assoc_quale_reflects` *The quale association relation is also reflected back into the source structure:*

$$[\![x \in tgt.endurants;\ y \in tgt.endurants]\!] \Longrightarrow \varphi^{-1}\ x \rightsquigarrow_s q = x \rightsquigarrow_t q$$

If there is a bijection between two particular structures, their possible world sets present several useful symmetries, as shown in the following lemmas:

This is a refined version of the `world_preserve_inv_img` (introduced in `particular_struct_inje`

Lemma 34. `world_preserve_inv_img1` *The image of a possible world of the target structure under the inverse of the morphism function is also a possible world in the source structure:*

$$w_t \in tgt.\mathcal{W} \Longrightarrow \varphi^{-1} \text{ `` } w_t \in src.\mathcal{W}$$

Lemma 35. `phi_phi_inv_world` *The image of the inverse function is a right-inverse of the image of the morphism function with respect to the set of possible in worlds in the target structure:*

$$w \in tgt.\mathcal{W} \Longrightarrow \varphi \text{ `` } \varphi^{-1} \text{ `` } w = w$$

Lemma 36. `phi_inv_phi_world` *Similarly, the image of the inverse function is a left-inverse of the image of the morphism function with respect to the set of possible in worlds in the source structure:*

$$w \in src.\mathcal{W} \Longrightarrow \varphi^{-1} \text{ `` } \varphi \text{ `` } w = w$$

Lemma 37. `src_world_corresp_unique` *For every possible world in the source structure, there is an unique correspondent possible world in the target structure:*

$$[\![w \Leftrightarrow w_1;\ w \Leftrightarrow w_2]\!] \Longrightarrow w_1 = w_2$$

Lemma 38. `tgt_world_corresp_unique` *Conversely, for every possible world in the target structure, there is an unique correspondent possible world in the source structure:*

$$[\![w_1 \Leftrightarrow w;\ w_2 \Leftrightarrow w]\!] \Longrightarrow w_1 = w_2$$

Lemma 39. `src_world_corresp_image_iff` *In fact, the only possible world of the target structure that corresponds to a given possible world of the source structure is exactly the image of the source possible world:*

$$w \in src.\mathcal{W} \Longrightarrow w \Leftrightarrow w' = (w' = \varphi \text{ `` } w)$$

Lemma 40. `tgt_world_corresp_inv_image_iff` *Conversely, the only possible world of the source structure that corresponds to a given possible world of the target structure is exactly the inverse image of the target possible world:*

$$w \in tgt.\mathcal{W} \Longrightarrow w' \Leftrightarrow w = (w' = \varphi^{-1} \text{ `` } w)$$

Lemma 41. `inv_is_bijective_morphism` *Finally, we have that the inverse function of a bijective morphism function is also a morphism. More specifically, it is also a bijective morphism:*

$$particular_struct_bijection\ \Gamma_2\ \Gamma_1\ \varphi^{-1}$$

4.4 Structural Properties of Particulars

4.4.1 Isomorphically Unique Particulars [3]

We can study a particular's formal properties in a particular structure through the various possible mappings it may have thorugh morphisms that initiate or target that particular structure. One of these properties if that of being isomorphically unique, defined by considering the bijective morphisms from the particular structure.

The existence of a bijective morphism between particular structures Γ_1 and Γ_2 implies that both structures express the same configuration, differing, if they do, only in representation of particulars.

We say that a particular x is isomorphically unique in a particular structure Γ if and only if, for every bijective morphism φ from Γ to some particular structure Γ', and for every morphism σ from Γ to Γ', φ and σ agree with respect to x.

Since particular structures are polymorphic on the type of particulars and Isabelle/HOL does not allow quantification over types, we need to define the notion of an isomorphically unique particular using some fixed choice of representation for the target particular structures. We chose here to use the `ZF` Isabelle/HOL type of ZF sets as the target representation, assuming that any domain of particulars is representable using ZF sets.

Thus, the formal definition of isomorphically unique particulars is:

definition $isomorphically_unique_particulars\ (\langle\mathcal{P}_{\simeq!}\rangle)$
  where $\langle\mathcal{P}_{\simeq!} \equiv \{\ x\ .\ x \in \mathcal{P} \wedge ($
    $\forall \varphi \in BijMorphs1_{\Gamma,TYPE(ZF)}.\ \forall \sigma \in Morphs_{\Gamma,MorphImg}\ \varphi\ \Gamma.$

[3] The full Isabelle/HOL code for this section can be found in subsection C.4.1.

```
∀y ∈ 𝒫.
σ y = φ x ⟷ y = x)}⟩
```

4.4.2 Permutability [4]

Besides the notion of isomorphically unique particulars, presented in subsection 4.4.1, we can also study the property of a particular in a particular structure through the endomorphisms of that structure.

We say that a particular x in a particular structure Γ is permutable just in case there is an endomorphism φ such that either $\varphi\ x \neq x$, i.e. it maps x to some other particular, or there is some particular $y \neq x$ such that $\varphi\ y = x$, i.e. it maps some other particular to x.

Conversely, we call it non-permutable if there is no other particular y such that x can be "seen" as y or that y can be "seen" as x, through an endomorphism.

Formally, we define a non-permutable particular as:

```
definition non_permutable :: ⟨'p ⇒ bool⟩ where
  ⟨non_permutable x ⟷ (∀φ ∈ EndoMorphsΓ.∀y ∈ 𝒫. φ y = x ⟷ y = x)⟩
```

The set of non-permutables particulars is defined as:

```
definition non_permutables (⟨𝒫1!⟩) where
  ⟨𝒫1! ≡ { x . x ∈ 𝒫 ∧ non_permutable x }⟩
```

4.4.3 Collapsability [5]

While in subsection 4.4.2, we defined the notion of a permutable particular as being one that can, in a sense, be "seen" as another, through an endomorphism, we here introduce the stronger notion of a collapsable particulars.

A particular x of a particular structure Γ is considered to be collapsable if and only if there is at least one other particular y of Γ and at least one Γ-endomorphism that maps x and y to the same particular.

This notion captures the idea of a plurality of particulars that can be seen as being essentially the same, with respect to Γ's structure.

[4] The full Isabelle/HOL code for this section can be found in subsection C.4.2.
[5] The full Isabelle/HOL code for this section can be found in subsection C.4.3.

Formally, we define a collapsable particular as follows:

definition *collapsable* :: ‹'p ⇒ bool› **where**
‹collapsable x ≡ x ∈ $\mathcal{P}$ ∧ (∃φ ∈ $EndoMorphs_{\Gamma}$.∃y ∈ $\mathcal{P}$. x ≠ y ∧ φ x = φ y)›

The set of non-collapsable particulars of Γ is defined as:

definition *nonCollapsableParticulars* (‹$\mathcal{P}_{nc}$›) **where**
‹$\mathcal{P}_{nc}$ ≡ { x . x ∈ $\mathcal{P}$ ∧ ¬ collapsable x }›

4.4.4 Permutability and Isomorphical Uniqueness

It turns out that the set of isomorphically unique particulars of a particular structure is identical to the set of non-permutable particulars, as shown in the following theorem and lemmas:

lemma *im_unique_particulars_are_non_permutable_particulars:*
‹$\mathcal{P}_{\simeq!}$ ⊆ $\mathcal{P}_{1!}$›

lemma *non_permutable_particulars_are_im_unique_particulars:*
‹$\mathcal{P}_{1!}$ ⊆ $\mathcal{P}_{\simeq!}$›

theorem *non_permutable_particulars_are_the_unique_particulars:*
‹$\mathcal{P}_{1!}$ = $\mathcal{P}_{\simeq!}$›

The set of non-permutable particulars is a subset of the set of non-collapsable particulars, although the converse does not hold, in general:

theorem *non_permutables_are_non_collapsable:* ‹$\mathcal{P}_{1!}$ ⊆ $\mathcal{P}_{nc}$›

4.5 Twin Spheres Examples [6]

The examples of particulars structures presented in this section are (loosely) based on the twin spheres case describe in (BLACK, 1952).

First we specify a domain of quales with three values:

datatype *quale* =

[6] The full Isabelle/HOL code for this section can be found in section C.6.

```
    Q_Dist200M
  | Q_Radius10M
  | Q_Radius20M
```

The values correspond to:

- *Q_Dist200M*: is a distance quale corresponding to 200 meters;
- *Q_Radius10M*: is a radius length quale corresponding to 10 meters;
- *Q_Radius10M*: is a radius length quale corresponding to 20 meters.

Next, we specify a domain of endurants:

```
datatype endurant =
      Sphere1
    | Sphere2
    | Sphere1Radius10M
    | Sphere2Radius20M
    | Sphere1Dist200M
    | Sphere2Dist200M
```

The endurants are:

- *Sphere1*: the first sphere;
- *Sphere2*: the second sphere;
- *Sphere1Radius10M*: a moment representing the first sphere having a 10 meter radius;
- *Sphere2Radius20M*: a moment representing the second sphere having a 20 meter radius;
- *Sphere1Dist200M*: a moment representing the first sphere being 200 meters from the second sphere;
- *Sphere2Dist200M*: a moment representing the second sphere being 200 meters from the first sphere.

We specify our set of possible worlds using the following function, to construct configurations where there are only two possible worlds, one of which is the empty world and the other having all the endurants:

```
definition tw_worlds ::
    ‹endurant set ⇒ endurant set set›
  where
```

```
⟨tw_worlds X = {∅,X}⟩
```

The next function creates an inherence relation predicate from a base set of endurants. It associates the first sphere (*Sphere1*) to the moments *Sphere1Radius10M* and *Sphere1Dist200M* and the second sphere (*Sphere2*) to the moments *Sphere2Radius20M* and *Sphere2Dist200M*. The associations can be restricted by the function's *X* parameter: associations are only made between endurants that are in *X*.

```
definition tw_inheres_in ::
    ⟨endurant set ⇒ endurant ⇒ endurant ⇒ bool⟩
  where
    ⟨tw_inheres_in X y x ⟷
       x ∈ X ∧ y ∈ X ∧
       (x = Sphere1 ∧ (y = Sphere1Radius10M ∨ y = Sphere1Dist200M) ∨
        x = Sphere2 ∧ (y = Sphere2Radius20M ∨ y = Sphere2Dist200M))⟩
```

This function creates an quale association predicate from a base set of endurants. It associates the quales *Q_Radius10M* and *Q_Radius20M* to the moments *Sphere1Radius10M* and *Sphere2Radius20M*, respectively, and the quale *Q_Dist200M* to the moments *Sphere1Dist200M* and *Sphere2Dist200M*. The associations can be restricted by the function's *X* parameter: associations are only made between endurants that are in *X*.

```
definition tw_assoc_quale ::
    ⟨endurant set ⇒ endurant ⇒ quale ⇒ bool⟩
  where
    ⟨tw_assoc_quale X x q ⟷ x ∈ X ∧
        (x = Sphere1Radius10M ∧ q = Q_Radius10M ∨
         x = Sphere1Dist200M ∧ q = Q_Dist200M ∨
         x = Sphere2Radius20M ∧ q = Q_Radius20M ∨
         x = Sphere2Dist200M ∧ q = Q_Dist200M) ⟩
```

This function creates an towardness predicate from a base set of endurants. It associates the substantials *Sphere2* and *Sphere1* to the moments *Sphere1Dist200M* and *Sphere2Dist200M*, respectively. The associations can be restricted by the function's *X* parameter: associations are only made between endurants that are in *X*.

```
definition tw_towards ::
  ⟨endurant set ⇒ endurant ⇒ endurant ⇒ bool⟩
    where
  ⟨tw_towards X x y ⟷
     x ∈ X ∧ y ∈ X ∧
     (x = Sphere1Dist200M ∧ y = Sphere2 ∨
      x = Sphere2Dist200M ∧ y = Sphere1)⟩
```

The following is the definition of the maximal set of quality spaces that shall be used in the examples:

```
definition tw_all_quality_spaces ::
  ‹quale set set›
    where
  ‹tw_all_quality_spaces ≡
    { { Q_Dist200M},
      {Q_Radius10M,Q_Radius20M} }›
```

The actual set of quality spaces used in each example is given by the following function, which restricts `tw_all_quality_spaces` to include only the quales that are associated with a moment that is present in the configuration (given by the parameter `X`):

```
definition tw_quality_spaces ::
  ‹endurant set ⇒ quale set set›
    where
  ‹tw_quality_spaces X =
     { Q | Q x q . Q ∈ tw_all_quality_spaces ∧
                   tw_assoc_quale X x q ∧ q ∈ Q}›
```

The configuration of each example is constructed by the following function, which has a single parameter, `X`, that specifies the set of endurants present in the configuration:

```
definition tw_part_structure ::
  ‹endurant set ⇒ (endurant,quale) particular_struct›
    where
  ‹tw_part_structure X ≡ ⦇
      ps_quality_spaces = tw_quality_spaces X,
      ps_worlds = tw_worlds X,
      ps_inheres_in = tw_inheres_in X,
      ps_assoc_quale = tw_assoc_quale X,
      ps_towards = tw_towards X ⦈›
```

This is the set of endurants in the domain:

```
definition ‹all_endurants ≡
        {Sphere1,Sphere1Radius10M,Sphere1Dist200M,
         Sphere2,Sphere2Radius20M,Sphere2Dist200M}›
```

This is the set that includes only the substantials:

```
definition
  ‹just_the_substantials ≡ {Sphere1,Sphere2}›
```

The first configuration is a particular structure that includes all endurants, including all the moments with their associations: [7]

```
definition ‹conf1 ≡ tw_part_structure all_endurants›
```

The second configuration is similar to the first, but excludes the moments that represent the radii of the spheres:

```
definition ‹conf2 ≡
    tw_part_structure (
      all_endurants - {Sphere1Radius10M, Sphere2Radius20M}
    )›
```

The last configuration only includes the substantials, excluding all moments:

```
definition ‹conf3 ≡
    tw_part_structure just_the_substantials
  ›
```

The three configurations satisfy the axioms for an UFO particular structure:

```
theorem conf1_particular_struct[simp,intro!]:
    ‹particular_struct conf1›

theorem conf2_particular_struct[simp,intro!]:
    ‹particular_struct conf2›

theorem conf3_particular_struct[simp,intro!]:
  ‹particular_struct conf3›
```

4.6 Final Considerations

In this chapter, we introduced the notion of UFO particular structures and UFO particular structure morphisms, as elements that form a category in the categoric-theoretic sense.

We also presented definitions of some structural properties of particulars that are useful in the analysis of the concepts of identity and individuality: *isomorphically unique particulars*, *permutable* particulars and *non-collapsable particulars*. Furthermore, we presented theorems that prove their logical relationships.

[7] The full Isabelle/HOL code for this section can be found in **??**.

We end the chapter in section 4.5 by providing a few examples of particular structures and particular structure morphisms based on Max Black's Twin Spheres example (see (BLACK, 1952)).

The formal approach presented in this chapter can be adapted for use in the context of other foundational ontologies, by specifying a suitable notion of models of the foundational ontology's theory of particulars and a suitable notion of meaning-preserving morphisms.

The concepts presented in this chapter comprise the theoretical basis in which the theories presented in chapter 5 and chapter 6 are constructed upon.

5 Individuality and Identifiability

5.1 Introduction

This chapter presents a formal analysis of the concepts of Identity and Individuality in the context of the UFO theory of particulars, using the framework defined in chapter 4.

The goal is to present definitions for theses concepts that capture their intuition and that rely solely on the elements introduced in chapter 3 and chapter 4. In other words, these definitions should provide the means for asserting whether or not a particular has or not "identity" and "individuality", given a configuration represented by a particular structure.

For example, given the examples presented in section 4.5, it should be possible to determine, objectively, whether or not the spheres posses "identity" and "individuality" in each sample structure. We achieve this goal by presenting *structural* definitions for these concepts, based on the structural properties of particulars presented in chapter 4.

The chapter proposes a definition of what does it mean for a particular to possess *individuality* in a particular structure (in section 5.2), as being those that are not collapsable, according to the definition of (non-)collapsability presented in subsection 4.4.3.

The intuition here is that the existence of an endomorphism that "merges" two or more particulars into a single one in its image is an evidence that the structure provides no grounding for asserting that theses particulars are, in fact, two distinct individuals.

For the concept of identity, this chapter presents three distinct but ultimatelly equivalent characterizations.

The first one, described in subsection 5.3.1 and subsection 5.3.2, characterizes a particular that possess identity as being one for which a suitable definite description predicate exists, i.e. a predicate that is able to "pick" exactly that particular in the particular structure that is being considered as a context and that also picks the corresponding particular in all particular structures that are isomorphic to the first one.

The second one considers particulars that possess identity as being those that are *isomorphically unique*, or, equivalently, that are *non-permutable*, according to the respective definitions provides in subsection 4.4.1 and subsection 4.4.2.

The proof that the set of particulars that fit the first characterization is identical to those that fit the second one is provided in subsection 5.3.3.

Based on the later characterization, we present in subsection 5.3.4 the notion of

the correpondence function of a particular (in a particular structure), capturing the idea of the identity of that particular in a class of isomorphic particular structures or of an particular's identity across all possible representations of a particular structure .

The third characterization of particulars that possess identity is made though the notion of identity anchors in subsection 5.4.1. An anchor for a particular in a particular structure captures the idea of a context that is sufficient for fixing the identity of a particular in particular structure. A particular that has at least one anchor in a particular structure is called a *anchored particular*. In **??** we show that the set of anchored particulars is identical to the sets that correspond to the first two definitions of particulars with identity.

In addition to the definitions of identity provided in this chapter, we also present a classification of identity anchors, in section 5.6, i.e. the classes of minimal, finite, infinite, intrinisc and extrinsic anchors, which can be used to characterize particulars whose identity is anchored in a finite context, or those whose identity can be characterized solely based on intrinsic properties.

The notion of anchoring is also used, in subsection 5.6.3, to present relationships of *identificational dependency* between particulars that can serve as tools for studying the identity conditions of particulars.

5.2 Individuality [1]

In (GUIZZARDI, 2005), Guizzardi explains that the version of UFO presented there "exclude the so-called quasi-objects, i.e., individuals that have determinate countability but indeterminate identity". The reason given for such decision is that there is an "associated belief that quasi-objects do not exist in the macroscopic level". However, in sequence, Guizzardi admits that quasi-objects could be included by extending the theory appropriately.

In this section, we provide a formal definition for what a quasi-object, or what we call here a *particular with individuality* is, or what differentiates it from a particular that lacks individuality. The approach here is not to characterize a particular with individuality as one for which a "determinate countability" exists (GUIZZARDI, 2005), but, instead, to characterize the *individuality* as a structural property of the particular, i.e. a consequence of existing in a particular configuration of things.

First, lets define the particulars with individuality in a UFO particular structure as being exactly those particulars that are *non-collapsable*:

[1] The full Isabelle/HOL code for this section can be found in section D.1.

abbreviation `particularsWithIndividuality` ($\langle\mathcal{P}_{ind}\rangle$) **where**
$\langle\mathcal{P}_{ind} \equiv \mathcal{P}_{nc}\rangle$

To see what this definition implies, consider two simple examples, based on Black's twin spheres example (BLACK, 1952):

1. Let U_1 be an universe in which spheres S_1 and S_2 exist. For all intentions and purposes, S_1 and S_2 have the same intrinsic properties, e.g. radius, color, mass, etc. No material relation is recorded between the two spheres;

2. Let U_2 be an universe in which spheres S_1 and S_2 exist. For all intentions and purposes, S_1 and S_2 have the same intrinsic properties, e.g. radius, color, mass, etc. The spheres S_1 and S_2 are 1 km apart from each other.

In the first configuration spheres S_1 and S_2 are *collapsable*, i.e. they can be mapped to the same particular (e.g.S_1) by a particular structure morphism that preserves the formal properties of the particulars. In other words, there is no evidence, in the first example, that spheres S_1 and S_2 are in fact, distinct, since no formal property is lost by considering them to be the same.

Contrast this with the second example, for which no such morphism can be found, due to the existence of a distance relation (of 1km) between the spheres: if we consider them to be a single sphere, we arrive at an impossible situation of having a physical object being 1km apart from itself.

5.3 Identity Through Permutability [2]

In this section, we describe the formal characterization of the identity of particulars in UFO through the notion of permutability introduced in chapter 4. We also present a characterization based on the existence of an identifying predicate and prove its logical equivalence to the definition based on permutability.

5.3.1 Isomorphically-Invariant Identity Predicates [3]

Before we present the characterization of identity through the structural notion of permutability, we first characterize the usual notion of the identifiability of a particular, as the existence of a predicate, here called an identity predicate, that satisfies some requirements.

[2] The full Isabelle/HOL code for this section can be found in section D.2.
[3] The full Isabelle/HOL code for this section can be found in subsection D.2.1.

Given a particular structure Γ and a particular *x* of Γ, we say that a binary predicate *P* over the set of particular structures and the set of particulars is an identity predicate for *x* of Γ if it satisfies the following conditions:

1. it identifies *x* in Γ, i.e., it is only satisfied by *x* in Γ;
2. even if we change the representation of particulars in Γ, i.e. by an isomorphism to another structure Γ', the particular that corresponds to *x* in Γ' is still identifiable by *P*.

The reason we require invariance under isomorphisms (or change of representation) is that the predicate should rely only on the structure of Γ and not in the particularities of whatever representation is being used for particulars in Γ. For example, suppose we represent particulars as integers and suppose *x* is represented as the integer *12*. A predicate defined as $P\ \Gamma\ y \equiv y = 12$ would surely identify *x* in Γ. Furthermore, it identifies *x* without using any properties or relations provided by the UFO theory of particulars (inherence, quale association, towardness). However, this predicate would fail in case we change the representation by using, for example, the morphism $\varphi\ x = x + 1$.

However, if the predicate "fixes" *x*'s identity by using only the properties and relations provided by the UFO theory of particulars, which compose the UFO particular structure, then it is invariant under isomorphisms. For example, suppose the same *x* in the previous example bears a moment *m* associated with a certain quale *q*. Suppose also that *x* is the only particular in Γ that bears a moment associated to *q*. In that case, the predicate $P\ \Gamma\ y \equiv \exists z.\ z \lhd_\Gamma y \wedge z \longrightarrow q$ is an (isomorphically invariant) identifying predicate for *x* of Γ.

To express this definition in Isabelle/HOL we face a challenge due to the limited polymorphism provided by the Isabelle/HOL type system, which does not allow us to quantify over types (and thus, over representations for particulars). Instead, we require the predicate to be invariant under isomorphisms from Γ to particular structures built with Zermelo-Fraenkel sets as particular representations. We do so assuming that the Isabelle/HOL type of ZF sets is sufficiently large to include an infinite number of sets that are isomorphic to the set of particulars of Γ.

Formally, we have an identity predicate defined as:

```
definition identity_pred ::
  ‹('p,'q) particular_struct ⇒
   ((ZF,'q) particular_struct ⇒ ZF ⇒ bool) ⇒ 'p ⇒ bool›
  where ‹
    identity_pred Γ P x ⟷ x ∈ particulars Γ ∧
      BijMorphs1_{Γ,TYPE(ZF)} ≠ ∅ ∧
```

```
(∀Γ' ∈ IsoModels_{Γ,TYPE(ZF)}.
  ∀φ ∈ Morphs_{Γ,Γ'}.
  ∀y. P Γ' y ⟷
    (∀z ∈ particulars Γ. y = φ z ⟷ z = x))›
```

As expected, an identity predicate is capable of "picking" the correspondent of x under an isomorphism:

```
lemma identity_respects_isomorphisms:
  assumes
    ‹identity_pred Γ P x›
    ‹φ ∈ BijMorphs1_{Γ,TYPE('p_1)}›
  shows ‹identity_pred (MorphImg φ Γ) P (φ x)›
```

5.3.2 Identifiable Particulars [4]

With the definition of an identity predicate, we can define the concept of an *identifiable particular* of a particular structure as being one for which an identifying predicate exists:

```
definition identifiables (‹𝒫_=›) where
  ‹𝒫_= ≡ { x | x P . x ∈ 𝒫 ∧ identity_pred Γ P x }›
```

If we take Isabelle/HOL to be an example of a formal logic with the power to express any common-sensical identifying predicate using UFO particular properties and formal relations, then we can consider the definition of identifiable particulars as a definition for what a *particular with identity* is, using an approach that requires the existence of a predicate.

5.3.3 Identifiability, Isomorphical Uniqueness and Permutability [5]

While we introduced, in the previous section, a definition for what consists the "identity" of a particular in terms of the existence of a suitable *identifying predicate*, in this section we show that it is not necessary to refer to the existence of an element that is ontologicaly extraneous to the UFO particular structure (the identifying predicate), to characterize the concept of the "identity" of a particular.

[4] The full Isabelle/HOL code for this section can be found in subsection D.2.2.
[5] The full Isabelle/HOL code for this section can be found in subsection D.2.3.

We achive this by proving the logical equivalence between the notion of identifiability, as described in the previous section, and the notions of isomorphical uniqueness and non-permutability, that were introduced in subsection 4.4.1 and subsection 4.4.2.

The equivalence of the set of isomorphically-unique particulars and the set of identifiable particulars is shown by the following lemmas and theorem:

lemma *im_uniques_are_identifiables:* ‹$\mathcal{P}_{\simeq!} \subseteq \mathcal{P}_{=}$›

lemma *identifiables_are_im_uniques:* ‹$\mathcal{P}_{=} \subseteq \mathcal{P}_{\simeq!}$›

theorem *identifiables_are_the_im_uniques:* ‹$\mathcal{P}_{=} = \mathcal{P}_{\simeq!}$›

Since the set of isomorphically unique particulars is equivalent to the set of non-permutable particulars, then we can also say that the identifiable particulars are exactly those that are non-permutable:

theorem *identifiables_are_the_non_permutables:* ‹$\mathcal{P}_{=} = \mathcal{P}_{1!}$›

5.3.4 Correspondence Function [6]

We can leverage the ability of an identifying predicate to pick up a particular of a particular structure in any isomorphic structure to construct a representation-independent notion of the "identity of the particular". We achieve that by using the *correspondence function* (π), which, for any particular structures Γ_1 and Γ_2, and any particular *x* of Γ_1, gives the element *y* of Γ_2 that is uniquely mapped by any Γ_1-Γ_2 isomorphism (assuming *x* is isomorphically unique).

In fact, the way we define π is by choosing any bijection between Γ_1 e Γ_2 and using it to identity the particular that corresponds to *x* in Γ_2, since *x* is isomorphically invariant:

definition π *::* ‹*'p*$_1$ $\Rightarrow$ *'p*$_2$› **where**
‹$\pi \equiv$ *SOME* σ. *particular_struct_bijection_1* Γ_1 $\sigma \wedge \Gamma_2$ = *MorphImg* σ Γ_1›

Similarly, we can define, for any isomorphically unique particular *x* of Γ, a function that, given any particular structure Γ*'* isomorphic to Γ, identifies the correspondent of *x* in it. We call this function Γ*_corresp* and define it formally as:

definition Γ*_corresp ::* ‹*'p* $\Rightarrow$ *('p*$_1$*,'q) particular_struct* $\Rightarrow$ *'p*$_1$› **where**

[6] The full Isabelle/HOL code for this section can be found in subsection D.3.1.

```
⟨Γ_corresp x Γ' ≡
    let φ = (SOME σ. particular_struct_bijection_1 Γ σ ∧ Γ' = MorphImg σ Γ)
    in φ x ⟩
```

Now, given some `x` in Γ_1, `Γ_corresp x` is a function from particular structures (of any particular type) to particulars of that structure. This function represents the identity of `x` in all particular structures isomorphic to Γ, independent of the specific representation of the particulars in them.

We propose that this function, `Γ_corresp x`, is a good structurally-invariant representation of the identity of `x`.

5.4 Identity Through Identity Anchors

In this section, we describe another approach for characterizing the identity of a particular in a particular structure. Though we prove that this approach is logically equivalent to the characterizations based on identifiability, non-permutability or isomorphical uniqueness, it nevertheless is able to better highlight the context that characterizes the identity of a particular.

5.4.1 Anchoring [7]

The value of an ontology, or conceptual model, lies in the information it carries about the concepts and assumptions that characterize a domain. One of the most important users of a conceptual model, in the context of the development of Information Systems, is the database designer, and some of the informations the DB designer expects to find in the conceptual model are the identity conditions of the elements in the domain.

The definitions provided so far for the identity of particulars were (1) by identifiability, (2) by isomorphical uniqueness, and (3) by non-permutability. The first one has the disadvantage of requiring the existence of an predicate (and of a formal language) at a fundamental level in the foundational ontology, even though such predicate would be useful for the purposes of the DB designer. The other two, though not requiring the existence of elements that are not in the particular structure, simply provides us a yes or no answer to whether a particular has identity.

Here we introduce the notion of an *identity anchor* as a structure that represents the *identity neighborhood* of a particular in a particular structure, i.e., the elements of the structure that play a role in the identification of the particular. Note that we are not

[7] The full Isabelle/HOL code for this section can be found in subsection D.5.1.

referring to the identity condition itself, which would be a predicate, but to the elements of the structure that would participate in some identity predicate.

Given a particular structure Γ and a particular x of Γ, and given another particular structure Γ_x, a particular y of Γ_x, and a morphism φ from Γ_x to Γ, we say that (Γ_x, y, φ) anchors x in Γ, written as $y \ -\Gamma_x,\varphi\rightarrow_1 \ x$, or that (Γ_x, y, φ) is an anchor for x (in Γ), if and only if, for every morphism σ from Γ_x to Γ, $\sigma \ y$ is always x.

In other words, there are sufficient elements in Γ_x to make it so that y (in Γ_x) cannot be seen as anything but x in Γ. Formally, we have:

definition *anchors* ::
  ‹'p_2 ⇒ ('p_2,'q) particular_struct ⇒
   ('p_2 ⇒ 'p) ⇒ 'p ⇒ bool›
   (‹_ −_,_→$_1$ _› [74,1,1,74] 75) **where**
‹y −Γ_x,φ→$_1$ x ≡
   x ∈ $\mathcal{P}$ ∧ Γ_x ≪$_\varphi$ Γ ∧ y ∈ particulars Γ_x ∧
   (∀σ ∈ Morphs$_{\Gamma_x,\Gamma}$.
    ∀z ∈ particulars Γ_x. σ z = x ⟷ z = y)›

Note that, since x is invariant with respect to the morphisms from Γ_x to Γ, the choice of the morphism φ doesn't matter. Thus, we can just say that (Γ_x, y) anchors x, or simply that Γ_x is an anchor for x. Note that from a particular structure with a single particular, y, we can always have a morphism to Γ that maps y to x. However, this configuration would only work as anchor for x if x is the only substantial in Γ. Otherwise, there would be morphisms from the single-particular structure to any substantial in Γ. Thus, it always possible to remove enough elements from an anchor in such a way that it stops being an anchor. Conversely, if Γ' is an anchor for x, then the addition of new elements to Γ', while maintaing the existence of at least one morphism to Γ, will not remove its status as an anchor for x.

definition *anchored_particulars* :: ‹'p set› (‹$\mathcal{P}_\downarrow$›) **where**
  ‹$\mathcal{P}_\downarrow$ ≡ { x | x (y :: ZF) Γ_x φ . y −Γ_x,φ→$_1$ x }›

The relationship between anchoring and the definitions of identity presented so far is given by the following lemmas and theorem, which show that being anchored (having at least one anchor) is logically equivalent to being non-permutable and, thus, that anchoring can be used interchangeably with the other definitions:

lemma *non_permutable_are_anchored:* ‹$\mathcal{P}_{1!} \subseteq \mathcal{P}_\downarrow$›

lemma *anchored_are_non_permutable:* ‹$\mathcal{P}_\downarrow \subseteq \mathcal{P}_{1!}$›

theorem *anchored_are_the_non_permutable:* ‹$\mathcal{P}_{\downarrow}$ = $\mathcal{P}_{1!}$›

5.5 Identity Characterization Equivalences

Here is a summary of the logical equivalences between the concepts that can be used to characterize identity:

theorem *identifiable_particulars_are_the_non_permutables:*
‹$\mathcal{P}_{=}$ = $\mathcal{P}_{1!}$›

theorem *identifiable_particulars_are_the_isomorphically_unique:*
‹$\mathcal{P}_{=}$ = $\mathcal{P}_{\simeq!}$›

theorem *identifiable_particulars_are_the_anchored_particulars:*
‹$\mathcal{P}_{=}$ = $\mathcal{P}_{\downarrow}$›

5.6 Anchoring Classes and Identification Dependence [8]

In this section, we propose a classification of the anchors of particulars of a particular structure and a definition of some relations that express various degrees of *identity dependence*.

5.6.1 Anchoring Classes [9]

The anchors of particulars of a structure can be classified into several classes. The first is a *minimal anchor*: we say that an anchor Γ_x for `x` in Γ is *minimal* if and only the only other anchors for `x` in Γ that are substructures of Γ_x, i.e. that have an injective morphism to Γ_x, are those that are isomorphical to Γ_x. In other words, a minimal anchor represents a minimal context for the identification of a particular. Formally:

definition *minimallyAnchors ::*
‹'p_2 ⇒ ('p_2,'q) particular_struct ⇒
('p_2 ⇒ 'p) ⇒ 'p ⇒ bool›
(‹_ −_,_→⊥ _› [74,1,1,74] 75)
where

[8] The full Isabelle/HOL code for this section can be found in section D.6.
[9] The full Isabelle/HOL code for this section can be found in subsection D.6.1.

```
‹y −Γx,φ→⊥ x ≡
  y −Γx,φ→1 x ∧
  (∀Γ' z σ1 σ2.
      Γ' ≪σ1 Γx ∧ z −Γ',σ2→1 x ⟶
      Γ' ∈ IsoModelsΓx,TYPE(ZF))›
```

We say that a particular is minimally anchored just in case there is at least one minimal anchor for it:

```
definition minimallyAnchored :: ‹'p set› (‹𝒫⇓›)
  where
  ‹𝒫⇓ ≡ { x | x (y :: ZF) Γx φ . y −Γx,φ→⊥ x }›
```

Similarly, we say that a particular is finitely anchored if there is at least one finite anchor for it, i.e., an anchor that has a finite set of particulars. In other words, the existence of a finite anchor for a particular is the evidence that it's possible to identify finitely, e.g. using a finite predicate in first-order logic.

```
definition finitelyAnchored :: ‹'p set› (‹𝒫fin↓›) where
  ‹𝒫fin↓ ≡ { x | x (y :: ZF) Γx φ. y −Γx,φ→1 x ∧
                finite (particulars Γx) }›
```

We say that a particular is intrinsically anchored if and only if it is anchored by at least one anchor that only has a single substantial (but which may have many moments directly or indirectly inherent to the single substantial). The existence of such an anchor is the evidence that the particular can be identified without referring to other substantials, i.e., by referring only to its intrinsic properties. Formally, we have:

```
definition intrinsicallyAnchored :: ‹'p set› (‹𝒫int↓›)
  where
  ‹𝒫int↓ ≡ { x | x (y :: ZF) Γx φ. y −Γx,φ→⊥ x ∧
    (∀z ∈ particulars Γx. (ps_inheres_in Γx)** z y) }›
```

Finite anchors imply the existence of minimal anchors, i.e, finitely anchored particulars are also minimally anchored ones:

```
lemma finitely_anchored_are_minimally_anchored: ‹𝒫fin↓ ⊆ 𝒫⇓›
```

5.6.2 Anchoring Sets [10]

In this section, we describe a series of sets extracted from the classes of anchors for a particular, as defined in the previous section.

An anchoring set for a particular x in Γ are simply the image (in Γ) of sets of particulars of anchors for x. They represent the identity neighborhood of x in the context of the other particulars of the structure. For example, if a particular x that represents a human can be anchored by a structure that contains a substantial y that anchors x and other two substantials, representing the parents of y, with corresponding relational moments representing the parental relationship, we can say that the set consisting of the parents of x is an anchor set for x. Formally, we have:

definition *anchorSets* :: ‹'p ⇒ 'p set set› (‹$\mathcal{A}$_› [1] 1000)
where
‹$\mathcal{A}_x$ ≡ { φ ‘ particulars Γ_x | φ Γ_x (y :: ZF) . y $-\Gamma_x,\varphi\rightarrow_\perp$ x }›

Mirroring the classes defined in the previous section, we have the following types of anchor sets, representing finite and intrinsic anchors, respectively:

definition *finiteAnchorSets* ::
‹'p ⇒ 'p set set›
(‹$\mathcal{A}^{fin}$_› [1] 1000)
where
‹$\mathcal{A}^{fin}{}_x$ ≡ {X . X ∈ $\mathcal{A}_x$ ∧ finite X}›

definition *intrinsicAnchorSets* ::
‹'p ⇒ 'p set set›
(‹$\mathcal{A}^{int}$_› [1] 1000)
where
‹$\mathcal{A}^{int}{}_x$ ≡ {X . X ∈ $\mathcal{A}_x$ ∧ (∀y ∈ X. (◁)** y x)}›

For example, supposing every human being could be identified solely by the DNA sequence, and supposing that we represent the association of a human being to a DNA sequence through a moment associated with a quale that represents the sequence, any human being x with DNA sequence S could be anchored by a particular structure consisting in a substantial and a moment associated with S. Thus, x and its moment that is associated with the DNA sequence S are both finite anchor sets and intrinsic anchor sets.

We call the *identity core* of x the set of particulars that appear in all anchor sets of x. Such set, if non-empty, represent the elements of the domain that are necessary, though not always sufficient, to identify x. For example, if for all human beings in the particular

[10] The full Isabelle/HOL code for this section can be found in subsection D.6.2.

structure, a moment associated with a DNA sequence appears in his or hers identity core, then we can deduce that the association with a DNA sequence is strictly necessary for the identification of human beings (in the particular structure). Formally we have:

definition *identityCore ::*
  ‹*'p ⇒ 'p set*›
  (‹*IdCore _*› *[1] 999)*
  where
    ‹$IdCore_x \equiv \bigcap \mathcal{A}_x$›

We widen the identity core by intersecting only the anchor sets that are intrinsic or finite. These would indicate the particulars that are necessary for the identification, if the identification is to be done intrisically or in a finite way:

definition *intrinsicIdentityCore ::*
  ‹*'p ⇒ 'p set*›
  (‹*IntIdCore _*› *[1] 999)*
  where
    ‹$IntIdCore_x \equiv \bigcap \mathcal{A}^{int}{}_x$›

definition *finiteIdentityCore ::*
  ‹*'p ⇒ 'p set*›
  (‹*FinIdCore _*› *[1] 999)*
  where
    ‹$FinIdCore_x \equiv \bigcap \mathcal{A}^{fin}{}_x$›

5.6.3 Identity-Related Dependency Relations [11]

Using the concept of anchor sets, we can define a notion of *identification dependence*. We call *y* an idenfication dependence of *x*, written as *IdDep x y*, just in case all anchoring sets of *y* are also anchoring sets of *x*. In other words, it is impossible to identify *x* without also identifying *y*. If the converse does not hold, then we can say that it is necessary to identify *y* before indentifying *x*. Formally, we have:

definition *identification_dep ::* ‹*'p ⇒ 'p ⇒ bool*› (‹*IdDep*›)
  where ‹$IdDep\ x\ y \longleftrightarrow (\forall X \in \mathcal{A}_x.\ \exists Y \in \mathcal{A}_y.\ Y \subseteq X)$›

[11] The full Isabelle/HOL code for this section can be found in subsection D.6.3.

Identification dependence is a *pre-order*, i.e. it is a reflexive and transitive relation:

lemma *identification_dep_refl[intro!]:* ‹*IdDep x x*›

lemma *identification_dep_trans[trans]:*
  assumes ‹*IdDep x y*› ‹*IdDep y z*›
  shows ‹*IdDep x z*›

5.7 Final Considerations

In this Chapter we presented a novel definition for the concepts of Identity and Individuality in UFO, based on the framework presented in chapter 4.

The notion of individuality was characterized through the concept of non-collapsable particulars.

For the notion of identity, three alternate characterizations were provided: the first is one that refers to the existence of a suitable identifying predicate; the second is one that characterizes identity through the non-permutability or isomorphical uniqueness of a particular; and the last one characterizes identity through the existence of identity anchors.

By showing the logical equivalence between the three definitions, the chapter proved that a linguistic characterization of identity (through identity predicates) can be replaced by a structural characterization (through non-permutability or anchoring).

Besides proving the adequacy of the structural definitions, this chapter also presented a classification of anchoring contexts that can be used to characterize the concepts of finite, intrinsic and extrinsic identity.

It also presents the concept of mininmal anchors, which serves as a tool to investigate the identity conditions for a particular.

The chapter also presented two concepts of identification dependency: weak and strong identification dependency. These relations can be used to model the identity relations between particulars, e.g. between a human being and its DNA sequence.

6 Sortality

6.1 Introduction

The concept of a *sortal universal* plays an important role in UFO's ontology of substantial universals. The importance of sortals in UFO comes from the following facts:

1. UFO assumes that every substantial in the domain of discourse "possess" identity;
2. UFO assumes that the identity of a substantial is "supplied" by a unique sortal universal it is an instance of, called a *substance sortal* or *ultimate sortal*;
3. if a substantial universal is a sortal then all the universals that subsume it are also sortals;
4. if a substantial universal is not a sortal, then all the universals subsumed by it are also non-sortals.

The consequence of these facts is that, when working with UFO as a foundational ontology, for a conceptualization to be considered minimally complete, for every substantial in the domain of discourse there must be at least one sortal in the conceptualization it is an instance of. These assumptions and logical consequences thereof allows the conceptual modeler to identity several problems in a conceptual model base in UFO, such as in an OntoUML model:

- non-sortals must not be sub-classes of sortals;
- if two sortals have a common sub-class, then they must have a common super-class that is also a sortal;
- there should be no example substantial in the domain that is being modeled that does not instantiate at least one sortal in the model;

Although these constraints are considered useful in the process of conceptual modeling, their application depends on whether the judgment of the sortality of a substantial universal is well-defined. However, contrary to some other important concepts in UFO, such as rigidity, existential dependency, mereological complexity, etc., the notion of sortality is not formally defined and its informal definition is based on the existence of a principle of identity as a special kind of predicate, tying up a fundamental property of substantial universals to a linguistic concept and presuming, implicitly, the existence or even a choice

of a formal language in which the predicate is expressed. As a consequence, the evaluation of the sortality of a substantial universal can be dificult in practice.

The goal of this chapter is to present a definition for the notion of sortality in the UFO that is (1) completely formal, (2) independent of linguistic concepts, and (3) completely determinable from the elements that comprise the models of the theory of UFO particulars. The chapter starts with an informal discussion about the approach used in the characterization, followed by a formal theory, in Isabelle/HOL, that defines the notion of sortality that satisfies these requirements.

6.2 Sortals: a change in perspective

The approach used in the definition of sortality presented in this chapter is based on a shift in perspective with respect to the relationship between substantial universals and identity. The link between these two is usually expressed thorught the statement that sortal (substantial) universals "carry a principle of identity" for its instances. It is also present in the definition of *ultimate sortals* that characterize them as being substantial universals that "supply a principle of identity" to its instances.

This idea can be illustrated by a simple example, as follows.

Suppose, for the sake of the argument, that all human beings, but not all animals in general, can be uniquely identified by their DNA sequence; furthermore, suppose that there are no critera that can be uniformly applied to all animals to estabilish their identity.

Given these assumptions and the characterization of sortality as an universal that "carries a principle of identity"for its instances, we can say that the universals *Human Being* and *Graduate Student* are Sortals while the substantial universal *Animal* is *not*. The first two are considered sortals because there is a criteria (same DNA) that can be uniformilly applied to its instances to establish their identity. The later, however, is not, due to the assumption that no uniform criteria for identity exist for all animals.

Furthermore, the sortal *Human Being* might be considered an *ultimate sortal* if there are no other substantial universals that are strictly more abstract then *Human Being* (having at least the same instances as *Human Being*) and for whose instances there is a uniformly appliable criteria of identity.

But what does "carrying" or "supplying" means, precisely?

We can explain "to supply" in terms of "to carry" by saying that a substantial universal "supplies" a principle of identity just in case it "carries" a principle of identity while none of its proper abstractions also carry one. Thus, to find the substantial universals that supply a principle of identity, one just has to follow the ordering graph univerals (orderer by subsumption) from the least to the most abstract, starting from the least

abstract universals of the conceptualization that "carry" a principle of identity and going "up" on the graph until the most abstract universal that still "carries" a principle of identity is found.

We can also explain "to carry" in terms of "to supply", as follows: a substantial universal is said to "carry" a principle of identity for its instances just in case it subsumes a universal that "supplies" a principle of identity for its instances.

In both cases, the definitions rely on the fact that if there is a principle of identity for all instances of an universal, then this principle is also applicable to all instances of any universal that subsumes than the former, or, equivalently, whose set of instances is a proper subset of the former (in all possible worlds).

To avoid a circular (and meaningless) definition, however, it is necessary to define "to supply" or "to carry" using some strategy that does require the other.

The approach taken in UFO is to define "to supply" (ultimate sortals) through "to carry" and define "to carry" through the existence of a principle of identity.

A principle of identity for the universal U is, in this case, a binary predicate P the encodes the identity relation between the instances of U.

A first question is: can we define P in a general form?

To answer this question, we need to distinguish between *identity*, an ontological property, and *equality*, a *logical* relation. This distinction is necessary to avoid a naive (and useless) definition for P as:

$$P(x, y) \leftrightarrow x = y$$

The reason we cannot use logical equality as a synonym for identity is that in making so we introduce an (extra logical) implicit assumption that all the entities in our domain of discourse "have an identity", since logical equality applies to all entities in the domain. Since, in this case, every universal would have a principle of identity and since we expect the question of whether or not an universal has a principle of identity to be a meaningful and non-trivial question, we must discard this approach.

Another option is to use Leibniz's principle of the identity of indiscernibles:

$$(\forall P.P(x) \longleftrightarrow P(y)) \longleftrightarrow identical(x, y)$$

This principle states that two particulars can be considered identical if they are indiscernible with respect to monadic predicates. Note that this principle presumes that distinguishability in general is reducible to distinguishability using monadic predicates.

Using this principle, we can formulate P as follows:

$$P(x, y) \longleftrightarrow \forall Q.Q(x) \longleftrightarrow Q(y)$$

Again, this presumes that monadic predicates are sufficient for distinguishing the elements of the domain. We can extend this to all arities by using Second Order Logic and adding a (coutable) infinite number of axioms that capture all the distinguishability cases for each predicate arity.

In both cases, however, we are left we the following issue: P might be considered well-defined, but this definition is too strong, in the sense that (1) it applies uniformily to all elements in the domain, indiscriminately; and (2) it is not informative at all.

All formal categories and relations in a foundational ontology are expected to be informative, in the sense that their content should reveal relevant ontological properties of the elements they are applicable to. For example, suppose that the author's car and the author's car engine are represented, respectively, by the particulars p_1 and p_2, and suppose that the relationship *is a component of* is represented by a class of directed moments called $CompOfMoments$. Whether or not p_2 is, in fact, a component of p_1 can be verified by checking the existence of some moment $m \in CompoOfMoments$ such that m inheres in p_2 and is directed towards p_1. Thus, the question of whether or not p_2 is component of p_1 or, equivalently, the contents of $CompOfMoments$ cannot be determined *a priori*, but only by actually observing the configuration of reality.

Similarly, a principle of identity should reveal the specific conditions under which a class of particulars can be identified and should not be determinable *a priori*, i.e. without knowledge regarding the nature of this class of particulars.

Furthermore, the fact that this approach does not depend whatsover in which universal a particular is an instance of means that if it is to be considered the definition of a principle of identity, then it is "available" for all universals and, again, all substantial universals would be considered sortals.

What we are looking for, then, is for some predicate formula $\Delta_{x,y}$, such that

$$\forall x.\forall y.x \in Insts(U) \wedge y \in Insts \wedge \Delta_{x,y} \longleftrightarrow \forall Q.Q(x) \longleftrightarrow Q(y)$$

for monadic and, if monadic predicates are not considered sufficient to determine distinguishability, the set of corresponding formulas for each arity, where $\Delta_{x,y}$ is some *useful* and *informative* formula regarding the identity of instances of U.

But how can we determine the usefulness of $\Delta_{x,y}$ or the degree to which $\Delta_{x,y}$ is informative? Since $\Delta_{x,y}$ is a formula in some formal language $\mathcal{L}$, this answer will be highly dependent upon the syntax and semantics of the $\mathcal{L}$ itself.

Furthermore, whatever language $\mathcal{L}$ is chosen, the set of relations that can be definied in $\mathcal{L}$ is dependent on $\mathcal{L}$ itself. Should $\mathcal{L}$ be First Order Logic, Monadic Second Order Logic,

or Second Order Logic? Should it be a modal logic? If it's a (Monadic) Second Order Logic, which semantics should be considered, the standard semantics or Henkin semantics?

Should we then quantify over all formal languages, i.e. a universal is a sortal if and only if there is at least one formal language in which an informative and useful predicate can be expressed?

But, even if we arrive into an answer, say, by quantifying over all formal languages, we get into the following dillema: as a fundamental property of a purely ontological concept (substantial universals), the definition of sortality should not imply a choice of formal language, or even the existence of a formal language at all.

It is at this that the author considers, seriously, abandoning his carrer and ingressing in a more relaxing one, such as being a stock market operator.

However, before doing so, let's consider an different perspective on the problem of identity and sortals.

So far, we have reduced the question of whether or not a substantial universal is a sortal to the existence or not of a predicate expression written in some formal language that (1) capture the relation of identity of the instances of that substantial and (2) is considered adequate under some criteria of usefulness or degree of information. Given a suitable criteria for (2) this strategy would lead to a definition of the necessary and sufficient conditions for sortality.

However, this strategy has a significant issue: it reduces a question that is ontologically fundamental, in the sense that its answer should not depend on any particular categories, into a linguistic question, that depend on the existence of at least one formal language and on its particularities. Let's call the later the linguistic question for sortality.

Now, instead of pursuing an answer to the linguistic question, we can investigate whether or not we can establish some of the necessary conditions for an universal to be a sortal solely from the elements of the model of reality that is produced by the underlying foundational ontology.

A reason for doing so is that the practice of ontological investigation, as done, for example, by an ontology engineer, the direction of fit is always from the reality to the theory (or conceptual model). More specifically, given a choice of foundation ontology (such as UFO), we first filter the reality according to the structure of the foundational ontology's models, producing the intended model, and then try to find a theory that fits the intendended model as tightly as possible. Thus, the assertion of whether or not a universal is a sortal should be answerable just by examining the intended model.

Let's call the following question the structural question for sortality: given and intended model, a conceptualization for the particulars in the intended model and a

substantial universal of that conceptualization, can we prove that it is impossible to establish the identity of the instances, no matter which formal language we might use to express it?

This question is, of course, weaker than the linguist question, since a negative answer would not guarantee that the identity relation could be expressed in some formal language. However, this is good enough, because in the scope of a foundation ontology, the question of characterization of sortality should be restricted to the elements of the foundational ontology itself. Considering linguistic aspects is, in this sense, going out of the scope of a foundational ontology.

The goal of this chapter is, then, to provide a structural definition of sortality that follows the structural approach and that:

- is framed in terms of the structure of an intended model;
- reveals useful information about the conditions that rule the identity of substantials in the intended model;
- does not depend on the existence of a formal language.

The key to achieving this goal is to consider an alternative intuition for sortality. Instead of saying that "a sortal is a substantial universal that carries a principle of identity for its instances", we could say that "a non-sortal is a substantial universal that is too abstract", in the sense that the "filtering" it induces in reality produces an intended model in which it is impossible to express (no matter what language) the identity of its instances.

We can also change the perspective with respect to what having a principle of identity means: instead of being the existence a binary predicate that express the identity relation for the instances of an universal, it can be the existence of a monadic predicates (definite descriptions) that identifiy istances of the universal.

The reason for that having a way to prove whether or not two particulars are identical, using a binary predicate, is not sufficient for identifying the particulars. In practice, the user of a conceptual model, e.g. a database designer, looks for answers to questions such as "How can I identify instances of the substantial universal U?" or "Are there a set of fields derived from the properties and relations that characterize a substantial universal U that can serve as a key for the table that represents the universal U?". For that, it is not enough to be able to distinguish the instances of U, using the identity relation. What we are looking for is for the identification of each instance of U.

This chapter proposes a definition of sortality, in the context of the theory of UFO particular structures presented in chapter 4, that follows the strategy described in this

section and uses the concept of identifiability described in chapter 5 as the notion of principle of identity.

6.3 Finite Particular Structures

We present a theory of sortality in a context that imposes some restrictions on the UFO particular theory. Namely, we require that the set of particulars of the UFO particular structures be *finite* and that all the particulars posses *individuality* already, i.e. are non-collapsable.

Under this restriction, any injective morphism to an isomorphic structure is also a bijection:

lemma *mono_to_iso_is_an_iso:*
  assumes
    ‹*particular_struct_injection* Γ Γ' φ›
    ‹∃σ. *particular_struct_bijection* Γ Γ' σ›
  shows ‹*particular_struct_bijection* Γ Γ' φ›

Furthermore, any morphism to an isomorphic structure is an injection (and thus a bijection).

lemma *morphism_to_injective:*
  assumes
    ‹*particular_struct_morphism* Γ Γ' φ›
    ‹∃σ. *particular_struct_bijection* Γ Γ' σ›
  shows ‹*particular_struct_injection* Γ Γ' φ›

Consequently, all endomorphisms are bijective and, thus are permutation morphisms:

lemma *all_endomorphisms_are_perms:* ‹$EndoMorphs_{\Gamma} \subseteq Perms_{\Gamma}$›

6.4 Isomorphism-invariant Instantiation [1]

From chapter 3 to chapter 5, we mainly discussed concepts related to UFO *particulrs*, with a brief mention, on section 3.8, of a minimal theory of universals and instantiation. However, since this chapter focuses on the concept of *sortality*, which is applicable to UFO *substantial universals*, it's necessary to present a more detailed theory of instantiation.

[1] The full Isabelle/HOL code for this section can be found in section E.2.

In particular, it is necessary to adapt the usual notion of instantiation to the structural framework introduced in chapter 4. This section describes a characterization of identity that follows a strategy similar to the one used in characterizing isomorphically invariant identification predicate described in section 5.3. We use a shift in the type of particulars to hide the details of a particular representations and guarantee that the instantiation relation only relies on the formal elements that are provided in the UFO particular structure. In other words, the instantiation relation is ismorphically invariant.

The identity relation, in UFO, is a contigent relation between particulars and universals. Whether it holds or not depends also on which possible world we are considering it. On the other hand, the instantiation relation may also depend on other particulars and their properties, e.g. the universal *Being Next to a Black Cat*. Thus, we condider an instantiation predicate one that satisfies the following type:

```
type_synonym ('q,'u) iof_predicate =
  ‹(ZF,'q) particular_struct ⇒ ZF ⇒ ZF set ⇒ 'u ⇒ bool›
```

However, we also want the instantiation relation to fit into our structural approach. To do so, we require it to satisfy the following axioms, where `zf_iof::(ZF, 'a) particular_struct ⇒ ZF ⇒ ZF set ⇒ 'b ⇒ bool` is the instantiation relation and the term `zf_iof Γ x w U` denotes that "`x`, a particular of Γ, instantiates `U` in the possible world `w` of Γ":

Axiom 39. `zf_iof_scope` *If* `zf_iof Γ x w U` *holds, then* `w` *is a possible world of* Γ *and* `x` *exists in* `w`*:*

```
zf_iof Γ x w U ⟹ w ∈ ps_worlds Γ ∧ x ∈ w
```

Axiom 40. `invariant_under_isomorphisms zf_iof` *must be invariant under* Γ*-bijections:*

```
⟦φ ∈ BijMorphs1_{Γ,TYPE(ZF)};
x ∈ particulars Γ;
w ∈ ps_worlds Γ⟧ ⟹
zf_iof Γ x w U = zf_iof (MorphImg φ Γ) (φ x) (lift_world φ w) U
```

Up to this point, we defined an instantiation relation as a relation over particular structures whose type of particulars is the type of ZF sets. However, since we require the

existence of an injection from whatever type of particulars a particular structure uses to the type of ZF sets, we can define the usual ternary instantiation relation in the context of an arbitrary particular structure as:

```
definition iof (‹_ ::_ _› [74,1,74] 75) where
  ‹x ::w U ⟷
    (∃f. inj_on f 𝒫 ∧ x ∈ 𝒫
      ∧ w ∈ 𝒲
      ∧ zf_iof (MorphImg f Γ) (f x) (f ‘ w) U)›
```

In other words, we apply *zf_iof* to the particular structure that is the image of some injection from $\mathcal{P}$ to the type of ZF sets, which gives rise to an particular structure that is isomorphic to Γ.

The characterization of instantiation through an isomorphically invariant relation satisfies the aixoms of an instantiatio theory (section 3.8):

```
sublocale
  iso_instantiation ⊆ instantiation_thy
  where iof = iof
```

6.5 Universal Trimming Operation

The basic intuition behind our characterization of sortality is that the main "function" of an universal (a concept) is to abstract away details from the reality. In this sense, an universal is not characterized as something that adds something to reality, e.g. it does not "supply" an identity principle, as much as something that subtracts from reality.

As abstract entities, universals are completely determined by their formal properties. More specifically, they are completely determined by their application conditions. In other words, if the application conditions of two universals are logically equivalent, then they are identical.

Moreover, since there can be many distinct application conditions that are logically equivalent, we can say that UFO universals are determined by the instantiation relation denoted by this class of conditions.

What do we mean, however, when we refer to an "application condition"? At first glance, it could be simply a formal logic expression denoting an intensional instantiation

relation, e.g. a particular x is an instance of U in the possible world w if and only if such and such condition is true.

However, this approach suffers from the sames issues regarding the reliance on the existence of languages that identification predicates do:

- it implies that universals can only be understood in an ontology that also presupposes the existence of formal languages[2];
- it is too weak, in the sense that even if an universal's application condition can be expressed in a simple and clear way, this approach allows also the same universal to be characterized by expressions that are much more complex or confusing;
- it has more of a descriptive role than an explanatory role: given a choice of expression, it might not be clear why that choice was made, since many others could be equally valid.

Ideally, we should characterize universals by means of the properties that are already present in the structure of UFO particular models, without introducing formal languages in the domain of discourse. One alternative is to characterize universals by their extensions in each possible world, but such a characterization could hardly be considered informative or explanatory, since we don't usually grasp universals by listing their potential instances.

Fortunately, UFO already provides a notion that gives us an adequate alternative: the characterizing relation between substantial universals and moment universals. Given a substantial universal U and a moment universal u, we say that U is characterized by u, or that u characterizes U, if and only if whenever a substantial instantiates U, it also bears a moment that instantiates u. For example, the substantial universal *human being* is characterized by the moment universal *skin color*, i.e. every human being has a skin color. The set of all characterizing universals of a substantial universal U is called the characterizing set of U.

Characterizing sets are not necessarily unique, given a a particular structure and an instantiation relation. However, a case in which two distinct substantial universals (having distinct instantiation relations) have the same characterizing set can be considered a form of descriptive incompleteness: the fact that a person is able to distinguish these two universals means that at least one extra property could be added to the ontology, in the form of an extra moment universal and its instances, that can explain the distinction.

Thus, if we assume that characterizing sets are unique, we can invert their role, from being a description of a universal's properties to being a determiner of the universal

[2]note that this is not the same as affirming that we need a language to talk about an ontology

instances. In other words, the characterizing set can play the role of an application condition: two substantial universals are considered identical if and only if they have the same characterizing sets, and a substantial is considered an instances of an universal *U* in a world *w* if and only if it bears, in *w*, at least one moment corresponding to each moment universal in the characterizing set of *U*.

With unique characterizing sets as application conditions, we can also show how the determining property of a substantial universal, i.e. its application condition, explains the way in which a universal "substracts" from the reality. The idea is that the characterizing set determines the assumptions one is allowed to make, given the knowledge that a particular instantiates a certain universal. In other words, the characterizing set represents the "filter" that a universal imposes on reality.

For example, consider the universals *physical object* and *piece of wood*: since pieces of wood are also physical objects, every moment universal that characterizes physical objects also characterize pieces of wood (subsumption if reducible to the subset relation between characterizing sets). However, not all characterizing universal of pieces of wood are also characteze physical objects. For example, the universal *wood vein pattern* characterizes pieces of wood, but not physical objects, since not all physical objects have a wood vein pattern. Thus, when a piece of wood is seen through the filter of the physical object universal, the wood vein pattern property is abstracted away.

In this section, we define formally two concepts that capture this abstraction process: detailing moments of a substantial universal and the trimming of a particular structure by a substantial universal.

6.5.1 Detailing moments [3]

The detailing moments of a substantial universal *U* are those moments that represent details that are abstracted away by *U*. Any moment *m* that inheres in an instance *x* of *U* is either an instance of a characterizing universal of *U* or it is not. The detailing moments of *U* are comprised of precisely the later and of the moments that inhere on them. Formally, we have:

```
inductive_set detailingMoments ::
      ‹'u ⇒ 'p set›
      (‹Δ_› [999] 1000)
      for U
      where
    non_char_moments:
      ‹⟦ U ∈ 𝒰_𝒮 ; x ∈ Insts U ; y ◁ x ;
```

[3]The full Isabelle/HOL code for this section can be found in subsection E.4.1.

```
      ∀w u. y ::w u ⟶ ¬ char_by U u
    ⟧ ⟹ y ∈ ΔU⟩
  | sub_moments: ‹⟦ x ∈ ΔU ; y ◁◁ x ⟧ ⟹ y ∈ ΔU⟩
```

Given a particular structure with set of possible worlds $\mathcal{W}$, we call the set of possible worlds *trimmed by* `U`, written as `trim_worlds U`, the set composed by the each possible world minus the detailing moments of `U`:

definition ‹`trim_worlds U ≡ { w - ΔU | w . w ∈ 𝒲 }`›

Similarly, we call the inherence relation *trimmed by* `U`, written as `trim_inheres_in U`, the restriction of the inherence relation to those particulars that are not detailing moments of `U`, and, proceeding the same way, we can build restricted forms of all of the formal relations of the UFO theory of particulars:

definition ‹`trim_inheres_in U x y ≡ x ◁ y ∧ x ∉ ΔU ∧ y ∉ ΔU`›

definition ‹`trim_assoc_quale U x q ≡ x ⇝ q ∧ x ∉ ΔU`›

definition ‹`trim_towards U x y ≡`
`x ⟶ y ∧ x ∉ ΔU ∧ y ∉ ΔU`›

definition ‹`trim_quality_spaces U ≡`
`{ Q | Q x q . Q ∈ 𝒬𝒮 ∧ q ∈ Q`
`∧ trim_assoc_quale U x q }`›

Using the restricted versions of the formal elements of Γ, we can define the *trimming operation*: given a particular structure Γ and a universal `U`, we call Γ trimmed by `U`, written as `trim U`, the particular structure constructed by removing the detailing moments of `U` from Γ, leaving it otherwise the same:

definition ‹`trim U = (|`
`ps_quality_spaces = trim_quality_spaces U,`
`ps_worlds = trim_worlds U,`
`ps_inheres_in = trim_inheres_in U,`
`ps_assoc_quale = trim_assoc_quale U,`
`ps_towards = trim_towards U`
`|)`›

Given an universal `U`, we use the expression `trim_non_permutables U` to denote the particulars that are non-permutable in Γ trimmed by `U`:

abbreviation ‹`trim_non_permutables U ≡`

```
ufo_particular_theory_sig.non_permutables
  (trim_worlds U)
  (trim_inheres_in U)
  (trim_quality_spaces U)
  (trim_assoc_quale U)
  (trim_towards U)⟩
```

The set of detailing moments of an universal *U* is invariant under bijection morphisms (φ is a bijection morphism for Γ):

lemma *detailingMoments_invariant:*
⟨*inst2.detailingMoments U = φ ‘ detailingMoments U*⟩

The trimmed particular structure is also a particular structure:

lemma *trim_particular_struct[intro!,simp]:*
⟨*particular_struct (trim U)*⟩

There is at least one injective morphism from Γ trimmed by *U* (*trim U*) to Γ, i.e. the trimmed structure can be considered a sub-structure of Γ:

lemma *trim_injective[intro!,simp]:*
⟨*particular_struct_injection (trim U) Γ id*⟩

6.6 Formal Characterization of Sortality

Our approach to the definition of sortality changes the perspective from trying to define sortality thorugh the existence of a suitable identity condition to defining non-sortality through the abstraction of the properties that identity instances of a substantial universal. We arrive at a formal definition that follows this strategy by using the *trimming* operation.

First, let's note that the permutability of particulars of Γ is preserved under trimming by an universal, i.e. a permutable particular of Γ is also a permutable particular of *trim U*:

lemma *permutables_preserved_under_trimming:*
fixes *x* **and** *U :: ’u*

assumes *A:* ‹$x \in \mathcal{S}$› ‹$x \notin \mathcal{P}_{1!}$›
shows ‹$x \notin$ *trim_non_permutables U*›

Conversely, if a particular is non-permutable in the trimmed structure, it is also a non-permutable in Γ, i.e. it is identifiable in Γ:

lemma *non_permutables_under_trimming_are_the_identifiable:*
fixes *x* **and** *U*
assumes ‹$x \in \mathcal{S}$› ‹$x \in$ *trim_non_permutables U*›
shows ‹$x \in \mathcal{P}_{=}$›

We define the set of sortal universals as the set of substantial universals whose instances are non-permutable when trimmed by them. More specifically, a substantial universal *U* is a sortal just in case any instance of *U* is non-permutable in *trim U*. Formally, we have:

definition ‹*Sortals* $\equiv$ { *U* . *U* $\in \mathcal{U}_{\mathcal{S}} \wedge$ *Insts U* $\subseteq$ *trim_non_permutables U*}›

In other words, a sortal is a substantial universal that does not abstract enough details from its instances to the point that identification of the later becomes impossible.

Note that this definition does not require the existence of an identity condition, nor it implies that such an identity condition exists. Nevertheless, since permutability imply non-identifiability of particulars, the *non-sortality* implies the impossibility of such an identity condition to exist, since the impossibility of constructing an identifying predicate for an specific instance implies that the construction of a general identifying predicate for the set of instances is also impossible.

Thus, the definition of sortality provided here is, in one hand, weaker than a characterization of sortality through the existence of an identity criteria. However, it is completely determinable from the model of the UFO theory of particulars itself., i.e. it's grounded solely on the model structure. In contrast, the usual characterization of sortality is grounded in the existence of a suitable predicate, an entity that is not necessarily present in a particular structure, e.g. in those that languages are not represented.

As expected from the definition, all instances of a sortal are identifiable:

lemma *sortals_instances_are_identifiable:*
assumes ‹$U \in$ *Sortals*› ‹$x \in$ *Insts U*›
shows ‹$x \in \mathcal{P}_{=}$›

6.7 Final Considerations

In this chapter we presented a characterization of the concept of sortality in the context of the UFO ontology of substantial universals. The proposed characterization differs from the original one in that (1) it's completely formalized (in Isabelle/HOL); (2) it's definition relies only on the structural properties of the models of the UFO theory of endurant particulars, represented by the UFO particular structures and particular structure morphisms defined in chapter 4; and (3) since the definition relies on the structural characterization of identity presented in chapter 5, it avoids the assumption of the existence of a linguistic object.

The definition presented in this chapter was based on a perspective that considers the relationship between identity and universals to be a *negative* one, i.e. universals that abstract too many details from their instances may preclude the identification the same. This perspective constrasts with a *positive* view regarding the relationship between identity and universals, in which a sufficiently concrete universals "supplies" a principle of identity for its instances.

This shift in perspective is reflected in the pursuit of the formal conditions that characterize the impossibility of the identification of a substantial as an evidence of non-sortality, which can be framed using UFO particular structures without extraneous elements. This constrasts with the positive view, in which one has to provide the evidence of the existence of a predicate, definable in some language, that satisfy conditions that are hard to frame formally.

As a result, using the proposed definition, the evaluation of the sortality of a substantial universal becomes a completely objective and deterministic process that is a function of the properties of the substantials in the domain.

7 Applications

7.1 Introduction

This chapter describe some of the theoretical and practical applications of the theories and concepts presented in the previous chapters.

We start by discussing in section 7.2 the criterias for the falsifiability of the concepts of identifiability, individuality and sortality that can be derived from the definitions of the same concepts presented in chapter 5 and chapter 6.

In section 7.3, we discuss how to extend and use the UFO formalization in Isabelle/HOL in further theoretical research concerning UFO.

7.2 Falsifiability of Identifiability, Individuality and Sortality

During the process of conceptual modeling of a particular domain, it is usual for the conceptual modeler to collect examples of what the domain specialists or the system's stakeholders consider to be instances of the concepts that are being included in the model.

This collection serves the conceptual modeler in two ways: first, it provides a basis for the construction of inductive hypothesis regarding the invariants of the domain, and second, it provides counter-examples with which the conceptual modeler can falsify his or her hypothesis.

For example, during a interview, an specialists might inform the conceptual modeler that a certain concept C is characterized by properties P_1, P_2 and P_3.

The conceptual modeler then asks for examples and, after being given a few, notices that all examples seem to present also property P_4. From this, he or she can elaborate an hypothesis: are all Cs also characterized by P_4?

He then tries to falsify this hypothesis by asking the specialist whether he or she can think of an example of C that does not present property P_4. If a suitable example is given, then the hypothesis does not hold. If no examples can be given, the conceptual modeler would propably ask the specialist to reconsider the definition given for C.

One of the benefits os a rich foundational ontology, such as UFO, is that along with the concepts it presents, it also provides many potential questions that can be used to falsify hypothesis regarding these concepts and ther interelationships. For example:

- Is the concept C rigid?

  - Can you think of an example of C that could exist without being a C?

- Is the concept C a *Role*?
  - Can you think of an example of C that does have a relational property (does not bear a relational moment)?

- Is the concept C_1 a sub-concept of C_2?
  - Can you think of an example of C_1 that is not a C_2?

In summary, all formal properties presented in UFO should either be self-evident, in the sense that no doubt rest upon its application or not, or there should be some objective criteria for its validation.

The same can be said about UFO assumptions: if UFO assumes that such and such proposition is true, the evaluation of such proposition should either be self-evident or there should be an objective criteria that determines it.

Otherwise, there would be an element of subjectivity in the application of UFO's concepts that would jeopardize the construction of a shared agreement and may lead to a *false agreement*, since two conceptual modelers could arrive into two distinct models when presented with the same information of the domain.

The theories provided in chapter 5 and chapter 6 provide objective criteria for the evaluation of question regarding individuality, identity and sortality, leading to the following hypothesis/counter-example questions:

7.2.1 Individuality and Identity

Working with the assumption that all endurants in the domain possess identity (and individuality), are the samples gathered so far by the conceptual modeler consistent with this assumption?

How to answer:

1. construct the formal model (UFO particular strucure) of the samples (see section 4.2);

2. analyze the endomorphisms of the structure:

   - Is there an endomorphism that maps two or more particulars into a single one?

     yes Can the specialist provide more characteristics for the corresponding samples such that no such endomorphism can be found? Or, in other words, why does the specialist consider these to be distinct particulars?

     yes The particulars possess individuality.

no Either the assumption that all endurants possess individuality is false or the specialist's belief that these samples are distinct is false or the specialist lacks sufficient knowledge about the domain.

no So far, all samples present individuality.

- Is there an endomorphism that maps a particular into another (distinct) particular?

  no So far, all samples present identity.

  yes Can the specialist provide more characteristics for the corresponding samples such that no such endomorphism can be found? (How does the specialist pinpoints which are which among these particulars?)

  yes The particulars possess identity.

  no Either the assumption that all endurants possess identity is false or the specialist lacks sufficient knowledge about the domain.

By carrying these verifications continously with maybe some help of a CASE software tool that searches for theses morphisms, the conceptual modeler ensures that there will be no obvious obstacle to the investigation of the identity criteria for the particulars in the domain.

More specificaly, it provides an assurance that the basic assumptions hold regarding the identifiability of the particulars in the domain, allowing the evaluation of the sortality of substantial concepts to be done, as follows:

7.2.2 Sortality

One of the steps that are taken in the validation of UFO based ontologies, and, more specificaly, in the evaluation of the correctness of OntoUML conceptual models, is to classify endurant concepts according to their sortality. The reason is that there is a basic syntactical constraint, derived from the intuition about identity criteria and concept subsumption, that all concepts that subsume sortal concepts are necessarily sortals also, and, conversely, that all concepts subsumed by a non-sortal concept must also be, necessarily, non-sortals.

Thus, after evaluating the sortality of the concepts, if the conceptual modeler find a non-sortal concept in the model that subsumes a sortal concept, something must be wrong.

However, unless there is an objective method for evaluation sortality, the decision on whether or not a concept is sortal will have a degree of subjectivity that may introduce a hidden mistake in the model.

As we saw in chapter 6, if we use the existence or non-existence of an identity criteria for a concept's instances as the evidence for its sortality/non-sortality, there will be dificulties due to the introduction of either an arbitrary choice of language for the predicates or by the consideration of a the (open) scope of all formal languages. Besides these, there is also the issue regarding the characterization of what is to be considered a proper identity criteria.

However, we provided also in chapter 6 an alternate characterization of sortality that avoids these issues.

Here we consider, in practice, how one evaluates the sortality of a concept.

First, we build a particular structure based on the samples collected during the analysis of the domain. Then, we adjoin the set of universals and the instantiation relation between them and the particulars and possible worlds in the particular structure. And let U be the substantial universal whose sortality we wish to check.

First we compute the set of *detailing moments* of U, described in subsection 6.5.1,that are the moments of instances of U that are *abstracted away* by U. This set, written as Δ_U, is the smallest set satisfying the following:

1. if m is a moment that inheres in some instance x of U that does not instantiate some moment universal that characterizes U, then m is in Δ_U;

2. if m is a moment that inheres directly or indirectly in some moment in Δ_U, then m is also in Δ_U.

We then remove the particulars in Δ_U from all possible worlds in the particular structure. The resulting structure, is said to be the original structure *trimmed* by U. Afterwards, we try to find an endomorphism ϕ on the trimmed structure such that at least one instance of U is mapped by ϕ to some other particular. U is then considered a sortal just in case no such endomorphism can be found.

As an example, consider the configuration depicted in Figure 3. In this example we have a configuration with a single non-empty possible world, consisting in two spheres, S_A and S_B, with respective radius measuring 10m and 20m, separated by a distance of 200 meters, and a cube S_C, with an edge length of 5 meters, separated by 100 meters from both spheres. The moments representing the radius of the spheres are $M_{A,3}$, for S_A, and $M_{B,3}$, for S_B, while those that represent their distances from each other are $M_{A,1}$ (S_A being 200 meters from S_B) and $M_{B,1}$ (S_B being 200 meters from S_A). The moment that represents S_C's edge length is $M_{C,3}$ and those representing S_C's distance to S_A and S_B are, respecively, $M_{C,1}$ and $M_{C,2}$. The moments that represent S_A and S_B's distance to S_C are, respectively, $MA, 2$ and $MB, 2$.

Substantial	Inherent Moments	Instantited Universals
S_A	$m_{A,1}$, $m_{A,2}$, $m_{A,3}$	Physically-Separated Object, Round Separated Object
S_B	$m_{B,1}$, $m_{B,2}$, $m_{B,3}$	Physically-Separated Object, Round Separated Object
S_C	$m_{C,1}$, $m_{C,2}$, $m_{C,3}$	Physically-Separated Object, Cube-Shaped Separated Object

Moment	Bearer	Quale	Towardness	Instantited Universals
$m_{A,1}$	S_A	200m (distance)	S_B	Distance Moment
$m_{B,1}$	S_B	200m (distance)	S_A	Distance Moment
$m_{A,2}$	S_A	100m (distance)	S_C	Distance Moment
$m_{B,2}$	S_B	100m (distance)	S_C	Distance Moment
$m_{C,1}$	S_A	100m (distance)	S_A	Distance Moment
$m_{C,2}$	S_B	100m (distance)	S_B	Distance Moment
$m_{A,3}$	S_A	10m (radius)		Radius Moment
$m_{B,3}$	S_B	20m (radius)		Radius Moment
$m_{C,3}$	S_C	5m (edge length)		Edge-Length Moment

Figure 3 – Sortality example - Initial Structure

There are three substantial universals: *Physically-Separated Object*, characterized by the moment universal *Distance Moment*, *Round Separated Object*, characterized by *Distance Moment* and *Radius Moment*, and *Cube-Shaped Separated Object*, characterized by *Distance Moment* and *Edge-Length Moment*.

To identify which of the substantial universals are sortals, we have to trim the structure by each universal. For the two most concrete univerals, *Round Separated Object* and *Cube-Shaped Separated Object*, the trimmed structure is identical to the original, but for the universal Physically-Separated Object the resulting structure is depicted in Figure 4.

The only automorphism in the original structure is the identity morphism. However, in the structure depicted in Figure 4, there is one non-trivial permutation, ϕ, given by the mapping described in Figure 5. With this result, we can determine that *Round Separated Object* and *Cube-Shaped Separated Object* are sortals and that *Physically-Separated Object* is a non-sortal.

7.3 UFO in Isabelle/HOL as a Formal Framework

The main product produced during the research done in this thesis is the Isabelle/HOL theory that describes a fragment of the UFO foundational ontology. One of the reasons why this work was done only for a fragment of the UFO ontology is the large effort that is necessary to produce a complete theory of the UFO ontology.

However, the theory described in this work can serve as a basis not only for the

Substantial	Inherent Moments	Instantited Universals
S_A	$m_{A,1}$, $m_{A,2}$, $m_{A,3}$	Physically-Separated Object
S_B	$m_{B,1}$, $m_{B,2}$, $m_{B,3}$	Physically-Separated Object
S_C	$m_{C,1}$, $m_{C,2}$, $m_{C,3}$	Physically-Separated Object

Moment	Bearer	Quale	Towardness	Instantited Universals
$m_{A,1}$	S_A	200m (distance)	S_B	Distance Moment
$m_{B,1}$	S_B	200m (distance)	S_A	Distance Moment
$m_{A,2}$	S_A	100m (distance)	S_C	Distance Moment
$m_{B,2}$	S_B	100m (distance)	S_C	Distance Moment
$m_{C,1}$	S_A	100m (distance)	S_A	Distance Moment
$m_{C,2}$	S_B	100m (distance)	S_B	Distance Moment

Figure 4 – Sortality example - Structure after trimming by *Physically-Separated Object*

Source	Target		Source	Target
S_A	S_B		S_C	S_C
$m_{A,1}$	$m_{B,1}$		$m_{C,1}$	$m_{C,2}$
$m_{A,2}$	$m_{B,2}$		$m_{C,2}$	$m_{C,1}$
S_B	S_A			
$m_{B,1}$	$m_{A,1}$			
$m_{B,2}$	$m_{A,2}$			

Figure 5 – Non-Trivial Permutation for the structure in Figure 4.

formalization of the whole UFO ontology, but also as the basis for further research on UFO that requires a rigorous theoretical treatment. In this section we describe the process by which one extends the theory presented in this thesis.

The first step is to express the change in the UFO particular theory, described in chapter 3. That may require the addition of primitive symbols to the signature, of new definitions that use these symbols and the addition of the extension axioms. This process starts with the choice of an extension point, an Isabelle/HOL locale from which the extension shall be made.

The choice of which locale to extend depends on how much of the existing theory one intends to keep. For example, if one intends to keep all the elements of the theory defined in chapter 3, the extension should start at the `ufo_particular_theory_sig` and `ufo_particular_theory` locales, specified in section 3.7. On the other hand, if one wants to exclude or reframe the *towardnesss* and *quale association* relations, then the starting point should be the `inherence_sig` and `inherence` locales, specified in section 3.3.

Then, one extends the signature locale, whose name ends in "_sig", with the primitives that are to be included, if any, creating a new locale, also ended in "_sig",

without any axioms. Then, one extends this locale with all the desired definitions. Let's call this new signature locale `X_sig`.

Now, if there any axioms to be added, a new locale should be specified, let's call it `X`, extending both the new signature locale `X_sig` and the non-signature locale that serves as an extension point (e.g. `ufo_particular_theory`, `inherence`) and including all the desired axioms. At this point, any lemmas or theorems one wants to prove in the context of the intended theory can be done in the context of `X`. Finally, the new locales should be named `ufo_particular_theory_sig` and `ufo_particular_theory`, to use it in the theories that are defined in the subsequent chapters, if so desired.

If one intends to use the revised particular theory in the structural framework specified in chapter 4 or whishes to analyze the concepts of identity, individuality or sortality in the context of the new theory, it is necessary to extends also the theory of particular structures and particular structure morphisms. For that, it is necessary, first, to change the definition of the Isabelle/HOL record (section 4.2) that represents the particular structure, so that it reflects the set of primitive symbols of the revised theory. Then, if needed, abbreviations corresponding to these symbols should be added or removed from the context of the `particular_struct_sig` locale. Afterwards, any lemma specified in the `ParticularStructure.thy` Isabelle/HOL theory file whose proof does not validate anymore has to be revised and the proof remade, if necessary.

After getting the particular structure theory ready, it is necessary to change the particular structure morphism theory, specified in section 4.3, by removing or adding axioms that specify what does it mean for an instance of the new particular structure to be mapped into another while maintainig its invariants. This choice depends on the primitives of the theory and on the invariants that are to be preserved. Afterwards, the lemmas specified in the Isabelle/HOL theory files present in the `ParticularStructures` directory should have their proofs remade, if necessary.

From this point, one should revise, using the Isabelle software, all the theories that depend on those, correcting the proofs if necessary. When all the theories are proven valid again by the Isabelle proof checking, the theory can be considered updated.

7.4 Final Considerations

In this chapter, we presented some of the practical applications that arise from the results described in the previous chapters. More specifically, we explained how the formal definitions that were proposed for the concepts of identity, individuality and sortality can be translated into prcoedures for ascertaining, in a objective manner, the corresponding status of the particular examples that are collected during the process of conceptual modelling and the sortality status of the concepts that are being modeled.

We also described how the Isabelle/HOL theory that forms the core of this thesis can be extended to support investigations regarding the UFO theory that require a rigorous and verified treatment, by leveraging the support given by the Isabelle software environment.

8 Conclusion

This thesis presented the results of a research with the main goal of producing a formal specification, in Isabelle/HOL, of the concepts of identity, individuality and sortality in the context of the UFO ontology of endurants. In that pursuit we described a series of Isabelle/HOL theories, in chapters 3, 4, 5 and 6, that presented:

1. a fragment of the UFO-A ontology that includes a minimal conceptual framework for representing substantials, moments and their intrinsic and relational properties;

2. a theoretical framework within which the concepts of identity, individuality and sortality can be analyzed structuraly, by using a category of UFO particular structures, i.e. models of the UFO-A fragment, with suitables morphisms that preserve their invariants and by using several structural properties of UFO particulars that were defined through the framework of UFO particulars structure morphisms;

3. a formal definition for the concepts of individuality and identity that characterizes these notions in terms of the structural properties of UFO particulars, where the later's definition is presented in three forms, using a definite description predicate, using non-permutability and using identity anchors, were all approaches were proven logically equivalent but the later two avoid the assumption that some predicate or some formal language must exist;

4. a formal characterization of notion of sortality, based on the proposed definition of identity as non-permutability, that avoids the dependency on the existence of some criteria of identity, by considering non-sortals as concepts that are "too abstract", in the sense that if one "filters" the world through them, they lose the ability to uniquely identify their instances, as they become permutable.

The validity of the lemmas and theorems that were introduced in the analysis presented in this work and of the proof of the consistency of the proposed theories of UFO particulars and UFO particular structures was asserted through the automatic verification provided by the Isabelle proof-assistant software system.

Besides the theoretical results presented in Isabelle/HOL form, we also described, in chapter 7, some pragmatic applications of the proposed definitions of identity, individuality and sortality in the process of conceptual modeling, by presenting clear and objective procedures by which one can assert whether or not the particulars that are collected as samples during the conceptual modeling process possess individuality and identity, and whether or not a substantial universal is a sortal.

Furthermore, we also explained, in chapter 7, how the proposed theoretical framework written in Isabelle/HOL can serve as a basis for the theoretical analysis of UFO concepts and theories, confirming the usefulness of proof-assistant frameworks, such as Isabelle, in theoretical investigations in the area of Applied Ontology.

As the result of this research work, we were able to confirm the hypothesis stated in section 1.2:

1. it is possible to define the concepts of individuality and identity in the context of the UFO theory of particulars, as was demonstraded by presenting the definitions in Isabelle/HOL;

2. it is possible to define the concept of identity without requiring the existence of some predicate, and the implicit existence of a formal language, by framing the definition in terms of structural properties of UFO particulars;

3. the structural definitions of identity and individuality lead to a precise definition of sortality that can be completely determined from the structural properties of UFO particulars and its instantiation relation to UFO universals and from which an objective and complete method can be derived to ascertain the sortality of concepts in OntoUML;

4. the use of a higher order logic (Isabelle/HOL) instead of a first-order logic in the analysis presented in this thesis enabled the representation and quantification over UFO particular theory models, morphisms, and identifying predicates, all of are or contain second- or higher-order elements.

5. the usage of a proof-assistant framework that allows a continuous process of theory construction and automatic verification has shown to be useful in the identification of imprecisions and ambiguities in the original theory, as could be seen in the analysis of the *inherence* relation (see section 3.3).

8.1 Future Works

During the research process that produced this thesis, three potential directions in which this work could be improved were identified. The first one is to complete the formalization of the UFO theory of particulars by including the UFO concepts that were left out due to the focus of the research. This could be done in a step-wise manner, adding some concept, such as UFO part-whole relations, relators or complex quality spaces, along with a suitable revision of the theory of UFO particular structures and morphisms and with the reconstruction, of the proofs of the theories.

The second is to achieve a full abstraction of the definitions of identity, individuality and sortality, by framing them solely in terms of category-theoretic concepts, making them idependent of a particular structure choice and allowing them to be applied in other foundational ontologies.

The third direction is to relax the UFO assumptions that particulars possess individuality and identity, adapting UFO's theory of endurants to accomodate particulars without identity or individuality and exploring the consequences thereof.

Bibliography

ADAMEK, J.; HERRLICH, H.; STRECKER, G. *Abstract and Concrete Categories: The Joy of Cats*. [S.l.]: Dover Publications, 2009. (Dover books on mathematics). ISBN 9780486469348. Cited on page 30.

BERTOT, Y. et al. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. [S.l.]: Springer Berlin Heidelberg, 2004. (Texts in Theoretical Computer Science. An EATCS Series). ISBN 9783540208549. Cited on page 28.

BLACK, M. The identity of indiscernibles. *Mind*, [Oxford University Press, Mind Association], v. 61, n. 242, p. 153–164, 1952. ISSN 00264423, 14602113. Disponível em: <http://www.jstor.org/stable/2252291>. Cited 3 times on pages 76, 81, and 85.

BORST, W.; BORST, W. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. Tese (Doutorado) — UT, Netherlands, set. 1997. Cited on page 24.

BRUIJN, D. Checking mathematics with computer assistance. *Notices of the American Mathematical Society*, v. 38, p. 8–15, 1991. Cited on page 28.

BRUIJN, N. G. de. Automath, a language for mathematics. In: *Automation of Reasoning*. [S.l.]: Springer, 1983. p. 159–200. Cited on page 27.

CONSTABLE, R. L. et al. *Implementing Mathematics with the Nuprl Proof Development System*. USA: Prentice-Hall, Inc., 1986. ISBN 0134518322. Cited on page 28.

GORDON, M. J. C.; MELHAM, T. F. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. USA: Cambridge University Press, 1993. ISBN 0521441897. Cited on page 28.

GRUBER, T. R. Toward principles for the design of ontologies used for knowledge sharing. *International Journal Human-Computer Studies*, v. 43, p. 907–928, August 1993. Disponível em: <http://tomgruber.org/writing/onto-design.pdf>. Acesso em: 16 mar 2014. Cited on page 23.

GRUBER, T. R. A translation approach to portable ontology specifications. *Knowledge Acquisition*, v. 5, p. 199–220, 1993. Cited on page 15.

GUARINO, N. Formal ontology and information systems. In: . [S.l.]: IOS Press, 1998. p. 3–15. Cited on page 24.

GUERSON, J. et al. OntoUML Lightweight Editor: A model-based environment to build, evaluate and implement reference ontologies. In: HALLÉ, S.; MAYER, W. (Ed.). *19th IEEE International Enterprise Distributed Object Computing Workshops (EDOCW 2015)*. Adelaide, SA, Australia: IEEE Computer Society Conference Publishing Services (CPS), 2015. p. 144–147. Cited on page 19.

GUIZZARDI, G. *Ontological foundations for structural conceptual models*. [S.l.]: CTIT, Centre for Telematics and Information Technology, 2005. Cited 11 times on pages 16, 24, 35, 36, 39, 40, 48, 49, 51, 57, and 84.

HARRISON, J. *The HOL Light System Reference*. 2016. <https://www.cl.cam.ac.uk/~jrh13/hol-light/reference.pdf>. Cited on page 28.

MACLANE, S. *Categories for the Working Mathematician*. [S.l.]: Springer New York, 2013. (Graduate Texts in Mathematics). ISBN 9781461298397. Cited on page 30.

NICOLA, G. G. J. Individual determinacy and identity criteria in ontology-driven information systems. In: BORGO PASCAL HITZLER, O. K. S. (Ed.). *Formal Ontology in Information Systems*. [S.l.]: IOS Press, 2018. (Frontiers in Artificial Intelligence and Applications, v. 306), p. 83–95. Cited on page 20.

NIPKOW, T.; KLEIN, G. *Concrete Semantics: With Isabelle/HOL*. [S.l.]: Springer International Publishing, 2014. ISBN 9783319105420. Cited on page 28.

Appendix

APPENDIX A – Opening the Isabelle/HOL Sources

The Isabelle/HOL sources included in this document can be found in the `isabelle-sources.ta` file, embedded in the PDF file of the thesis. They can also be obtained by sending a request by email to the author.

To open the files in Isabelle, it is necessary to install the Isabelle environment first. To do so, download the Isabelle 2021 distribution for you operating system (Linux, macOS or Windows) at <https://isabelle.in.tum.de/>. After following the instructions provided in the web site and installing Isabelle, extract the embedded file with a PDF reader that supports this feature (Adobe Acrobat Reader, Firefox, etc.) and extract its contents, which consist in a folder called `Isabelle/` that contains the Isabelle/HOL source for the thesis.

To view, edit and verify the theories, open the Isabelle JEdit program (`Isabelle2021` in Isabelle's home folder) and open the theory files in the `extracted path/Isabelle/..` folders.

APPENDIX B – UFO Particulars and Universals

B.1 Particulars and Possible Worlds

```
theory PpossibleWorlds
  imports  Main "../Misc/Common"
begin type_synonym 'p world =  ‹'p set›

locale possible_worlds_sig =
  fixes 𝒲 :: ‹'p world set›
    and Typ_p :: ‹'p itself› — HIDE
begindefinition 𝒫 :: ‹'p set›
   where ‹𝒫 ≡ ⋃𝒲›definition
    ed :: ‹'p ⇒ 'p ⇒ bool›
  where
    ‹ed x y ≡ x ∈ 𝒫 ∧ y ∈ 𝒫 ∧
              (∀w ∈ 𝒲. x ∈ w ⟶ y ∈ w)›definition
    interdep :: ‹'p ⇒ 'p ⇒ bool›
  where
    ‹interdep x y ≡ ed x y ∧ ed y x›definition
    indep :: ‹'p ⇒ 'p ⇒ bool›
  where
    ‹indep x y ≡ x ∈ 𝒫 ∧ y ∈ 𝒫
               ∧ ¬ ed x y ∧ ¬ ed y x›

end

locale possible_worlds =
    possible_worlds_sig
      where Typ_p = ‹Typ_p›
  for
    Typ_p :: ‹'p itself› +
  assumes
    injection_to_ZF_exist:
      ‹∃(f :: 'p ⇒ ZF). inj f› and
    at_least_one_possible_world: ‹𝒲 ≠ ∅› and
    particulars_do_not_exist_in_some_world:
      ‹x ∈ 𝒫 ⟹ ∃w∈𝒲. x ∉ w›
```

```
beginend

context possible_worlds_sig
begin

lemma 𝒫_I[intro]:
  assumes ‹w ∈ 𝒲› ‹x ∈ w›
  shows ‹x ∈ 𝒫›
  using assms 𝒫_def by auto

lemma 𝒫_E[elim]:
  assumes ‹x ∈ 𝒫›
  obtains w where ‹w ∈ 𝒲› ‹x ∈ w›
  using assms 𝒫_def by auto

lemma edI[intro!]:
  assumes
      ‹x ∈ 𝒫› ‹y ∈ 𝒫›
      ‹⋀w. ⟦ w ∈ 𝒲 ; x ∈ w ⟧ ⟹ y ∈ w›
  shows ‹ed x y›
  using ed_def assms by auto

lemma edE[elim!]:
  assumes ‹ed x y›
  obtains ‹x ∈ 𝒫› ‹y ∈ 𝒫›
          ‹⋀w. ⟦ w ∈ 𝒲 ; x ∈ w ⟧ ⟹ y ∈ w›
  using ed_def assms by auto

lemma edD:
  assumes ‹ed x y› ‹w ∈ 𝒲› ‹x ∈ w›
  shows ‹y ∈ w›
  using ed_def assms by auto

lemma ed_scope:
  assumes ‹ed x y›
  shows ‹x ∈ 𝒫› ‹y ∈ 𝒫›
  using ed_def assms by auto

lemma indepI[intro!]:
  assumes ‹x ∈ 𝒫› ‹y ∈ 𝒫›
          ‹¬ ed x y›  ‹¬ ed y x›
  shows ‹indep x y›
```

```
  using assms by (auto simp: indep_def)

lemma indepI1:
  fixes x y w_x w_y
  assumes ‹w_x ∈ 𝒲› ‹x ∈ w_x› ‹y ∉ w_x›
          ‹w_y ∈ 𝒲› ‹y ∈ w_y› ‹x ∉ w_y›
  shows ‹indep x y›
  using assms by auto

lemma indepE[elim!]:
  assumes ‹indep x y›
  obtains ‹x ∈ 𝒫› ‹y ∈ 𝒫›
          ‹¬ ed x y›  ‹¬ ed y x›
  using assms
  by (auto simp: indep_def)

lemma ed_refl:
  ‹x ∈ 𝒫 ⟹ ed x x›
  by (auto simp: ed_def)

lemma ed_trans:
  ‹⟦ ed x y ; ed y z ⟧ ⟹ ed x z›
  by (auto simp: ed_def)

lemma interdep_refl:
  ‹x ∈ 𝒫 ⟹ interdep x x›
  by (auto simp: interdep_def ed_def)

lemma interdep_symm[sym]:
  ‹interdep x y ⟹ interdep y x›
  by (auto simp: interdep_def)

lemma interdep_trans:
  assumes ‹interdep x y› ‹interdep y z›
  shows ‹interdep x z›
  using assms by (simp add: interdep_def ; metis ed_trans)

lemma indep_sym[sym]:
  ‹indep x y ⟹ indep y x›
  by blast

end
```

```
context possible_worlds
begin

lemmas just_possible_worlds_axioms =
  at_least_one_possible_world
  particulars_do_not_exist_in_some_world

lemmas all_possible_worlds_axioms =
  just_possible_worlds_axioms

lemma worlds_are_made_of_particulars:
  ‹w ∈ 𝒲 ⟹ w ⊆ 𝒫›
  by (auto simp: 𝒫_def)

lemma particulars_exist_in_some_world:
  ‹x ∈ 𝒫 ⟹ ∃w∈𝒲. x ∈ w›
  by (auto simp: 𝒫_def)

lemma non_empty_particulars_at_least_two_worlds:
    ‹𝒫 ≠ ∅ ⟹ ∃w₁∈𝒲. ∃w₂∈𝒲. w₁ ≠ w₂›
proof -
  assume ‹𝒫 ≠ ∅›
  then obtain x where ‹x ∈ 𝒫›
    by blast
  then show ‹?thesis›
    using
      particulars_exist_in_some_world
      particulars_do_not_exist_in_some_world
    by metis
qed

lemma
  no_empty_worlds_imp_two_particulars:
  ‹∅ ∉ 𝒲 ⟹ ∃x₁∈𝒫. ∃x₂∈𝒫. x₁ ≠ x₂›
proof -
  assume ‹∅ ∉ 𝒲›
  then obtain w₁ where ‹w₁ ∈ 𝒲› ‹w₁ ≠ ∅›
    using at_least_one_possible_world by blast
  then obtain x₁ where ‹x₁ ∈ w₁› by blast
  then have ‹x₁ ∈ 𝒫›
    using ‹w₁ ∈ 𝒲› worlds_are_made_of_particulars
```

```
    by blast
  then obtain w2 where ‹w2 ∈ 𝒲› ‹w2 ≠ ∅› ‹x1 ∉ w2›
    using particulars_do_not_exist_in_some_world
          ‹∅ ∉ 𝒲›
    by blast
  then obtain x2 where ‹x2 ∈ w2› ‹x2 ≠ x1› by blast
  then have ‹x2 ∈ 𝒫›
    using worlds_are_made_of_particulars ‹w2 ∈ 𝒲›
    by blast
  then show ‹?thesis›
    using ‹x1 ∈ 𝒫› ‹x2 ≠ x1› by blast
qed

endcontext possible_worlds_sig
begin end

context possible_worlds
begin end
```

B.1.1 Modes of Existence

```
abbreviation (in possible_worlds_sig)
  endurants (‹ℰ›) where ‹ℰ ≡ 𝒫›


end
```

B.2 Substantials, Moments and Inherence

```
theory Inherence
  imports PossibleWorlds
begin

no_notation nth (infixl ‹!› 100)

locale inherence_sig =
    possible_worlds_sig where Typ_p = ‹Typ_p›
  for
     Typ_p :: ‹'p itself› +
  fixes
    inheresIn :: ‹'p ⇒ 'p ⇒ bool› (infix ‹◁› 75)
beginend
```

```
locale inherence_base =
    possible_worlds where Typ_p = ‹Typ_p›  +
    inherence_sig where Typ_p = ‹Typ_p›
  for
     Typ_p :: ‹'p itself› +
  assumes
    — Axiom (3) on page 213
    inherence_scope:
      ‹x ◁ y ⟹ x ∈ ℰ ∧ y ∈ ℰ› and
    — Axiom (4) on page 213
    inherence_imp_ed:
      ‹x ◁ y ⟹ ed x y› and
    — Axiom (9) on page 214
    moment_non_migration:
      ‹⟦ x ◁ y ; x ◁ z ⟧ ⟹ y = z›
begin

lemmas just_inherence_base_axioms =
  inherence_scope inherence_imp_ed
lemmas all_inherence_base_axioms =
  all_possible_worlds_axioms
  just_inherence_base_axiomsend

context inherence_sig
begin definition ℳ where ‹ℳ ≡ { x . ∃y. x ◁ y }›

lemma ℳ_I[intro]:
  ‹x ◁ y ⟹ x ∈ ℳ›
  using ℳ_def by auto

lemma ℳ_E[elim]:
  assumes ‹x ∈ ℳ›
  obtains y where ‹x ◁ y›
  using assms ℳ_def by auto

definition 𝒮 where ‹𝒮 ≡ ℰ - ℳ›

lemma 𝒮_I[intro!]:
  ‹⟦ x ∈ ℰ ; x ∉ ℳ ⟧ ⟹ x ∈ 𝒮›
  using 𝒮_def by auto
```

```
lemma 𝒮_E[elim!]:
  assumes ‹x ∈ 𝒮›
  obtains ‹x ∈ ℰ› ‹x ∉ ℳ›
  using assms 𝒮_def by autodefinition bearer :: ‹'p ⇒ 'p› (‹β›)
  where ‹β m ≡ THE x. m ◁ x›abbreviation
  inheres_in_trancl :: ‹'p ⇒ 'p ⇒ bool›
  (infix ‹◁◁›  75)
    where ‹x ◁◁ y ≡ (◁)^++ x y›abbreviation
  inheres_in_by :: ‹'p ⇒ nat ⇒ 'p ⇒ bool›
  (‹_ ◁◁- _› [74,1,74] 75)
    where ‹x ◁◁^n y ≡ ((◁)^^n) x y›definition
  nth_bearer :: ‹'p ⇒ nat ⇒ 'p› (‹#β›)
    where ‹#β m n ≡ THE x. m ◁◁^n x›

endcontext inherence_base
begin lemma bearer_eqI:
  assumes ‹x ◁ y›
  shows ‹β x = y›
proof
    (simp add: bearer_def ;
     intro the1_equality assms)
  show ‹∃!y. x ◁ y›
    apply (intro ex1I[of _ y] assms)
    using moment_non_migration assms
  by metis
qedlemma bearerI:
  assumes ‹x ∈ ℳ›
  shows ‹x ◁ β x›
proof
  (simp add: bearer_def ;
   rule the1I2 ;
   assumption?)
  have ‹∃y. x ◁ y› using assms ℳ_def by blast
  then show ‹∃!y. x ◁ y›
    apply (intro ex_ex1I, assumption)
    using moment_non_migration assms by metis
qedlemma nth_inherence_unique_cond:
  assumes ‹x ◁◁^n y› ‹x ◁◁^n z›
  shows ‹y = z›
proof (insert assms ; induct n arbitrary: y z rule: less_induct)
  fix i y z
  assume A: ‹⋀j y z. ⟦ j < i ; x ◁◁^j y ; x ◁◁^j z ⟧ ⟹ y = z›
```

```
        ‹x ◁◁^i y› ‹x ◁◁^i z›
  consider (zero) ‹i = 0› | (succ) j where ‹i = Suc j›
    using not0_implies_Suc by blast
  then show ‹y = z›
  proof(cases)
    case zero
    then show ?thesis using A(2,3) by auto
  next
    case succ
    then obtain y₁ where Y1: ‹x ◁◁^j y₁› ‹y₁ ◁ y› using A(2)
      by auto
    obtain z₁ where Z1: ‹x ◁◁^j z₁› ‹z₁ ◁ z› using A(3) succ
      by auto
    have ‹j < i› using succ by auto
    then have ‹y₁ = z₁›
      using A(1) Y1(1) Z1(1) by metis
    then have ‹y₁ ◁ z› using Z1(2) by simp
    then show ?thesis
      using moment_non_migration Y1(2) by metis
  qed
qedlemma nth_bearer_eq_nth_iteration_of_bearer:
  assumes
    nth_bearer_exists: ‹∃y. x ◁◁^n y›
  shows ‹#β x n = (β ^^ n) x›
proof (simp add: nth_bearer_def ; intro the1_equality)
  show ‹∃!y. x ◁◁^n y›
  proof (intro ex_ex1I nth_bearer_exists)
    fix y z
    assume ‹x ◁◁^n y› ‹x ◁◁^n z›
    then show ‹y = z›
      using nth_inherence_unique_cond
      by metis
  qed
next
  show ‹x ◁◁^n (β ^^ n) x›
    using nth_bearer_exists apply (induct n ; simp)
    subgoal for n
      apply (intro relcomppI[of _ _ ‹(β ^^ n) x›])
      subgoal G1 by blast
      apply (intro bearerI ; simp add: ℳ_def)
      using moment_non_migration nth_inherence_unique_cond by blast
    done
```

```
qedlemma inherence_mid_point[intro]:
  ‹y₁ ◁◁^(n-m) y₂›
  if assms: ‹x ◁◁^m y₁› ‹x ◁◁^n y₂› ‹m ≤ n›
proof -
  have ‹n = m + (n - m)› using ‹m ≤ n› by presburger
  then have ‹x ◁◁^(m + (n - m)) y₂› using assms(2) by simp
  then have X: ‹(((◁)^^m) OO ((◁)^^(n - m))) x y₂› using relpowp_add by metis
  obtain z where A: ‹x ◁◁^m z› ‹z ◁◁^(n - m) y₂›
    using X[THEN relcomppE]
    by metis
  then have ‹z = y₁› using nth_inherence_unique_cond assms(1) by metis
  then show ‹y₁ ◁◁^(n - m) y₂› using A(2) by simp
qed


endcontext inherence_sig
begin

definition
    is_an_ultimate_bearer_of
    :: ‹'p ⇒ 'p ⇒ bool›
  where
    ‹is_an_ultimate_bearer_of x y ⟷
            (◁)** y x ∧ x ∈ 𝒮›definition
  has_order
  :: ‹'p ⇒ nat ⇒ bool›
  where
    ‹has_order x n ⟷
      (∃y. x ◁◁^n y ∧
           is_an_ultimate_bearer_of y x)›

endcontext inherence_base
begin

lemma moments_are_endurants: ‹ℳ ⊆ ℰ›
  by (auto simp: ℳ_def inherence_scope)

lemma ultimate_bearer_unique:
  assumes
    ‹x ∈ ℰ›
    ‹is_an_ultimate_bearer_of y x›
    ‹is_an_ultimate_bearer_of z x›
```

```
  shows ‹y = z›
proof -
  obtain Y: ‹(◁)** x y› ‹⋀w. ¬ y ◁ w› ‹y ∈ ℰ›
    using assms(1,2) is_an_ultimate_bearer_of_def
          𝒮_def ℳ_def
    by auto
  obtain Z: ‹(◁)** x z› ‹⋀w. ¬ z ◁ w› ‹z ∈ ℰ›
    using assms(1,3) is_an_ultimate_bearer_of_def
          𝒮_def ℳ_def
    by auto
  have cases: ‹y = z ∨ y ◁◁ z ∨ z ◁◁ y›
    if A: ‹x ◁◁ y› ‹x ◁◁ z›
    for x y z
  proof (intro verit_and_pos(4) ; rule ccontr)
    assume B: ‹y ≠ z› ‹¬ y ◁◁ z› ‹¬ z ◁◁ y›
    show False
      using A(1,2) B
    proof (induct arbitrary: z rule: tranclp_induct)
      case (base y z)
      then show ?case
      proof (cases ‹x ◁ z›)
        assume ‹x ◁ z›
        then have ‹z = y›
          using ‹x ◁ y› moment_non_migration
          by auto
        then show False using ‹y ≠ z› by simp
      next
        assume ‹¬ x ◁ z›
        then obtain w where ‹x ◁ w› ‹w ◁◁ z›
          using ‹x ◁◁ z›
          by (metis rtranclpD tranclpD)
        then have ‹w = y›
          using ‹x ◁ y› moment_non_migration
          by auto
        then have ‹y ◁◁ z›
          using ‹w ◁◁ z› by simp
        then show False
          using ‹¬ y ◁◁ z› by simp
      qed
    next
      case (step y₁ z₁ z₂)
      then show False
```

```
        by (metis moment_non_migration rtranclpD
                  tranclp.trancl_into_trancl tranclpD)
    qed
  qed
  obtain ‹¬ y ◁◁ z› ‹¬ z ◁◁ y›
    using Y(2) Z(2)
    by (metis tranclpD)
  then show ‹y = z›
    using cases Y(1) Z(1)
    by (metis rtranclpD)
qedlemma ultimate_bearer_unique_order:
  assumes
    ‹x ∈ ℰ›
    ‹is_an_ultimate_bearer_of y x›
  shows ‹∃!n. has_order x n›
proof (cases ‹x ∈ ℳ›)
  assume ‹x ∉ ℳ›
  then have ‹x ∈ 𝒮›
     using ‹x ∈ ℰ› 𝒮_def by simp
  then have ‹has_order x 0›
    by (auto simp: has_order_def
            is_an_ultimate_bearer_of_def)
  have ‹n = 0› if ‹has_order x n› for n
    using that
    apply (auto simp: has_order_def
           is_an_ultimate_bearer_of_def
           𝒮_def ℳ_def)
    apply (rule ccontr ; simp)
    using ‹x ∉ ℳ› ℳ_def
    by (metis mem_Collect_eq
              tranclp_power tranclpD)
  then show ?thesis
    using ‹has_order x 0› by metis
next
  assume ‹x ∈ ℳ›
  show ?thesis
  proof -
    have ‹x ∈ ℰ›
      using assms(1) moments_are_endurants
      by auto
    have R1: ‹Suc i = j›
      if ‹i = j - 1› ‹0 < j›
```

```
    for i j
    using that
    by auto
  have R2: ‹δ = 0›
    if A: ‹x ◁◁^n y› ‹x ◁◁^(n+δ) y›
          ‹∀z. ¬ y ◁ z›
    for x y n δ
  proof -
    obtain z where ‹x ◁◁^n z› ‹z ◁◁^δ y›
      using A(2)
      by (metis relcomppE relpowp_add)
    then have ‹z = y›
      using A(1) nth_inherence_unique_cond
            moment_non_migration
      by metis
    then have ‹y ◁◁^δ y›
      using ‹z ◁◁^δ y› by simp
    have False if ‹0 < δ›
    proof -
      obtain γ where ‹δ = Suc γ›
        using ‹0 < δ› gr0_conv_Suc
        by blast
      then obtain q where ‹y ◁ q›
        using ‹y ◁◁^δ y›
        by (meson relpowp_Suc_D2)
      then show False using A(3) by simp
    qed
    then show ?thesis by auto
  qed
  have R3: ‹n1 = n2›
    if A: ‹x ◁◁^n1 y› ‹x ◁◁^n2 y›
          ‹∀z. ¬ y ◁ z› for x y n1 n2
  proof -
    consider
        (C1) ‹n1 = n2›
      | (C2) δ where ‹n2 = n1 + δ›
      | (C3) δ where ‹n1 = n2 + δ›
      by (meson le_Suc_ex linear)
    then show ?thesis
    proof (cases)
      case C1
      then show ?thesis by assumption
```

```
    next
      case C2
      show ?thesis
        using A apply (simp add: C2)
        using R2 by metis
    next
      case C3
      show ?thesis
        using A apply (simp add: C3)
        using R2 by metis
    qed
  qed
  obtain R4: ‹x ◁◁ y› ‹y ∈ 𝒮›
             ‹∀z. ¬ y ◁ z› ‹y ∈ ℰ›
    using assms(2)
    by (metis CollectI Diff_iff 𝒮_def
         ‹x ∈ ℳ› ℳ_def is_an_ultimate_bearer_of_def
         rtranclpD)
  have R5: ‹n₁ = n₂›
    if ‹x ◁◁^n₁ y› ‹x ◁◁^n₂ y›
    for n₁ n₂
    using that R3 R4(3)
    by metis
  obtain n where R6: ‹x ◁◁^n y›
    using R4(1)
    by (meson tranclp_power)
  have R7: ‹n₁ = n› if ‹x ◁◁^n₁ y› for n₁
    using R5 R6 that
    by metis
  have R8: ‹has_order x n›
    using R6 assms(2) has_order_def
    by (auto simp: has_order_def)
  obtain ‹x ∉ 𝒮›
    using 𝒮_def ‹x ∈ ℳ› by auto
  then have R9: ‹n₁ = n›
    if ‹has_order x n₁› for n₁
    using that
    apply (auto simp: has_order_def)
    subgoal by (simp add: assms(1))
    subgoal premises P for z
      using P R7 assms(1) assms(2)
            ultimate_bearer_unique
```

```
        by blast
      done
    show ?thesis
      using R8 R9 by metis
  qed
qedlemma inherence_scopeE[elim]:
  assumes ‹x ◁ y›
  obtains ‹x ∈ ℳ› ‹y ∈ ℰ›
  using assms inherence_scope by auto

lemma endurantI1[intro]:
  ‹x ∈ ℳ ⟹ x ∈ ℰ›
  using inherence_scope by blast

lemma endurantI2:
   ‹x ◁ y ⟹ y ∈ ℰ›
   by auto

lemma endurantI3[intro]:
  ‹x ∈ 𝒮 ⟹ x ∈ ℰ›
  by blast

lemma substantials_are_endurants:
  ‹𝒮 ⊆ ℰ›
  by auto

lemma subst_moments_are_disj:
  ‹𝒮 ∩ ℳ = ∅›
  by auto

lemma endurants_eq_un_moments_subst:
  ‹ℰ = 𝒮 ∪ ℳ›
  by auto

lemma endurant_cases[cases set]:
  assumes ‹x ∈ ℰ›
  obtains (substantial) ‹x ∈ 𝒮›
        | (moment) ‹x ∈ ℳ›
  using assms
  by auto

lemma bearer_ex1[intro]:
```

```
    ‹∃!y. x ◁ y› if ‹x ∈ 𝓜›
  using moment_non_migration
        ex_ex1I[of ‹λy. x ◁ y›]
        ‹x ∈ 𝓜›
  by blast

lemma inheres_in_bearerI[intro]:
  ‹x ◁ β x› if ‹x ∈ 𝓜›
  using ‹x ∈ 𝓜› bearer_eqI
      moment_non_migration
  by blast

lemma inheres_in_by_scope:
  assumes ‹0 < n› ‹x ◁◁^n y›
  obtains ‹x ∈ 𝓜› ‹y ∈ 𝓔›
proof
  obtain k1 where ‹x ◁ k1›
    using assms
    by (metis less_not_refl2 relpowp_E2)
  then show ‹x ∈ 𝓜›
    using that by blast
  obtain k2 where ‹k2 ◁ y›
    using assms
    by (metis less_not_refl2 relpowp_E)
  then show ‹y ∈ 𝓔›
    using that by blast
qed

lemma trans_inheres_in_scope:
  assumes ‹x ◁◁ y›
  obtains ‹x ∈ 𝓜› ‹y ∈ 𝓔›
proof -
  obtain n where ‹x ◁◁^n y› ‹0 < n›
    using assms
    by (meson tranclp_power)
  then show ‹?thesis›
    using that inheres_in_by_scope
    by metis
qed

lemma nth_bearer_eqI[intro!]:
    ‹#β x n = y›
```

```
    if assms: ‹x ◁◁^n y›
    for x y n
  apply (simp only: nth_bearer_def)
  using the1_equality
        nth_inherence_unique_cond assms
  by metis

lemma nth_bearerI[intro!]:
    ‹x ◁◁^n (#β x n)›
    if assms: ‹∃y. x ◁◁^n y›
    for x n
  apply (simp only: nth_bearer_def)
  using the1I2 assms nth_inherence_unique_cond
  by (metis (no_types, hide_lams))

lemma zeroth_bearer[simp]: ‹#β x 0 = x›
  using nth_bearerI
  by auto

lemma nth_bearer_range:
  ‹#β x n ∈ ℰ›
  if assms: ‹∃y. x ◁◁^n y› ‹0 < n›
  for x n
proof -
  have ‹x ◁◁^n (#β x n)› using assms by blast
  then show ‹?thesis›
    using ‹0 < n› inheres_in_by_scope assms
    by blast
qed

lemma nth_bearer_range':
    ‹#β x n ∈ ℰ›
    if assms: ‹∃y. x ◁◁^n y› ‹x ∈ ℰ›
    for x n
  using assms
  by (cases ‹n = 0› ; auto intro!: nth_bearer_range)

lemma particular_cases_1[cases set]:
  assumes ‹x ∈ 𝒫›
  obtains (substantial) ‹x ∈ 𝒮›
        | (moment) ‹x ∈ ℳ›
  using assms by auto
```

```
end

locale inherence_original =
  inherence_base +
  assumes

  inherence_irrefl: ‹¬ x ◁ x› and

  inherence_asymm: ‹x ◁ y ⟹ ¬ y ◁ x› and

  inherence_intransitive: "⟦ x ◁ y ; y ◁ z ⟧ ⟹ ¬ x ◁ z"
begin endcontext inherence_sig
begin

definition order :: ‹'p ⇒ nat›
  where ‹order m ≡ THE n. has_order m n›

definition
    ultimateBearer
    :: ‹'p ⇒ 'p› (‹!β›)
  where
    ‹!β m ≡ THE x. is_an_ultimate_bearer_of x m›

endcontext inherence_original
begin

lemmas all_inherence_original_axioms =
  all_inherence_base_axioms
  inherence_irrefl
  inherence_asymm

end
```

B.2.1 No Ultimate Bearer Anomaly

```
context inherence_sig
begin

definition
  ‹has_moment_without_ultimate_bearer ⟷
    (∃m ∈ ℳ. ∀y. ¬ is_an_ultimate_bearer_of y m)›
```

```
endlemma (in inherence_base)
    has_moment_without_ultimate_bearerI[intro]:
  assumes ‹m ∈ 𝓜› ‹⋀n. ∃x. m ◁◁^n x›
  shows ‹has_moment_without_ultimate_bearer›
proof -
  have A: ‹x ∉ 𝓢› if ‹m ◁◁ x› for x
  proof -
    obtain i where ‹m ◁◁^i x›
      using ‹m ◁◁ x›
      by (meson tranclp_power)
    then obtain z where ‹m ◁◁^Suc i z›
      using assms(2)
      by blast
    then have ‹x ◁◁^(Suc i) - i z›
      using inherence_mid_point ‹m ◁◁^i x›
            Suc_leD
      by blast
    then have ‹x ◁ z› by auto
    then show ‹x ∉ 𝓢› using 𝓢_def 𝓜_def by auto
  qed
  have ‹∀y. ¬ is_an_ultimate_bearer_of y m›
  proof (intro allI notI)
    fix y
    assume ‹is_an_ultimate_bearer_of y m›
    then obtain ‹y ∈ 𝓢› ‹m ◁◁ y›
      using is_an_ultimate_bearer_of_def
      by (metis 𝓢_E assms(1) rtranclpD)
    then show False using A by auto
  qed
  then show ?thesis
    using ‹m ∈ 𝓜›
    by (auto simp:
        has_moment_without_ultimate_bearer_def)
qed

lemma (in inherence_base)
  has_moment_without_ultimate_bearerE[elim]:
  assumes ‹has_moment_without_ultimate_bearer›
  obtains m where ‹m ∈ 𝓜› ‹⋀n. ∃x. m ◁◁^n x›
proof -
  obtain m where ‹m ∈ 𝓜›
```

```
    and m: ‹⋀y. is_an_ultimate_bearer_of y m⟹ False›
    using assms
    by (auto simp: has_moment_without_ultimate_bearer_def)
  have A: ‹x ∈ ℳ›
    if ‹(◁)** m x› for x
    using that is_an_ultimate_bearer_of_def m
    by (metis ‹m ∈ ℳ› 𝒮_I rtranclpD
        trans_inheres_in_scope)
  have ‹∃x. m ◁◁^n x› for n
  proof (induct ‹n›)
    case 0
    then show ‹?case› by auto
  next
    case (Suc n)
    then obtain x where ‹m ◁◁^n x›
      by blast
    have ‹x ∈ ℳ›
    proof (cases ‹n = 0›)
      assume ‹n = 0›
      then show ‹x ∈ ℳ›
        using ‹m ∈ ℳ› ‹m ◁◁^n x›
        by auto
    next
      assume ‹n ≠ 0›
      then have ‹0 < n› by auto
      then have ‹m ◁◁ x› using ‹m ◁◁^n x›
        by (meson tranclp_power)
      then show ‹x ∈ ℳ› using A
        by (simp add: tranclp_into_rtranclp)
    qed
    then obtain y where ‹x ◁ y› by blast
    then have ‹m ◁◁^Suc n y›
      using ‹m ◁◁^n x›
      by auto
    then show ‹?case› by blast
  qed
  then show ‹?thesis›
    using that ‹m ∈ ℳ›
    by metis
qed
```

B.2.2 Cyclic Inherence Anomaly

```
definition ‹cyclic R ≡ ∃x. R^++ x x›lemma (in inherence_base)
  cyclic_inherence_has_moment_witout_ultimate_bearer[intro!,simp]:
  ‹cyclic (◁) ⟹ has_moment_without_ultimate_bearer›
proof -
  assume ‹cyclic (◁)›
  then obtain x where ‹x ◁◁ x›
    using cyclic_def  by metis
  then obtain n_x where ‹x ◁◁^n_x x› ‹0 < n_x›
    using tranclp_power by force
  have ‹x ∈ ℳ›
  proof -
    obtain y where ‹x ◁ y› ‹y ◁◁ x›
      by (metis ‹x ◁◁ x› converse_tranclpE)
    then show ‹x ∈ ℳ› using ℳ_def by blast
  qed
  show ‹?thesis›
  proof
    (intro has_moment_without_ultimate_bearerI[of ‹x›]
     ‹x ∈ ℳ›)
    fix n
    obtain a b
      where n: ‹n = a*n_x + b› ‹b < n_x›
      using ‹0 < n_x›
      by (metis le_less le_neq_implies_less
                less_irrefl_nat mod_less_divisor
                mod_mult2_eq mult.commute mult_0
                split_mod)
    have ‹x ◁◁^a*n_x x›
    proof (induct ‹a›)
      case 0
      then show ‹?case›
        by auto
    next
      case (Suc a)
      then have A: ‹x ◁◁^a * n_x + n_x x›
        using ‹x ◁◁^n_x x›
        by (metis relcomppI relpowp_add)
      have B: ‹a * n_x + n_x = Suc a * n_x›
        by simp
      show ‹?case› using A by (simp add: B)
```

```
    qed
    obtain z where ‹x ◁◁^b z›
      using ‹b < n_x› ‹x ◁◁^n_x x›
      by (metis ‹0 < n_x› less_imp_Suc_add
          nat.inject nat_neq_iff
          relcomppE relpowp_E relpowp_add)
    then have ‹x ◁◁^(a*n_x + b) z›
      using ‹x ◁◁^(a*n_x) x›
      by (metis relcomppI relpowp_add)
    then have ‹x ◁◁^n z› using n by simp
    then show ‹∃z. x ◁◁^n z› by blast
  qed
qedlemma inherence_original_allows_cycles:
  ‹∃(𝒲 :: nat set set) inheresIn.
     inherence_original 𝒲 inheresIn ∧
     cyclic inheresIn›
proof -
  let ‹?𝒲› = ‹{∅,{1,2,3 :: nat}}›
  define inheresIn where inheresIn[simp]:
    ‹inheresIn x y ≡
          (x = 1 ∧ y = 2)
        ∨ (x = 2 ∧ y = 3)
        ∨ (x = 3 ∧ y = 1)›
    for x y :: ‹nat›
  interpret possible_worlds ‹?𝒲›
    apply (unfold_locales ; simp?)
    using inj_nat2Nat by auto
  have I_eq[simp]: ‹𝒫 = {1,2,3}›
    by (simp only: 𝒫_def ; blast)
  have ed_iff[simp]:
    ‹ed x y ⟷ x ∈ {1,2,3} ∧
                y ∈ {1,2,3}›
    for x y
    by (simp only: ed_def ;
        simp only: I_eq ; blast)

  interpret inherence_base ‹?𝒲› ‹inheresIn›
    apply (unfold_locales)
    subgoal G1 using I_eq by auto
    subgoal G2 by (simp only: ed_iff ;
                   simp ; blast)
    by (simp ; presburger )
```

```
  interpret inherence_original ‹?𝒲› ‹inheresIn›
    apply (unfold_locales ; simp)
    by linarith+

  have ‹inheresIn^++ 1 1›
  proof -
    obtain ‹inheresIn 1 2›
           ‹inheresIn 2 3›
           ‹inheresIn 3 1› by simp
    then show ‹inheresIn^++ 1 1›
      by (metis tranclp.simps)
  qed
  then have ‹cyclic inheresIn›
    by (auto simp: cyclic_def)
  then show ‹?thesis›
    using inherence_original_axioms
    by blast
qedlemma inherence_original_has_ultimate_bearer_problem:
  ‹∃(𝒲 :: nat set set) inheresIn.
     inherence_original 𝒲 inheresIn ∧
     inherence_sig.has_moment_without_ultimate_bearer
          𝒲 inheresIn›
  using inherence_original_allows_cycles
    inherence_base.cyclic_inherence_has_moment_witout_ultimate_bearer
  using inherence_original_def
  by blast

locale inherence_V2 =
  inherence_original +
  assumes
    no_inherence_cycles: ‹¬ (◁)^++ x x›

begin lemmas all_inherence_V2_axioms =
    all_inherence_original_axioms
    no_inherence_cycles

lemma no_inherence_cycles_2[simp,dest!]:
  ‹(◁)^++ x x ⟹ False›
  using no_inherence_cycles by simp

end
```

B.2.3 Infinite Inherence Chain Anomaly

```
definition (in inherence_sig)
    inf_inh_asc_chain :: ‹'p set ⇒ 'p ⇒ bool›
  where
    ‹inf_inh_asc_chain X x ≡
      infinite X ∧ (∀y. y ∈ X ⟷ (◁)** x y)›definition (in inherence_sig)
  ‹has_inf_inh_asc_chain ≡
    ∃X x. inf_inh_asc_chain X x›definition (in inherence_sig)
  bearer_order :: ‹'p ⇒ 'p ⇒ nat› (‹Δβ›)
  where ‹Δβ m s ≡ THE n. m ◁◁n s›definition (in inherence_sig)
  bearers :: ‹'p ⇒ 'p set› (‹β*›)
  where ‹β* m ≡ { s . m ◁◁ s }›lemma (in inherence_V2) no_cycles:
  assumes ‹x ◁◁n y› ‹x ◁◁n' y›
  shows ‹n = n'›
proof (rule ccontr)
  assume ‹n ≠ n'›
  then consider
      (lt) ‹n' < n›
    | (gt) ‹n < n'›
    using nat_neq_iff by blast
  then obtain a b where
    AB: ‹a < b› ‹x ◁◁a y› ‹x ◁◁b y›
    by (meson assms)
  then have ‹y ◁◁b - a x›
    using inherence_mid_point
    by (meson no_inherence_cycles_2
              order.strict_implies_order
              tranclp_power zero_less_diff)
  then have ‹x ◁◁ x›
    by (meson AB ‹a < b› relpowp_imp_rtranclp
          rtranclp_tranclp_tranclp
          tranclp_power zero_less_diff)
  then show ‹False›
    using no_inherence_cycles by simp
qedlemma (in inherence_V2) bearer_order_ex1:
  assumes ‹s ∈ β* m›
  shows ‹∃!n. m ◁◁n s›
proof -
  have ‹m ◁◁ s› using assms bearers_def by auto
  then obtain n where ‹m ◁◁n s›
    using tranclp_power by metis
```

```
  then have A: ‹n' = n› if ‹m ◁◁^n' s› for n'
    using no_cycles that by blast
  then show ‹?thesis›
    by (intro ex1I[of _ ‹n›] ‹m ◁◁^n s› ; simp)
qedcontext inherence_V2
begin

lemma bearer_order_prop:
  assumes ‹s ∈ β* m›
  shows ‹m ◁◁^Δβ m s s›
  using theI' bearer_order_ex1 assms
        bearer_order_def
  by metis

lemma bearer_order_eq:
  assumes ‹m ◁◁^n s›
  shows ‹Δβ m s = n›
proof (cases ‹n = 0›)
  assume ‹n = 0›
  then have ‹s = m›
    using assms by auto
  have D0[simp]: ‹Δβ m m = 0›
  proof (simp only: bearer_order_def
         ; intro the_equality ; simp?)
    fix n'
    assume as: ‹m ◁◁^n' m›
    show ‹n' = 0›
    proof (rule ccontr)
      assume ‹n' ≠ 0›
      then have ‹0 < n'› by blast
      then have ‹m ◁◁ m›
        using as by (meson tranclp_power)
      then show ‹False›
        using no_inherence_cycles by simp
    qed
  qed
  show ‹?thesis›
    using ‹n = 0› D0 ‹s = m› by simp
next
  assume ‹n ≠ 0›
  then have ‹0 < n› by blast
  then have ‹m ◁◁ s›
```

```
    using assms by (meson tranclp_power)
  then have ‹s ∈ β* m›
    by (simp add: bearers_def)
  then have ‹m ◁◁^Δβ m s s›
    using bearer_order_prop by simp
  then show ‹?thesis›
    using no_cycles assms by simp
qedlemma bearer_order_inj: ‹inj_on (Δβ x) (β* x)›
proof
  fix y z
  assume ‹y ∈ β* x› ‹z ∈ β* x› and
   Δeq: ‹Δβ x y = Δβ x z›
  have A: ‹x ◁◁^Δβ x y y›
    using ‹y ∈ β* x› bearer_order_prop
    by metis
  have ‹x ◁◁^Δβ x z z›
    using ‹z ∈ β* x› bearer_order_prop
    by metis
  then have B: ‹x ◁◁^Δβ x y z›
    using Δeq by simp
  then show ‹y = z›
    using nth_inherence_unique_cond A
    by simp
qed

endlemma (in inherence_base)
  infinite_inherence_chain_imp_has_moment_without_ultimate_bearer:
  assumes ‹has_inf_inh_asc_chain›
  shows ‹has_moment_without_ultimate_bearer›
proof - proof (auto)
    show ‹y = x›
      if as: ‹y ∈ X› ‹y ∉ β* x› for y
    proof -
      have ‹(◁)** x y›
        using as(1) B by simp
      then have ‹x = y ∨ x ◁◁ y›
        by (meson rtranclpD)
      then show ‹y = x›
        using as(2) bearers_def
        by auto
    qed
    show ‹y ∈ X›
```

```
      if as: ‹y ∈ β* x› for y
      using B as
      by (auto simp: bearers_def)
  qed  proof (intro set_eqI ;
         simp add: bearers_def ;
         intro iffI)
    show ‹∃i. q = #β x i ∧ 0 < i ∧ i ≤ n›
       if ‹x ◁◁ q› for q
    proof -
      obtain i where ‹x ◁◁^i q› ‹0 < i›
        using ‹x ◁◁ q›
        by (meson tranclp_power)
      then have  ‹#β x i = q›
        using nth_bearer_eqI by simp
      have ‹i ≤ n›
      proof (rule ccontr)
        assume ‹¬ i ≤ n›
        then have ‹n < i› by auto
        then have AA: ‹y ◁◁^(i - n) q›
          using ‹x ◁◁^n y› ‹x ◁◁^i q›
                inherence_mid_point
          by simp
        have ‹0 < i - n›
          using ‹n < i› by auto
        then have ‹y ∈ 𝒩›
          using inheres_in_by_scope AA
          by metis
        then show ‹False›
          using ‹y ∈ 𝒮› by blast
      qed
      then show ‹?thesis›
        using ‹#β x i = q› that ‹0 < i›
        by metis
    qed
    show ‹x ◁◁ q›
      if as: ‹∃i. q = #β x i ∧ 0 < i ∧ i ≤ n›
      for q
    proof -
      obtain i where
        ‹q = #β x i› ‹0 < i› ‹i ≤ n›
        using as by metis
      obtain y' where ‹x ◁◁^i y'›
```

```
        using ‹x ◁◁^n y› ‹i ≤ n›
        by (metis antisym_conv2 less_imp_Suc_add
                  relcomppE
                  relpowp_Suc_E relpowp_add)
      then have AA: ‹x ◁◁^i #β x i›
        using nth_bearerI by metis
      then have BB: ‹x ◁◁ #β x i›
        by (meson ‹0 < i› tranclp_power)
      have ‹#β x i = y'›
        using ‹x ◁◁^i y'› nth_bearer_eqI
        by metis
      then show ‹x ◁◁ q›
        using ‹q = #β x i› BB by simp
    qed
  qedlemma inherence_V2_allows_infinite_inherence_chain:
  ‹∃ (𝒲 :: nat set set) inheresIn.
        inherence_V2 𝒲 inheresIn
      ∧ inherence_sig.has_inf_inh_asc_chain inheresIn›
proof -
  let ‹?𝒲› = ‹{∅,UNIV :: nat set}›
  let ‹?inheresIn› = ‹λx y. y = Suc x›
  interpret possible_worlds ‹?𝒲›
    apply (unfold_locales ; simp? )
    using inj_nat2Nat by blast
  interpret inherence_original ‹?𝒲› ‹?inheresIn›
    by (unfold_locales ; simp add: ed_def 𝒫_def)
  interpret inherence_V2 ‹?𝒲› ‹?inheresIn›
    apply (unfold_locales ; intro notI)
    using less_nat_rel by auto
  have A: ‹?inheresIn** = (≤)›
  proof (intro ext)
    fix x y
    show ‹?inheresIn** x y ⟷ x ≤ y›
      by (metis Nitpick.rtranclp_unfold
            le_eq_less_or_eq less_nat_rel)
  qed
  have ‹has_inf_inh_asc_chain›
    by (simp add: has_inf_inh_asc_chain_def
          ; intro exI[of _ ‹UNIV›]
          ; intro exI[of _ ‹0›]
          ; simp add: inf_inh_asc_chain_def A)
  then show ‹?thesis›
```

```
    using inherence_V2_axioms by auto
qedlemma inherence_V2_has_ultimate_bearer_problem:
  ‹∃ (𝒲 :: nat set set) inheresIn.
        inherence_V2 𝒲 inheresIn
      ∧ inherence_sig.has_moment_without_ultimate_bearer
        𝒲 inheresIn›
  using
  inherence_V2_allows_infinite_inherence_chain
  inherence_base.infinite_inherence_chain_imp_has_moment_without_ultimate_bearer

  inherence_original.axioms(1)
  inherence_V2.axioms(1)
  by blast

locale inherence_V3 =
  inherence_V2  +
  assumes
    no_infinite_inherence_chains:
      ‹¬ inf_inh_asc_chain X x›
begin lemmas all_inherence_V3_axioms =
  all_inherence_V2_axioms
  no_infinite_inherence_chainslemma no_ultimate_bearer_problem:
  ‹¬ has_moment_without_ultimate_bearer›
proof
  show ‹False›
    when ‹∃m. inf_inh_asc_chain ({m} ∪ β* m) m›
    using no_infinite_inherence_chains that
    by metis

  assume ‹has_moment_without_ultimate_bearer›
  then obtain m
    where ‹m ∈ ℳ›
      and A: ‹⋀n. ∃x. m ◁◁^n x›
    using has_moment_without_ultimate_bearerE
    by metis
  show ‹∃m. inf_inh_asc_chain ({m} ∪ β* m) m›
    apply (intro exI[of _ ‹m›]
          ; simp add: inf_inh_asc_chain_def)
    apply (intro conjI allI iffI disjCI
           ; (elim disjE)? ; simp?)
  proof  -
    have ‹Δ^β m ‘ β* m =
```

```
      {n | n. 0 < n ∧ m ◁◁^n #β m n}›
proof (auto simp: image_iff)
  show ‹0 < Δ^β m y› if ‹y ∈ β* m› for y
    using that bearer_order_prop
          bearers_def
    by fastforce
  show ‹∃z. m ◁◁^(Δ^β m y) z›
    if as1: ‹y ∈ β* m› for y
    using A by simp
  show ‹∃x∈β* m. n = Δ^β m x›
    if as2: ‹0 < n› ‹m ◁◁^n #β m n›
    for n
  proof (intro bexI[of _ ‹#β m n›])
    show ‹n = Δ^β m (#β m n)›
      using as2 bearer_order_eq that
      by auto
    show ‹#β m n ∈ β* m›
      using as2 bearers_def
      by (metis mem_Collect_eq tranclp_power)
  qed
qed
have ‹Δ^β m ` β* m = UNIV - {0}›
  apply (auto simp: image_iff set_eq_iff)
  subgoal
    using bearer_order_prop
          bearer_order_eq bearers_def
    by fastforce
  subgoal for n
    apply (intro bexI[of _ ‹#β m n›])
    subgoal using A
      by (metis bearer_order_eq
                nth_bearer_eqI)
    subgoal using A
      by (metis inherence_sig.bearers_def
            mem_Collect_eq
            nth_bearer_eqI tranclp_power)
    done
  done
then have ‹infinite (Δ^β m ` β* m)›
  by simp
then show ‹infinite (β* m)›
  by blast
```

```
    show ‹(◁)** m y› if ‹y ∈ β* m› for y
      using bearers_def that by auto
    show ‹y = m›
      if ‹y ∉ β* m› ‹(◁)** m y› for y
      by (metis bearers_def mem_Collect_eq
                rtranclpD that(1) that(2))
  qed
qed

endlemma (in inherence_base) no_ultimate_bearer_conditions:
  ‹¬ has_moment_without_ultimate_bearer ⟷
     ¬ cyclic (◁) ∧ ¬ has_inf_inh_asc_chain›
  (is ‹¬ ?A ⟷ ¬ ?B ∧ ¬ ?C›)
proof -
  have R1: ‹¬ A ⟷ ¬ B ∧ ¬ C›
     if ‹B ⟹ A›
       ‹C ⟹ A›
       ‹⟦ A ; ¬ B ⟧ ⟹ C›
     for A B C
    using that by blast
  note R2 = R1[
       where ?A = ‹?A› and ?B = ‹?B› and ?C = ‹?C›,
       OF cyclic_inherence_has_moment_witout_ultimate_bearer
       infinite_inherence_chain_imp_has_moment_without_ultimate_bearer,
       simplified]
  show ‹?thesis›
  proof (intro R2)
    assume as:
      ‹has_moment_without_ultimate_bearer›
      ‹¬ cyclic (◁)›
    interpret inherence_original ‹𝒲› ‹inheresIn›
      apply(unfold_locales)
      subgoal for x
        using ‹¬ cyclic (◁)› cyclic_def
        by blast
      subgoal for x
        using ‹¬ cyclic (◁)›
        by (meson cyclic_def tranclp.simps
                  tranclp_into_tranclp2)
      subgoal for x y z
        using ‹⋀x. ¬ x ◁ x› moment_non_migration
        by auto
```

```
      done
    interpret inherence_V2 ‹𝒲› ‹inheresIn›
      apply(unfold_locales)
      using ‹¬ cyclic (◁)› cyclic_def by blast
    show ‹has_inf_inh_asc_chain› using as(1)
      using has_inf_inh_asc_chain_def
            inherence_V2_axioms
            inherence_V3.no_ultimate_bearer_problem
            inherence_V3_axioms_def
            inherence_V3_def
      by blast
  qed
qedcontext inherence_base
begin

definition suborders :: ‹'p ⇒ nat set›
  where
  ‹suborders x ≡ { n | n y . x ◁◁^n y }›

lemma subordersI[intro!]:
  ‹x ◁◁^n y ⟹ n ∈ suborders x›
  using suborders_def by auto

lemma subordersE[elim]:
  assumes ‹n ∈ suborders x›
  obtains y where ‹x ◁◁^n y›
  using assms suborders_def
  by autolemma has_ultimate_bearer_iff_suborders_finite:
  assumes ‹x ∈ ℳ›
  shows ‹(∃y. is_an_ultimate_bearer_of y x) ⟷
         finite (suborders x)›
proof
  assume ‹∃y. is_an_ultimate_bearer_of y x›
  then obtain y
    where ‹is_an_ultimate_bearer_of y x›
    by blast
  then obtain ‹x ◁◁ y› ‹y ∈ 𝒮›
    using is_an_ultimate_bearer_of_def assms
    by (metis 𝒮_E rtranclpD)
  then obtain n_y where ‹x ◁◁^n_y y›
    by (meson tranclp_power)
  have ‹¬ y ◁ z› for z
```

```
    using ‹y ∈ 𝒮› by blast
  then have ‹¬ y ◁◁ z› for z
    by (meson tranclpD)
  then have A: ‹n = 0› if ‹y ◁◁^n z›
    for n z
    by (meson neq0_conv that tranclp_power)
  have ‹suborders x = { i . i ≤ n_y }›
  proof (rule ccontr)
    assume ‹suborders x ≠ {i. i ≤ n_y}›
    then consider
        (lt) n where ‹n ∉ suborders x› ‹n ≤ n_y›
      | (gt) n where ‹n ∈ suborders x› ‹n_y < n›
      using suborders_def by fastforce
    then show ‹False›
    proof (cases)
      case (lt n)
      obtain k where ‹x ◁◁^n k›
        using ‹x ◁◁^n_y y› ‹n ≤ n_y›
        by (metis antisym_conv2 less_imp_Suc_add
                  relcomppE relpowp_Suc_E
                  relpowp_add)
      then have ‹n ∈ suborders x›
        using subordersI by blast
      then show ‹False›
        using lt by simp
    next
      case (gt n)
      then obtain z where ‹x ◁◁^n z›
        using subordersE by blast
      then have ‹y ◁◁^(n - n_y) z›
        using inherence_mid_point
              ‹x ◁◁^n_y y› gt
        by auto
      then have ‹n - n_y = 0›
        using A by metis
      then show ‹False›
        using ‹n_y < n› by auto
    qed
  qed
  then show ‹finite (suborders x)›
    by simp
next
```

```
  assume ‹finite (suborders x)›
  then obtain n
    where
      A: ‹n ∈ suborders x›
         ‹⋀m. m ∈ suborders x ⟹ m ≤ n›
    by (metis Max_ge Max_in emptyE
              relpowp_0_I subordersI)
  then obtain y where ‹x ◁◁^n y›
    by blast
  then have ‹y ∈ ℰ›
    by (metis assms endurantI1 gr0I
             inheres_in_by_scope
            nth_bearer_eqI zeroth_bearer)
  have ‹y ∉ ℳ›
  proof
    assume ‹y ∈ ℳ›
    then obtain z where ‹y ◁ z› by blast
    then have ‹x ◁◁^(Suc n) z›
      using ‹x ◁◁^n y› by auto
    then have ‹Suc n ∈ suborders x› by blast
    then have ‹Suc n ≤ n› using A(2) by blast
    then show ‹False› by simp
  qed
  then have ‹y ∈ 𝒮› using ‹y ∈ ℰ› by blast
  have ‹x ◁◁ y› using ‹x ◁◁^n y›
    by (metis ‹y ∉ ℳ› assms
          relpowp_imp_rtranclp rtranclpD)
  then show ‹∃y. is_an_ultimate_bearer_of y x›
    using ‹y ∈ 𝒮›
      is_an_ultimate_bearer_of_def
      by (meson tranclp_into_rtranclp)
qedlemma no_ultimate_bearer_problem_eq_noetherian:
  shows ‹¬ has_moment_without_ultimate_bearer ⟷
         wfP (◁)^-1-1›
proof -
  have S1: ‹¬ has_moment_without_ultimate_bearer ⟷
            (∀x ∈ ℳ. ∃y. is_an_ultimate_bearer_of y x)›
    by (auto simp: has_moment_without_ultimate_bearer_def)
  show ‹?thesis›
    when ‹wfP (◁)^-1-1 ⟷ (∀x. finite (suborders x))›
    using has_ultimate_bearer_iff_suborders_finite
          S1
```

```
  by (metis ℳ_def diff_is_0_eq
        finite_nat_set_iff_bounded
        finite_nat_set_iff_bounded_le gr0I
        mem_Collect_eq relpowp_E2 subordersE that)
have R1: ‹wfP (◁)⁻¹⁻¹›
  if as: ‹∀x. finite (suborders x)›
proof (intro wfI_min[to_pred])
  fix x :: ‹'p› and Q
  assume ‹x ∈ Q›
  define X where ‹X ≡ { n | y n . y ∈ Q ∧ x ◁◁^n y }›
  have X_subset_suborder:
    ‹X ⊆ suborders x›
    using X_def subordersI by blast
  then have ‹finite X›
    using infinite_super as by auto
  define nmax
    where ‹nmax ≡ Max X›
  obtain nmax:
    ‹nmax ∈ X› ‹⋀n. n ∈ X ⟹ n ≤ nmax›
    using ‹finite X›
    by (metis (full_types, lifting)
        Max_ge Max_in X_def
        ‹x ∈ Q› empty_Collect_eq nmax_def
        relpowp_0_I)
  obtain z
    where ‹z ∈ Q› ‹x ◁◁^nmax z›
    using ‹nmax ∈ X› X_def by auto
  have ‹∀y. (◁)⁻¹⁻¹ y z ⟶ y ∉ Q›
  proof (auto)
    fix y
    assume ‹z ◁ y› ‹y ∈ Q›
    then have A: ‹x ◁◁^(Suc nmax) y›
      using ‹x ◁◁^nmax z› by auto
    then have B: ‹Suc nmax ∈ suborders x›
      using subordersI by blast
    have C: ‹Suc nmax ∈ X›
      using A X_def ‹y ∈ Q› by blast
    then have ‹Suc nmax ≤ nmax›
      using nmax(2) by simp
    then show ‹False› by simp
  qed
  then
```

```
    show ‹∃z∈Q. ∀y. (◁)⁻¹⁻¹ y z ⟶ y ∉ Q›
      using ‹z ∈ Q› by blast
  qed
  have R2: ‹∀x. finite (suborders x)›
    if as: ‹wfP (◁)⁻¹⁻¹›
  proof (rule ccontr ; simp)
    assume ‹∃x. infinite (suborders x)›
    then obtain x
      where ‹infinite (suborders x)› by blast
    have ‹suborders x = UNIV›
    proof (intro set_eqI ; simp)
      fix n
      show ‹n ∈ suborders x›
      proof (rule ccontr)
        assume ‹n ∉ suborders x›
        then have ‹¬ x ◁◁ⁿ y›
          for y by blast
        then have ‹¬ x ◁◁ⁿ' y›
          if ‹n ≤ n'›  for y n'
          by (metis antisym_conv2
                    less_imp_Suc_add
                    relcomppE relpowp_Suc_E
                    relpowp_add that)
        then have ‹n' < n›
          if ‹n' ∈ suborders x› for n'
          using subordersE that
          by (meson leI)
        then have ‹finite(suborders x)›
          using bounded_nat_set_is_finite
          by blast
        then show ‹False›
          using ‹infinite (suborders x)›
          by simp
      qed
    qed
    then have sb_ex:
      ‹∃y. x ◁◁ⁿ y› for n
      using subordersE by blast
    have sb_ex1: ‹∃!y. x ◁◁ⁿ y› for n
    proof (intro ex_ex1I sb_ex)
      fix a b
      assume ‹x ◁◁ⁿ a› ‹x ◁◁ⁿ b›
```

```
      then show ‹a = b›
        using nth_inherence_unique_cond
        by simp
    qed
    then obtain f
      where f: ‹x ◁◁^n f n›
      for n by moura
    have f_suc: ‹f n ◁ f (Suc n)› for n
    proof -
      obtain AA: ‹x ◁◁^n f n›
                 ‹x ◁◁^(Suc n) f (Suc n)›
        using f by metis
      obtain k where ‹x ◁◁^n k› ‹k ◁ f (Suc n)›
        using AA(2)
        by auto
      then have ‹k = f n›
        using nth_inherence_unique_cond AA(1)
        by blast
      then show ‹?thesis›
        using ‹k ◁ f (Suc n)› by simp
    qed

    define X where ‹X ≡ { f n | n . True }›
    have ‹x ∈ X› using X_def f
      by (metis (mono_tags, lifting)
                CollectI relpowp.simps(1))
    obtain z
      where ‹z ∈ X›
        and z: ‹⋀y. z ◁ y ⟹ y ∉ X›
      using wfE_min[to_pred,OF as]
            ‹x ∈ X›
      by (simp ; metis)
    then obtain n_z
      where ‹z = f n_z›
      using X_def by blast
    then have ‹z ◁ f (Suc n_z)›
      using f_suc by simp
    then show ‹False›
      using z X_def by auto
  qed
  then show ‹wfP (◁)^-1-1 ⟷
              (∀x. finite (suborders x))›
```

```
    using R1 by metis
qed

end
```

B.2.4 Finite-Chain Inherence

```
locale noetherian_inherence =
    inherence_base +
  assumes
    inherence_is_noetherian[intro!,simp]:
      ‹wfP (◁)⁻¹⁻¹›
beginlemmas just_noetherian_inherence_axioms =
    inherence_is_noetherian

lemmas all_noetherian_inherence_axioms =
    all_inherence_base_axioms
    just_noetherian_inherence_axiomslemma inherence_is_acyclic[intro!]:
  ‹acyclicP (◁)›
  using acyclicP_converse
        wfP_acyclicP
        inherence_is_noetherian
  by metislemma inherence_is_irrefl: ‹¬ x ◁ x›
  by (metis inherence_is_acyclic
            acyclic_irrefl[to_pred]
            irreflp_def tranclp.r_into_trancl)lemma inherence_is_asymm:
  ‹x ◁ y ⟹ ¬ y ◁ x›
  using tranclp_trans inherence_is_acyclic
        acyclic_irrefl[to_pred]
        irreflp_def tranclp.r_into_trancl
  by metislemma inherence_intransitive_V1:
  ‹⟦ x ◁ y ; y ◁ z ⟧ ⟹ ¬ x ◁ z›
  using inherence_is_irrefl
        inherence_is_asymm
        moment_non_migration
  by metislemma unique_order:
  ‹n = n'› if assms: ‹x ◁◁^n y› ‹x ◁◁^n' y›
proof (rule ccontr)
  assume ‹n ≠ n'›
  have R: ‹False›
    if as: ‹x ◁◁^a y› ‹x ◁◁^b y›
           ‹a < b›
```

```
    for x y a b
  proof -
    have ‹a ≤ b› using ‹a < b› by auto
    then have ‹y ◁◁^(b - a) y›
      using inherence_mid_point as by metis
    then have ‹y ◁◁ y›
      using ‹a < b›
      by (meson tranclp_power zero_less_diff)
    then have ‹cyclic (◁)›
      using cyclic_def by auto
    then show ‹False›
      using
        cyclic_inherence_has_moment_witout_ultimate_bearer
        no_ultimate_bearer_problem_eq_noetherian
      by blast
  qed
  consider ‹n < n'› | ‹n' < n›
    using ‹n ≠ n'› nat_neq_iff by blast
  then show ‹False›
    apply cases
    using R assms by auto
qedlemmaultimate_bearer_ex1I:
  ‹∃!s. x ◁◁ s ∧ s ∈ 𝒮›
  (is ‹∃!s. ?P s›)
  if ‹x ∈ ℳ›
proof -
  have R1: ‹∃s. ?P s›
  proof -
    have ‹¬(∀n. ∃y. x ◁◁^n y)›
      using
      that
      no_ultimate_bearer_problem_eq_noetherian
      inherence_is_noetherian
      has_moment_without_ultimate_bearerI
      has_moment_without_ultimate_bearerE
      by meson
    then have A: ‹∃n. ∀y. ¬ x ◁◁^n y›
      by blast
    define N where ‹N = { n . ∀y. ¬ x ◁◁^n y }›
    then have ‹N ≠ ∅›
      using A by blast
    define nmin where
```

```
    ‹nmin ≡ LEAST n. ∀y. ¬ x ◁◁^n y›
    (is ‹nmin ≡ LEAST n. ?Q n›)
  have ‹∃n. ?Q n› using A by blast
  then have B: ‹¬ x ◁◁^nmin y› for y
    using LeastI_ex[of ‹?Q›] nmin_def
    by metis
  have C: ‹nmin ≤ n› if ‹?Q n› for n
    using Least_le[of ‹?Q›] nmin_def that
    by metis
  have D: ‹0 < nmin›
    using B  by fastforce
  then obtain nn where nn: ‹nmin = Suc nn›
    using Suc_pred' by blast
  then have ‹¬ ?Q nn› using C[of ‹nn›]
    using Suc_n_not_le_n by blast
  then obtain y where E: ‹x ◁◁^nn y›
    by blast
  then have ‹y ∈ ℰ›
    using inheres_in_by_scope that
    by (metis endurantI1 nth_bearer_eqI
              nth_bearer_range')
  have ‹¬ y ◁ z› for z
  proof
    assume ‹y ◁ z›
    then have ‹x ◁◁^Suc nn z›
      using E by auto
    then show ‹False›
      using B nn by simp
  qed
  then have ‹y ∈ 𝒮› using ‹y ∈ ℰ› 𝒮_I by auto
  then show ‹?thesis› using E
    by (metis 𝒮_E relpowp_imp_rtranclp
              rtranclpD that)
qed
have R2: ‹z = y›
  if as: ‹x ◁◁ z› ‹z ∈ 𝒮›
         ‹x ◁◁ y› ‹y ∈ 𝒮›
  for z y
proof -
  obtain n_z where Z: ‹0 < n_z› ‹x ◁◁^n_z z›
    using ‹x ◁◁ z› tranclp_power
    by metis
```

```
    obtain n_y where Y: ‹0 < n_y› ‹x ◁◁^n_y y›
      using ‹x ◁◁ y› tranclp_power
      by metis
    have A: ‹a = b›
      if ‹x ◁◁^n a› ‹x ◁◁^n' b›
         ‹n ≤ n'› ‹a ∈ 𝒮›
      for n n' a b
    proof -
      have AA: ‹a ◁◁^(n' - n) b›
        using that inherence_mid_point by simp
      have ‹n = n'›
      proof (rule ccontr)
        assume ‹n ≠ n'›
        then have ‹n < n'›
          using ‹n ≤ n'› by auto
        then have ‹a ◁◁ b›
          using AA
          by (meson 𝒮_E inheres_in_by_scope
                    that(4) zero_less_diff)
        then obtain k where ‹a ◁ k›
          using ‹a ∈ 𝒮› trans_inheres_in_scope
          by blast
        then have ‹a ∈ ℳ› by blast
        then show ‹False›
          using ‹a ∈ 𝒮› 𝒮_E by blast
      qed
      then show ‹a = b› using AA by simp
    qed
    consider ‹n_z ≤ n_y› | ‹n_y ≤ n_z›
      using nat_le_linear by blast
    then show ‹?thesis›
      apply cases
      using A Y(2) Z(2) that(2,4) by auto
  qed
  then show ‹?thesis›
    using ex_ex1I[OF R1] by metis
qed

lemma order_ultimate_bearer_ex1I:
  ‹∃!(n,y). y ∉ ℳ ∧ x ◁◁^n y›
proof(cases ‹x ∈ ℳ›)
  assume ‹x ∉ ℳ›
```

```
  show ‹?thesis›
  proof (intro ex1I[of _ ‹(0,x)›]
        ; auto simp: ‹x ∉ 𝓜›)
    fix y n
    assume ‹x ◁◁^n y›
    show ‹n = 0›
    proof (rule ccontr)
      assume ‹n ≠ 0›
      then obtain ‹x ∈ 𝓜›
        using ‹x ◁◁^n y› inheres_in_by_scope
        by blast
      then show ‹False›
        using ‹x ∉ 𝓜› by simp
    qed
    then show ‹y = x›
      using ‹x ◁◁^n y› by simp
  qed
next
  assume ‹x ∈ 𝓜›
  then have A: ‹∃!s. x ◁◁ s ∧ s ∈ 𝒮›
    using ultimate_bearer_ex1I by simp
  then obtain s where B: ‹x ◁◁ s› ‹s ∈ 𝒮›
    by blast
  then have ‹s ∉ 𝓜› by blast
  obtain n where C: ‹x ◁◁^n s› ‹0 < n›
    using tranclp_power B(1) by metis
  have n_unique: ‹n = n'›
    if as: ‹x ◁◁^n' s'› ‹s' ∈ 𝒮›
    for n' s'
  proof -
    have ‹s = s'›
      using A B as
      by (metis 𝒮_E ‹x ∈ 𝓜›
          relpowp_imp_rtranclp rtranclpD)
    then have AA: ‹x ◁◁^n' s›
      using as(1) by simp
    then show ‹n = n'›
      using C unique_order by simp
  qed
  show ‹∃!(n,y). y ∉ 𝓜 ∧ x ◁◁^n y›
  proof (intro ex1I[of _ ‹(n,s)›]
        ; (auto simp add: ‹s ∉ 𝓜› ‹x ◁◁^n s›)?)
```

```
    fix n' y
    assume ‹y ∉ ℳ› ‹x ◁◁^n' y›
    then have ‹n' ≠ 0›
      using ‹x ∈ ℳ› A by auto
    then have ‹0 < n'›  by auto
    then have ‹x ◁◁ y›
      using ‹x ◁◁^n' y› tranclp_power
      by metis
    then have ‹y ∈ ℰ›
      using ‹x ◁◁^n' y› ‹0 < n'› inheres_in_by_scope
      by metis
    then have ‹y ∈ 𝒮›
      using ‹y ∉ ℳ› by blast
    then show ‹n' = n›
      using n_unique ‹x ◁◁^n' y› by simp
    show ‹y = s›
      by (metis C(1) ‹n' = n›
                ‹x ◁◁^n' y› nth_bearer_eqI)
  qed
qedlemma order_eq_simp:
  assumes ‹x ∈ ℰ›
  shows ‹order x = n ⟷ (∃y. y ∈ 𝒮 ∧ x ◁◁^n y)›
proof -
  have A: ‹∃!n. has_order x n›
    using assms
    by (metis 𝒮_I all_noetherian_inherence_axioms(5)
         has_moment_without_ultimate_bearer_def
         no_ultimate_bearer_problem_eq_noetherian
         is_an_ultimate_bearer_of_def
         rtranclp.rtrancl_refl
         ultimate_bearer_unique_order)
  then have B: ‹order x = n›
    if ‹has_order x n› for n
    using order_def the1_equality that by metis
  have C: ‹has_order x (order x)›
    using the1I2 A order_def
    by metis
  have D: ‹has_order x n›
    if ‹order x = n› for n
    using that C A by metis
  then have E:
      ‹order x = n ⟷ has_order x n› for n
```

```
    using B by metis
  show ?thesis
    apply (simp add: E has_order_def
            is_an_ultimate_bearer_of_def)
    apply (intro iffI ; elim exE conjE)
    subgoal by blast
    subgoal for y
      by (intro exI[of _ y] conjI
          ; simp add: relpowp_imp_rtranclp)
    done
qedlemma ultimate_bearer_eq_simp:
  assumes ‹x ∈ ℰ›
  shows ‹!β x = y ⟷ y ∈ 𝒮 ∧ (◁)** x y›
proof -
  have A: ‹∃!z. is_an_ultimate_bearer_of z x›
    using assms apply (cases x)
    subgoal S
      apply (intro ex1I[of _ x] ;
             simp add: is_an_ultimate_bearer_of_def)
      by (metis inherence_sig.𝒮_E rtranclpD
            trans_inheres_in_scope)
    subgoal M premises p
      supply R1 = ultimate_bearer_ex1I[OF p]
      apply (rule ex1E[OF R1] ; elim conjE)
      subgoal for z
        apply (intro ex1I[of _ z])
        subgoal G1
          apply (thin_tac X for X)
          by (simp add:
              is_an_ultimate_bearer_of_def[of z x])
        subgoal premises P for q
          using G1 P(1) P(2) P(3) P(4) assms
              ultimate_bearer_unique by blast
        done
      done
    done
  note ultimateBearer_def[of x]
  note B = the1_equality[OF A,of y,
      simplified ultimateBearer_def[symmetric]]
  note C = the1I2[of "λz. is_an_ultimate_bearer_of z x"
        , simplified ultimateBearer_def[symmetric]
        , where ?Q="λb.  (b = y) = (y ∈ 𝒮 ∧ (◁)** x y)"
```

```
        , OF A]
  show ?thesis
    apply (intro C)
    subgoal for z
      using A
      by (auto simp add: is_an_ultimate_bearer_of_def)
    done
qed

lemma
  assumes ‹x ∈ ℰ›
  shows
    ultimate_bearer_and_order[intro!]: ‹x ◁◁^order x !β x›
  and
    ultimate_bearer_is_not_a_moment[simp]: ‹!β x ∉ ℳ›
  subgoal G1
    using assms
    by (metis order_eq_simp relpowp_imp_rtranclp
          ultimate_bearer_eq_simp)
  subgoal G2
    using assms
    by (meson 𝒮_E ultimate_bearer_eq_simp)
  donelemma ultimate_bearer_substantial[simp]:
  ‹x ∈ ℰ ⟹ !β x ∈ 𝒮›
  using ultimate_bearer_eq_simp by blastlemma ultimate_bearer_eqI1:
‹!β x = !β y› if as: ‹x ◁◁ y›
proof -
  obtain ‹x ∈ ℳ› ‹y ∈ ℰ›
    using as trans_inheres_in_scope
    by blast
  then have ‹x ∈ ℰ› by blast
  obtain n_x where A: ‹x ◁◁^n_x y› ‹0 < n_x›
    using as tranclp_power by metis
  then have B: ‹(◁)** x y›
    using rtranclp_power by metis
  obtain C: ‹y ◁◁^order y !β y› ‹!β y ∉ ℳ›
    using ultimate_bearer_and_order
      ultimate_bearer_is_not_a_moment
      ‹x ∈ ℰ› ‹y ∈ ℰ›
    by auto
  then have D: ‹(◁)** y (!β y)›
    using rtranclp_power by metis
```

```
  then have E: ‹(◁)** x (!β y)›
    using rtranclp_trans B by metis
  then show ‹!β x = !β y›
    apply (subst ultimate_bearer_eq_simp[
            OF ‹x ∈ ℰ›,of ‹!β y›]
          ; intro conjI ; simp)
    using C(2) ‹y ∈ ℰ› by simp
qed

lemma order_diff:
  ‹n = order y - order x› if assms: ‹y ◁◁^n x›
proof (cases ‹n = 0›)
  assume ‹n = 0›
  then have ‹x = y›
    using assms by auto
  then show ‹?thesis›
    using ‹n = 0› by simp
next
  assume ‹n ≠ 0›
  then have ‹0 < n› by blast
  then have ‹y ◁◁ x›
    using assms tranclp_power
    by metis
  then have byx[simp]: ‹!β y = !β x›
    using ultimate_bearer_eqI1 by simp
  have ‹y ∈ ℰ› using ‹y ◁◁ x›
    by (meson endurantI1 trans_inheres_in_scope)
  then have ordY: ‹y ◁◁^order y !β y›
    using ultimate_bearer_and_order by blast
  then have A: ‹y ◁◁^order y !β x›
    by simp
  have ‹x ∈ ℰ›  using ‹y ◁◁ x›
    by (meson endurantI1 trans_inheres_in_scope)
  then have B: ‹x ◁◁^order x !β x›
    by auto
  then have C: ‹y ◁◁^n + order x !β x›
    by (metis relcompp.relcompI assms relpowp_add)
  then have D: ‹order y = n + order x›
    using unique_order ordY by simp
  then show ‹n = order y - order x›
    by presburger
qed
```

```
lemma order_le: ‹order x < order y›
  if assms: ‹y ◁◁ x›
proof -
  obtain n where A: ‹0 < n› ‹y ◁◁^n x›
    by (meson assms tranclp_power)
  then have ‹n = order y - order x›
    using order_diff by simp
  then have ‹0 < order y - order x›
    using A by presburger
  then show ‹?thesis› by presburger
qed

lemma order_le1:
  ‹order x ≤ order y› if assms: ‹(◁)** y x›
  by (metis Nitpick.rtranclp_unfold eq_refl
       less_imp_le_nat order_le that)

lemma order_le2:
  ‹order x < order y› if assms: ‹y ◁ x›
  using order_le tranclp.intros(1) assms
  by metis

lemma inheres_in_by_order:
  ‹n ≤ order x› if assms: ‹x ◁◁^n y›
proof (rule ccontr)
  have ‹order y ≤ order x›
    using order_le1 assms rtranclp_power by metis
  assume ‹¬ n ≤ order x›
  then have ‹order x < n› by auto
  then obtain n' where A: ‹n = order x + n'›
    using less_imp_add_positive by blast
  have B: ‹n = order x - order y›
    using order_diff assms by simp
  then have ‹order x + n' = order x - order y›
    using A by simp
  then have ‹n' = 0 - order y› by simp
  then have ‹order y = 0›
    using B ‹order x < n› by linarith
  then have ‹order x = n› using B by simp
  then show ‹False›
    using ‹order x < n› by simp
```

```
qed

lemma nth_bearer_order_iff[simp]:
  assumes ‹x ∈ ℰ›
  shows ‹x ◁◁^n #β x n ⟷ n ≤ order x›
proof (intro iffI
        inheres_in_by_order[of _ _ ‹#β x n›]
       ; simp?)
  assume  A: ‹n ≤ order x›
  then obtain n' where B: ‹order x = n + n'›
    using le_Suc_ex by blast
  have C: ‹x ◁◁^order x !β x›
    using assms by blast
  then have D: ‹x ◁◁^n + n' !β x›
    using B by simp
  then obtain y where E: ‹x ◁◁^n y›
    by (metis relcomppE relpowp_add)
  then  have ‹#β x n = y›
    using nth_bearer_eqI by simp
  then show ‹x ◁◁^n #β x n› using E by simp
qed

lemma substantial_order:
  assumes ‹x ∈ 𝒮›
  shows ‹order x = 0›
  using assms 𝒮_def
  by (simp add: order_eq_simp)

lemma moment_order_Suc:
  assumes ‹x ◁ y›
  shows ‹order x = Suc (order y)›
proof -
  obtain ‹x ∈ ℰ› ‹y ∈ ℰ›
    using inherence_scope assms by blast
  then obtain
    A: ‹x ◁◁^order x !β x›
       ‹y ◁◁^order y !β y›
    by blast
  then have B: ‹y ◁◁^order y !β x›
    using assms ultimate_bearer_eqI1[of x y]
    by (simp add: tranclp.r_into_trancl)
  have C: ‹x ◁◁^1 y›
```

```
    using assms by auto
  have D: ‹1 ≤ order x›
    using C inheres_in_by_order by blast
  have E: ‹y ◁◁^order x - 1 !β x›
    using inherence_mid_point C A(1) D by metis
  then have ‹order y = order x - 1›
    using A(2)
    using B unique_order by blast
  then show ‹order x = Suc (order y)›
    using D by linarith
qedendsublocale
  noetherian_inherence ⊆ inherence_original
proof (unfold_locales)
  have acyclic_conv: ‹acyclicP ((◁)^-1-1)›
    using wfP_acyclicP
          inherence_is_noetherian
    by blast
  then have acyclic: ‹acyclicP (◁)›
    using acyclicP_converse by auto
  then have
    inh_acyclic: ‹¬ (◁)^++ x x›  for x
    by (simp add: Nitpick.tranclp_unfold
                  acyclic_def)
  then show
    inh_irrefl: ‹¬ x ◁ x› for x by blast
  show asym: ‹¬ y ◁ x› if ‹x ◁ y› for x y
  proof
    assume ‹y ◁ x›
    then obtain ‹(◁)** x y› ‹(◁)** y x›
      using ‹x ◁ y› by blast
    then have ‹x = y›
      by (metis acyclic_impl_antisym_rtrancl[to_pred]
                acyclic antisymD[to_pred])
    then show ‹False›
      using inh_irrefl ‹x ◁ y› by simp
  qed
  show ‹¬ x ◁ z›
    if  A: ‹x ◁ y› ‹y ◁ z› for x y z
    using A
    by (meson inherence_intransitive_V1)
qedsublocale noetherian_inherence ⊆ inherence_V2
proof (unfold_locales)
```

```
  have acyclic_conv: ‹acyclicP ((◁)⁻¹⁻¹)›
    using wfP_acyclicP inherence_is_noetherian
    by blast
  then have acyclic: ‹acyclicP (◁)›
    using acyclicP_converse by auto
  then show ‹¬ (◁)⁺⁺ x x›  for x
    by (simp add: Nitpick.tranclp_unfold acyclic_def)
qedsublocale
  noetherian_inherence ⊆ inherence_V3
proof (unfold_locales)
  have acyclic_conv: ‹acyclicP ((◁)⁻¹⁻¹)›
    using wfP_acyclicP inherence_is_noetherian
    by blast
  then have acyclic: ‹acyclicP (◁)›
    using acyclicP_converse by auto
  show ‹¬ inf_inh_asc_chain X x› for X x
  proof (auto simp: inf_inh_asc_chain_def)
    assume ‹∀y. y ∈ X ⟷ (◁)** x y›
    then have X_eq[simp]:
        ‹X = { y . (◁)** x y }› by blast
    then have ‹x ∈ X› by auto
    consider ‹x ∉ ℰ› | ‹x ∈ 𝒮› | ‹x ∈ ℳ›
      using 𝒮_def by auto
    then show ‹finite X›
    proof (cases)
      assume ‹x ∉ ℰ›
      then have ‹X = {x}›
        using X_eq inherence_scope
        by (smt Collect_cong Nitpick.rtranclp_unfold
            empty_iff inherence_base.endurantI1
            inherence_base.trans_inheres_in_scope
            inherence_base_axioms insert_compr)
      then show ‹finite X› by auto
    next
      assume ‹x ∈ 𝒮›
      then have ‹¬ x ◁ y› for y
        using 𝒮_def ℳ_def by blast
      then have ‹¬ (◁)⁺⁺ x y› for y
        by (meson tranclpD)
      then have ‹y = x› if ‹(◁)** x y› for y
        by (metis Nitpick.rtranclp_unfold that)
      then have ‹X = {x}› using X_eq by auto
```

```
    then show ‹finite X› by auto
  next
    assume ‹x ∈ 𝒩›
    have A1:
      ‹∀Q. (∃x. x ∈ Q) ⟶
            (∃z∈Q. ∀y. (◁)⁺⁺ z y ⟶ y ∉ Q)›
      using inherence_is_noetherian[THEN wfP_trancl]
      by (simp add:
            tranclp_converse wf_eq_minimal[to_pred]
          ; metis)
    have A2: ‹P›
      if assms:
          ‹t ∈ X›
          ‹⋀z. ⟦ z ∈ X ; ⋀y. (◁)⁺⁺ z y ⟹ y ∉ X ⟧ ⟹ P›
        for t P
    proof -
      obtain z where
        AA: ‹z ∈ X› ‹∀y. (◁)⁺⁺ z y ⟶ y ∉ X›
        using assms A1 by metis
      then show ‹P› using AA assms by metis
    qed
    show ‹finite X›
    proof (rule A2[of ‹x›,OF ‹x ∈ X›])
      fix z
      assume as: ‹z ∈ X› ‹⋀y. (◁)⁺⁺ z y ⟹ y ∉ X›
      then have AA: ‹(◁)** x z›
        using X_eq by auto
      then obtain n_z
        where nz: ‹((◁)^^n_z) x z›
        by (meson rtranclp_power)
      then have ‹z ∈ X›
        using X_eq as(1) by blast
      have ‹n ≤ n_z› if ‹((◁)^^n) x y› for y n
      proof (rule ccontr)
        assume ‹¬ n ≤ n_z›
        then have ‹n_z < n› by simp
        then have ‹n_z ≤ n› by auto
        then have ‹((◁)^^(n-n_z)) z y›
          using inherence_mid_point
                ‹((◁)^^n) x y› nz
          by blast
        then obtain k where ‹z ◁ k›
```

```
      using ‹n_z < n›
      by (meson ‹¬ n ≤ n_z›
          diff_is_0_eq relpowp_E2)
    then have ‹(◁)^++ z k› by simp
    then have ‹k ∉ X›
      using as(2) by simp
    have ‹(◁)^++ x k›
      using ‹(◁)^++ z k› ‹((◁)^^n_z) x z› AA
      by auto
    then have ‹k ∈ X›
      using X_eq by simp
    then show ‹False›
      using ‹k ∉ X› by simp
  qed
  define Y where ‹Y = { n . n ≤ n_z}›
  have ex_bearer:
    ‹∃y. ((◁)^^n) x y› if ‹n ∈ Y› for n
  proof -
    have ‹n_z = n + (n_z - n)›
      using that Y_def by simp
    then have ‹((◁)^^(n + (n_z - n))) x z›
      using nz by simp
    then have ‹(((◁)^^n) OO ((◁)^^(n_z - n))) x z›
      by (simp add: relpowp_add)
    then obtain t
      where ‹((◁)^^n) x t› ‹((◁)^^(n_z - n)) t z›
      by blast
    then show ‹?thesis› by metis
  qed
  have ex1_bearer:
    ‹∃!y. ((◁)^^n) x y› if ‹n ∈ Y› for n
    apply (intro ex_ex1I ex_bearer that)
    using nth_inherence_unique_cond by auto
  define F where
    ‹F n ≡ THE y. ((◁)^^n) x y› for n
  have ‹F n ∈ X› if ‹n ∈ Y› for n
  proof -
    have A: ‹∃!y. ((◁)^^n) x y›
      using ex1_bearer Y_def that by blast
    then obtain k where B: ‹((◁)^^n) x k›
      by blast
    then have F_eq: ‹F n = k›
```

```
      apply (simp only: F_def)
      using the1_equality[OF A B] by metis
    then have ‹((◁)^^n) x (F n)›
      using B F_eq by simp
    then show ‹F n ∈ X›
      using X_eq rtranclp_power
      by fastforce
  qed
  have exY: ‹∃n ∈ Y. F n = y› if ‹y ∈ X› for y
  proof (simp add: Y_def F_def)
    have ‹(◁)** x y›
      using that X_eq by blast
    then obtain n_y where ‹((◁)^^n_y) x y›
      by (meson rtranclp_imp_relpowp)
    have ‹n_y ≤ n_z›
    proof (rule ccontr)
      assume as1: ‹¬ n_y ≤ n_z›
      then have ‹n_z < n_y› by simp
      then have ‹n_z ≤ n_y› by simp
      then have ‹((◁)^^(n_y-n_z)) z y›
        using inherence_mid_point
              ‹((◁)^^n_y) x y› ‹((◁)^^n_z) x z›
        by metis
      then obtain k
        where ‹z ◁ k› using ‹n_z < n_y›
        by (meson as1 diff_is_0_eq relpowp_E2)
      then have ‹(◁)++  z k› by simp
      then have ‹k ∉ X› using as(2) by simp
      have ‹(◁)** x k›
        using ‹((◁)^^n_z) x z› ‹(◁)++  z k› AA
        by auto
      then have ‹k ∈ X› using X_eq by blast
      then show ‹False› using ‹k ∉ X› by simp
    qed
    then have ‹n_y ∈ Y› using Y_def by auto
    then have ‹The (((◁) ^^ n_y) x) = y›
      using ‹((◁)^^n_y) x y›
      by (simp add: nth_inherence_unique_cond
                the_equality)
    then show ‹∃n≤n_z. The (((◁) ^^ n) x) = y›
        using ‹n_y ≤ n_z› by blast
  qed
```

```
        have F_img: ‹F ` Y = X›
          using exY ‹⋀n. n ∈ Y ⟹ F n ∈ X› by blast
        have finite_Y: ‹finite Y› by (simp add: Y_def)
        show ‹finite X›
          using finite_surj finite_Y F_img by blast
      qed
    qed
  qed
qed

lemma (in inherence_V3)
  inherence_V3_impl_noetherian_inherence:
  ‹noetherian_inherence 𝒲 (◁)›
proof (unfold_locales)
  have ‹¬ has_moment_without_ultimate_bearer›
    using no_ultimate_bearer_problem by blast
  then show ‹wfP (◁)⁻¹⁻¹›
    using no_ultimate_bearer_problem_eq_noetherian
    by linarith
qedlemma inherence_V3_and_noetherian_inherence_are_eq:
  ‹inherence_V3 𝒲 inheresIn ⟷
    noetherian_inherence 𝒲 inheresIn›
  (is "?G1 ⟷ ?G2")
proof
  show A: "?G2" if "?G1"
    using inherence_V3.inherence_V3_impl_noetherian_inherence
          that
    by metis
  show B: "?G1" if "?G2"
  proof -
    interpret noetherian_inherence 𝒲 inheresIn
      using that by simp
    show "?G1"
      by (simp add: inherence_V3_axioms)
  qed
qed

locale inherence =
  noetherian_inherence +
  assumes
    inherence_is_wf[intro!,simp]: ‹wfP (◁)›
begin lemmas just_inherence_axioms =
```

```
  inherence_is_wf

lemmas all_inherence_axioms =
  all_noetherian_inherence_axioms
  just_inherence_axioms

lemma ultimte_bearer_ed:
    ‹ed x (!β y)› if ‹ed x y› for x y
    apply (auto simp: ed_def)
    subgoal G1 using that edE by auto
    subgoal G2
      apply (rule edE[OF that])
      by (simp add: endurantI3)
    subgoal G3 for w
      apply (rule edE[OF that])
      subgoal premises P
        supply R1 = P(5)[OF P(1,2)]
        using R1 P(1,4)
        apply (induct y rule: wfP_induct[
                OF inherence_is_noetherian]
              ; simp)
        subgoal premises Q for z
          using  Q(3)
          apply (cases z rule: endurant_cases
                 ; (elim 𝒩_E)?)
          subgoal G3_1
            using Q(2) Q(5)
            by (metis Q(3) rtranclp.rtrancl_refl
                ultimate_bearer_eq_simp)
          subgoal G3_2 premises T for t
            apply (rule edE[
                    OF inherence_imp_ed[OF T(2)]])
            subgoal premises V
              supply R2 = V(3)[OF ‹w ∈ 𝒲› ‹z ∈ w›]
              supply R3 = inherence_imp_ed[OF T(2)]
              supply R4 = ed_scope[OF R3] edD[OF R3]
              supply R5 = Q(1)[rule_format,
                               OF T(2) R2 R4(2)]
                               ‹z ◁ t› ‹z ∈ ℰ›
                               ‹t ∈ ℰ› ‹z ∈ w›
                               ‹t ∈ w› ‹w ∈ 𝒲›
              supply R6 = ultimate_bearer_eq_simp[
```

```
                    of z ‹!β t›,
                    THEN iffD2,
                    OF _ conjI,
                    symmetric,
                    THEN subst,
                    of ‹λx. x ∈ w›]
            apply (rule R6)
            subgoal G3_2_1
              by (simp add: V(1))
            subgoal G3_2_2
              using ultimate_bearer_is_not_a_moment
              by (simp add: V(2))
            subgoal G3_2_3
              using R5(2) ultimate_bearer_eqI1
                    ultimate_bearer_eq_simp
              by blast
            using R5(1) by simp
          done
        done
      done
    done
  done

lemma ultimate_beaer_idemp[simp]:
  assumes ‹x ∈ ℰ›
  shows ‹!β (!β x) = !β x›
  using assms
  by (metis rtranclpD
      ultimate_bearer_eqI1
      ultimate_bearer_eq_simp)lemma inherence_chains_are_finite:
  assumes A: ‹X ⊆ ℰ› ‹X ≠ ∅›
    ‹⋀x y. ⟦ x ∈ X ; y ∈ X ⟧ ⟹
        x ◁◁ y ∨ y ◁◁ x ∨ x = y›
  shows ‹finite X›
proof -
  have B: ‹inj_on order X›
  proof (intro inj_onI)
    fix x y
    assume AA: ‹x ∈ X› ‹y ∈ X›
               ‹order x = order y›
    { assume ‹x ≠ y›
      then consider
```

```
      (c1) ‹x ◁◁ y› | (c2) ‹y ◁◁ x›
      using A(3) AA(1,2) by metis
    then have False
    proof (cases)
      case c1
      then have ‹order x < order y›
        by (metis AA(3) order_le)
      then show ?thesis
        using AA by simp
    next
      case c2
      then have ‹order x < order y›
        by (metis AA(3) order_le)
      then show ?thesis
        using AA by simp
    qed }
  then show ‹x = y› by blast
qed
obtain a
  where Ca: ‹a ∈ X› ‹⋀y. y ◁◁ a ⟹ y ∉ X›
  using A(2) inherence_is_wf wfP_eq_minimal
  by (metis equals0I wfP_trancl)

obtain b
  where Cb: ‹b ∈ X› ‹⋀y. b ◁◁ y ⟹ y ∉ X›
  using A(2)
        inherence_is_noetherian wfP_eq_minimal
  by (metis conversep.intros empty_subsetI
          subset_antisym
          subset_iff tranclp_converseI
          wfP_trancl)

have D:
  ‹(◁)** a x› ‹(◁)** x b›
  if ‹x ∈ X› for x
  using that A(3) Ca Cb
  subgoal
    by (metis rtranclp.rtrancl_refl
              tranclp_into_rtranclp)
  by (metis A(3) Cb(1) Cb(2)
      Nitpick.rtranclp_unfold that)
then have E:
```

```
    ‹order b ≤ order x›
    ‹order x ≤ order a›
    if ‹x ∈ X› for x
    using ‹x ∈ X› order_le1 by metis+
  define Q :: ‹nat set› where
    ‹Q = { n . order b ≤ n ∧ n ≤ order a }›
  have ‹finite Q› using Q_def by auto
  have F: ‹order ` X ⊆ Q›
    by (auto simp: Q_def intro!: E)
  have ‹finite (order ` X)›
    using ‹finite Q› finite_subset[OF F]
    by simp
  then show ‹finite X›
    using B finite_imageD by blast
qed

end

end
```

B.3 Quality Spaces and Qualia

```
theory QualitySpaces
  imports "../Misc/Common"
begin type_synonym 'q quality_space = ‹'q set›

locale quality_space_sig =
  fixes
    𝒬𝒮 :: ‹'q quality_space set› and
    Typ_q :: ‹'q itself›

locale quality_space =
  quality_space_sig +
  assumes
    no_empty_quality_space: ‹∅ ∉ 𝒬𝒮› and
    quality_spaces_are_disjoint:
      ‹⟦ Q_1 ∈ 𝒬𝒮 ; Q_2 ∈ 𝒬𝒮 ; Q_1 ∩ Q_2 ≠ ∅ ⟧ ⟹ Q_1 = Q_2›

context quality_space
begin end
```

```
context quality_space_sig
begin definition qualia (‹𝒬›)
    where ‹𝒬 ≡ ⋃ 𝒬𝒮›definition qspace :: ‹'q ⇒ 'q set› where
  ‹qspace q ≡
      if q ∈ 𝒬 then THE Q. Q ∈ 𝒬𝒮 ∧ q ∈ Q
                else ∅›

end

context quality_space_sig
begin

lemma 𝒬_I[intro]:
  assumes ‹Q ∈ 𝒬𝒮› ‹q ∈ Q›
  shows ‹q ∈ 𝒬›
  using assms qualia_def by auto

lemma  𝒬_E[elim]:
  assumes ‹q ∈ 𝒬›
  obtains Q where ‹Q ∈ 𝒬𝒮› ‹q ∈ Q›
  using assms qualia_def by auto

lemma  qspace_eq_I1:
  assumes ‹∃!Q. Q ∈ 𝒬𝒮 ∧ q ∈ Q›
          ‹q ∈ Q› ‹Q ∈ 𝒬𝒮›
  shows ‹qspace q = Q›
proof -
  have A: ‹q ∈ 𝒬› using assms(2,3) by blast
  show ‹?thesis›
    apply (simp add: qspace_def assms A)
    by (intro the1_equality conjI assms)
qed

lemma  qspace_E1:
  assumes ‹∃!Q. Q ∈ 𝒬𝒮 ∧ q ∈ Q› ‹q ∈ 𝒬›
  obtains ‹qspace q ∈ 𝒬𝒮› ‹q ∈ qspace q›
proof -
  have A: ‹(THE Q. Q ∈ 𝒬𝒮 ∧ q ∈ Q) = qspace q›
    using qspace_def assms by simp
  let ‹?P› = ‹λQ. Q ∈ 𝒬𝒮 ∧ q ∈ Q›
  show ‹?thesis›
    by (metis assms(1) qspace_eq_I1 that)
```

```
qed

end

context  quality_space
begin

lemmas just_quality_space_axioms =
    no_empty_quality_space
    quality_spaces_are_disjoint

lemmas all_quality_space_axioms =
    just_quality_space_axioms

lemma qspace_ex1:
  assumes ‹q ∈ 𝒬›
  shows ‹∃!Q. Q ∈ 𝒬𝒮 ∧ q ∈ Q›
proof -
  obtain Q where A: ‹Q ∈ 𝒬𝒮› ‹q ∈ Q›
    using assms by blast
  show ‹?thesis›
  proof (intro ex1I[of _ ‹Q›] conjI A)
    fix Q'
    assume as: ‹Q' ∈ 𝒬𝒮 ∧ q ∈ Q'›
    then obtain ‹Q' ∈ 𝒬𝒮› ‹Q ∩ Q' ≠ ∅›
      using A by blast
    then show ‹Q' = Q›
      using quality_spaces_are_disjoint A
      by metis
  qed
qed

lemma qspace_eq_I[intro!]:
 assumes ‹q ∈ Q› ‹Q ∈ 𝒬𝒮›
 shows ‹qspace q = Q›
  using assms qspace_eq_I1[OF qspace_ex1]
  by blast

lemma qspace_E[elim]:
 assumes ‹q ∈ 𝒬›
 obtains ‹qspace q ∈ 𝒬𝒮› ‹q ∈ qspace q›
  using assms qspace_E1[OF qspace_ex1]
```

```
  by blast

lemma qsOf_not_in_Q_empty_iff[iff]:
    ‹qspace q = ∅ ⟷ q ∉ 𝒬›
  apply (cases ‹q ∈ 𝒬›)
  subgoal by blast
  by (simp add: qspace_def)

lemma non_empty_quality_space_1:
  assumes ‹Q ∈ 𝒬𝒮›
  obtains q where ‹q ∈ Q›
  using no_empty_quality_space assms
  by fastforce

end

end
```

B.4 Qualified Particulars

```
theory QualifiedParticulars
  imports  Inherence QualitySpaces
begin locale qualified_particulars_sig =
  inherence_sig where Typ_p = ‹Typ_p› +
  quality_space_sig where Typ_q = ‹Typ_q›
  for Typ_p :: ‹'p itself› and
      Typ_q :: ‹'q itself› +
  fixes
    assoc_quale :: ‹'p ⇒ 'q ⇒ bool› (infix ‹⇝› 75)
begin definition qualifiedParticulars (‹𝒫_q›) where
  ‹𝒫_q ≡ { x | x q . x ⇝ q }›definition quale_of (‹qOf›) where ‹qOf x ≡ THE q. x
⇝ q›definition qsOf where
  ‹qsOf x ≡
      if x ∈ 𝒫_q
      then THE Q. Q ∈ 𝒬𝒮 ∧ (∀q. x ⇝ q ⟶ q ∈ Q)
      else ∅›

end

locale qualified_particulars =
  inherence where Typ_p = ‹Typ_p› +
```

```
  quality_space where Typ_q = ‹Typ_q› +
  qualified_particulars_sig where Typ_p = ‹Typ_p› and Typ_q = ‹Typ_q›
  for Typ_p :: ‹'p itself› and
      Typ_q :: ‹'q itself› +
  assumes
    assoc_quale_scope:
      ‹x ⇝ q ⟹ x ∈ 𝓜 ∧ q ∈ 𝓠› and
    assoc_quale_unique:
      ‹⟦ x ⇝ q1 ; x ⇝ q2 ⟧ ⟹ q1 = q2› and
    quality_moment_unique_by_quality_space:
        ‹⟦ w ∈ 𝓦 ; y1 ∈ w ; y2 ∈ w ;
           y1 ◁ x ; y2 ◁ x ;
           y1 ⇝ q1 ; y2 ⇝ q2 ; q1 ∈ Q ; q2 ∈ Q ; Q ∈ 𝓠𝓢 ⟧ ⟹ y1 = y2› and
    every_quality_space_is_used:
      ‹Q ∈ 𝓠𝓢 ⟹ ∃x. ∃q ∈ Q.  x ⇝ q› and
    quale_determines_moment:
      ‹⟦   y1 ◁ x ; y2 ◁ x ; y1 ⇝ q ; y2 ⇝ q ⟧ ⟹ y1 = y2›
begin end

context qualified_particulars_sig
begin

lemma qualifiedParticularsI[intro]:
  assumes ‹x ⇝ q›
  shows ‹x ∈ 𝒫_q›
  using assms
  by (auto simp: qualifiedParticulars_def)

lemmaqualifiedParticularsE[elim]:
  assumes ‹x ∈ 𝒫_q›
  obtains q where ‹x ⇝ q›
  using assms
  by (auto simp: qualifiedParticulars_def)

lemma qualifiedParticulars_iff: ‹x ∈ 𝒫_q ⟷ (∃q. x ⇝ q)›
  by blast

lemma qOf_eq_I1:
  assumes ‹x ⇝ q› ‹∃!q. x ⇝ q›
  shows ‹qOf x = q›
  by (simp add: quale_of_def ; intro the1_equality assms)
```

```
lemma qOf_I1:
  assumes ‹∃!q. x ⇝ q›
  shows ‹x ⇝ qOf x›
  by (simp add: quale_of_def ; rule the1I2 ; simp add: assms)

lemma qsOf_eq_I1:
  assumes ‹∃!Q. Q ∈ 𝒬𝒮 ∧ (∀q. x ⇝ q ⟶ q ∈ Q)› ‹x ∈ 𝒫q› ‹Q ∈ 𝒬𝒮› ‹⋀q. x ⇝
q ⟹ q ∈ Q›
  shows ‹qsOf x = Q›
proof -
  show ‹?thesis›
    apply (simp only: qsOf_def assms(2) if_True)
    apply (intro the1_equality assms(1,2,3) conjI allI impI)
    by (frule assms(4) ; simp)
qed

lemma qsOf_I1:
  assumes ‹∃!Q. Q ∈ 𝒬𝒮 ∧ (∀q. x ⇝ q ⟶ q ∈ Q)› ‹x ∈ 𝒫q›
  obtains ‹qsOf x ∈ 𝒬𝒮› ‹⋀q w. x ⇝ q ⟹ q ∈ qsOf x›
proof -
  have R1: ‹qsOf x ∈ 𝒬𝒮›
    apply (simp only: qsOf_def assms(2) if_True)
    by (rule the1I2 ; (intro assms(1))? ; simp)
  have R2: ‹q ∈ qsOf x› if as: ‹x ⇝ q› for q
    apply (simp only: qsOf_def assms(2) if_True)
    apply (rule the1I2 ; (intro assms(1))? ; clarsimp)
    using as by blast
  show ‹?thesis›
    using R1 R2 that by metis
qed

lemma qsOf_non_qual_particular[simp]:
  assumes ‹x ∉ 𝒫q›
  shows ‹qsOf x = ∅›
  using assms by (auto simp: qsOf_def)

end

context qualified_particulars
begin

lemmas just_qualified_particulars_axioms =
```

```
    assoc_quale_scope
    assoc_quale_unique

lemmas all_qualified_particulars_axioms =
    all_possible_worlds_axioms
    all_quality_space_axioms
    just_qualified_particulars_axioms

lemma assoc_quale_scopeD:
  assumes ‹x ⇝ q›
  shows ‹x ∈ ℰ› ‹q ∈ 𝒬› ‹x ∈ ℳ›
  using assms assoc_quale_scope by auto

lemma assoc_quale_exI1[intro!]: ‹m ∈ 𝒫q ⟹ ∃!q. m ⇝ q›
  using assoc_quale_unique
  by blast

lemma qOf_eq_I[intro!]:
  assumes ‹x ⇝ q›
  shows ‹qOf x = q›
  apply (intro qOf_eq_I1 assms)
  using assoc_quale_exI1 assoc_quale_scope assms by blast


lemma qOf_assoc_quale_I[intro!]: ‹x ∈ 𝒫q ⟹ x ⇝ qOf x›
  using qOf_I1 assoc_quale_exI1
  by blast

lemma qsOf_ex1[intro!]:
  assumes ‹x ∈ 𝒫q›
  shows ‹∃!Q. Q ∈ 𝒬𝒮 ∧ (∀q. x ⇝ q ⟶ q ∈ Q)›
proof -
  obtain q where A: ‹x ⇝ q› using assms by blast
  then obtain B: ‹q ∈ 𝒬› using assoc_quale_scope by blast
  then obtain Q where C: ‹Q ∈ 𝒬𝒮› ‹q ∈ Q› by blast
  then have D: ‹Q = Q'› if as: ‹x ⇝ q'› ‹q' ∈ Q'› ‹Q' ∈ 𝒬𝒮› for q' Q'
    using assoc_quale_unique A as
    by (metis qspace_eq_I)
  have E: ‹∀q. x ⇝ q ⟶ q ∈ Q›
    by (metis D 𝒬_E assoc_quale_scope)
  show ‹?thesis›
    apply (intro ex1I[of _ ‹Q›] conjI E C(1) ; elim conjE)
```

```
    using A D by metis
qed

lemma qsOf_I[intro!]:
  assumes ‹x ∈ 𝒫q›
  shows ‹qsOf x ∈ 𝒬𝒮›
  using assms qsOf_I1[OF qsOf_ex1,of ‹x›] by metis

lemma qsOf_memb[dest]:
  assumes ‹x ∈ 𝒫q› ‹x ⇝ q›
  shows ‹q ∈ qsOf x›
  using assms qsOf_I1[OF qsOf_ex1,of ‹x›]
  by auto

lemma qspace_qOf_eq_qsOf[simp]:
  assumes ‹x ∈ 𝒫q› ‹w ∈ 𝒲› ‹x ∈ w›
  shows ‹qspace (qOf x) = qsOf x›
  using qsOf_memb assoc_quale_scope assms by blast

lemma qsOf_eq_I:
  assumes ‹x ⇝ q› ‹q ∈ Q› ‹Q ∈ 𝒬𝒮›
  shows ‹qsOf x = Q›
  using assms qsOf_I qsOf_memb qualifiedParticularsI
  by (metis qspace_eq_I)

lemma qualified_particulars_are_moments: ‹𝒫q ⊆ ℳ›
  using assoc_quale_scopeD(3) by blast

end

end
```

B.5 Towardness Relation

```
theory Towardness
  imports   Inherence
begin
```

B.5.1 Towardness theory

```
locale towardness_sig =
    inherence_sig where Typp = ‹Typp›
```

```
  for
    towards :: ‹'p ⇒ 'p ⇒ bool› (infix ‹⟶› 75) and
    Typ_p :: ‹'p itself›
begin end

locale towardness =
    towardness_sig where Typ_p = ‹Typ_p› +
    inherence where Typ_p = ‹Typ_p›
  for
    Typ_p :: ‹'p itself› +
  assumes
   towardness_scope: ‹x ⟶ y ⟹ x ∈ 𝓜 ∧ y ∈ 𝓢› and
   towardness_imp_ed[dest]: ‹x ⟶ y ⟹ ed x y› and
   towardness_diff_ultimate_bearers[dest]: ‹x ⟶ y ⟹ !β x ≠ y› and
   towardness_single: ‹⟦ x ⟶ y₁ ; x ⟶ y₂ ⟧ ⟹ y₁ = y₂›
beginend

context towardness_sig
begin definition directed_moments (‹𝓜→›) where
  ‹𝓜→ ≡ { m . ∃x. m ⟶ x }›

lemma directed_moments_I[intro]: ‹m ⟶ x ⟹ m ∈ 𝓜→›
  by (auto simp: directed_moments_def)

lemma directed_moments_E[elim]:
  assumes ‹m ∈ 𝓜→›
  obtains x where ‹m ⟶ x›
  using assms
  by (auto simp: directed_moments_def)

end

context towardness
begin

lemma towardness_unique[intro!]: ‹Uniq ((⟶) x)›
  using towardness_single
  by (auto simp: Uniq_def)

lemma towardness_scopeE[elim]:
  assumes ‹x ⟶ y›
  obtains ‹x ∈ 𝓜› ‹y ∈ 𝓟› ‹x ∈ 𝓟› ‹y ∈ 𝓢›
```

```
  using assms towardness_scope by auto

lemma towardness_scopeD:
  assumes ‹x ⟶ y›
  shows ‹x ∈ 𝓜› ‹x ∈ 𝓟› ‹y ∈ 𝓟›  ‹y ∈ 𝓢›
  using assms by auto

lemma towardness_apply_to_moments: ‹x ⟶ y ⟹ x ∈ 𝓜›
  using towardness_scope by simp

lemmas all_towardness_axioms =
    all_inherence_axioms
    towardness_apply_to_moments
    towardness_imp_ed

lemma directed_moments_are_moments: ‹𝓜→ ⊆ 𝓜›
  using towardness_scope directed_moments_E
  by blast

end

end
```

B.6 Theory of Particulars

```
theory Particulars
  imports Inherence QualifiedParticulars Towardness
begin

locale ufo_particular_theory_sig =
    inherence_sig where  Typ_p = ‹Typ_p› +
    qualified_particulars_sig where Typ_p = ‹Typ_p› and Typ_q = ‹Typ_q› +
    towardness_sig where Typ_p = ‹Typ_p›
  for
    Typ_p :: ‹'p itself› and
    Typ_q :: ‹'q itself›

locale ufo_particular_theory =
    ufo_particular_theory_sig where  Typ_p = ‹Typ_p› and Typ_q = ‹Typ_q› +
    inherence_sig where Typ_p = ‹Typ_p› +
    qualified_particulars where Typ_p = ‹Typ_p› and Typ_q = ‹Typ_q› +
```

```
    towardness where Typ_p = ‹Typ_p›
  for
    Typ_p :: ‹'p itself› and
    Typ_q :: ‹'q itself›  +
  assumes
    qualified_particulars_are_not_bearers: ‹x ∈ 𝒫_q ⟹ ¬ y ◁ x›
begin lemmas just_ufo_particular_theory_axioms =
  qualified_particulars_are_not_bearers

lemmas all_ufo_particular_theory_axioms =
      just_ufo_particular_theory_axioms
      all_inherence_axioms
      all_qualified_particulars_axioms
      all_towardness_axioms


end


end
```

B.7 Universals and Instantiation

```
theory Universals
  imports "../Particulars/Particulars"
begin locale instantiation_sig =
    ufo_particular_theory_sig where Typ_p = Typ_p and Typ_q = Typ_q
  for
    iof :: ‹'p ⇒ 'p set ⇒ 'u ⇒ bool› (‹_ ::_ _› [74,1,74] 75) and
    Typ_p :: ‹'p itself› and
    Typ_q :: ‹'q itself› and
    Typ_u :: ‹'u itself›
begin definition ‹𝒰 ≡ { u | x w u . x ::_w u}›definition subsumes (infix ‹⊑› 75) where
  ‹u_1 ⊑ u_2 ⟷ u_1 ∈ 𝒰 ∧ (∀x w. x ::_w u_1 ⟶ x ::_w u_2)›definition char_by where
  ‹char_by U u ⟷ U ∈ 𝒰 ∧ (∀x w. x ::_w U ⟶ (∃y. y ◁ x ∧ y ::_w u))›definition Insts
:: ‹'u ⇒ 'p set› where
  ‹Insts U ≡ { x | x w . x ::_w U }›definition ‹𝒰_𝒮 ≡ { u | x w u . x ::_w u ∧ x ∈ 𝒮}›definitio
‹𝒰_ℳ ≡ { u | x w u . x ::_w u ∧ x ∈ ℳ}›definition rigid :: ‹'u ⇒ bool› where
  ‹rigid u ≡ u ∈ 𝒰 ∧ (∀x w_1 w_2. x ::_w_1 u ∧ w_2 ∈ 𝒲 ∧ x ∈ w_2 ⟶ x ::_w_2 u)›

end
```

```
locale instantiation_thy =
    instantiation_sig where Typ_p = Typ_p and Typ_q = Typ_q and Typ_u = Typ_u +
    ufo_particular_theory where Typ_p = Typ_p and Typ_q = Typ_q
  for
    Typ_p :: ‹'p itself› and
    Typ_q :: ‹'q itself› and
    Typ_u :: ‹'u itself› +
  assumes
    iof_imp_existence: ‹x ::_W U ⟹ w ∈ 𝒲 ∧ x ∈ w›
begin end

locale universals =
    instantiation_thy where Typ_p = Typ_p and Typ_q = Typ_q and Typ_u = Typ_u
  for
    Typ_p :: ‹'p itself› and
    Typ_q :: ‹'q itself› and
    Typ_u :: ‹'u itself› +
  assumes
    substantial_moment_univs_separate: ‹𝒰_𝒮 ∩ 𝒰_ℳ = ∅› and
    moment_universals_are_rigid: ‹u ∈ 𝒰_ℳ ⟹ rigid u›
begin end

context instantiation_sig
begin

lemma 𝒰_I[intro]: ‹x ::_W u ⟹ u ∈ 𝒰›
  by (auto simp: 𝒰_def)

lemma𝒰_E[elim]:
  assumes ‹u ∈ 𝒰›
  obtains x w where ‹x ::_W u›
  using assms by (auto simp: 𝒰_def)

lemma 𝒰_iff: ‹u ∈ 𝒰 ⟷ (∃x w. x ::_W u)›
  by (auto simp: 𝒰_def)

lemma subsumesI[intro!]:
  assumes ‹u_1 ∈ 𝒰› ‹⋀x w. x ::_W u_1 ⟹ x ::_W u_2›
  shows ‹u_1 ⊑ u_2›
  using assms
  by (auto simp: subsumes_def)
```

lemma *subsumesE[elim]:*
assumes ‹$u_1 \sqsubseteq u_2$›
obtains ‹$u_1 \in \mathcal{U}$› ‹$u_2 \in \mathcal{U}$› ‹$\bigwedge x\ w.\ x ::_w u_1 \Longrightarrow x ::_w u_2$›
using *assms*
apply *(auto simp: subsumes_def)*
using *$\mathcal{U}$_I $\mathcal{U}$_E* **by** *metis*

lemma *subsumes_refl[intro!]:*
assumes ‹$u \in \mathcal{U}$›
shows ‹$u \sqsubseteq u$›
using *assms* **by** *blast*

lemma *subsumes_trans[trans]:*
assumes ‹$u_1 \sqsubseteq u_2$› ‹$u_2 \sqsubseteq u_1$›
shows ‹$u_2 \sqsubseteq u_1$›
using *assms* **by** *blast*

lemma *subsumes_D:*
assumes ‹$u_1 \sqsubseteq u_2$›
shows ‹$u_1 \in \mathcal{U}$› ‹$u_2 \in \mathcal{U}$› ‹$\bigwedge x\ w.\ x ::_w u_1 \Longrightarrow x ::_w u_2$›
using *assms* **by** *blast+*

lemma *char_by_I[intro!]:*
assumes ‹$U \in \mathcal{U}$› ‹$\bigwedge x\ w.\ x ::_w U \Longrightarrow \exists y.\ y \lhd x \wedge y ::_w u$›
shows ‹*char_by U u*›
using *assms* **by** *(auto simp: char_by_def)*

lemma *char_by_E[elim]:*
assumes ‹*char_by U u*›
obtains ‹$U \in \mathcal{U}$› ‹$u \in \mathcal{U}$› ‹$\bigwedge x\ w.\ x ::_w U \Longrightarrow \exists y.\ y \lhd x \wedge y ::_w u$›
using *assms*
apply *(auto simp: char_by_def)*
by *blast*

lemma *char_by_D[elim]:*
assumes ‹*char_by U u*›
shows ‹$U \in \mathcal{U}$› ‹$u \in \mathcal{U}$› ‹$\bigwedge x\ w.\ x ::_w U \Longrightarrow \exists y.\ y \lhd x \wedge y ::_w u$›
using *assms* **by** *auto*

lemma *char_by_mono:*
assumes ‹$U_2 \sqsubseteq U_1$› ‹$u_1 \sqsubseteq u_2$› ‹*char_by* U_1 u_1›

shows ‹*char_by* U_2 u_2›
using *assms*
by *(meson char_by_E subsumesE char_by_I)*

lemma *InstsI[intro]:*
assumes ‹x $::_w$ U›
shows ‹x $\in$ *Insts* U›
using *assms* **by** *(auto simp: Insts_def)*

lemma *InstsE[elim!]:*
assumes ‹x $\in$ *Insts* U›
obtains w **where** ‹x $::_w$ U›
using *assms* **by** *(auto simp: Insts_def)*

lemma *Insts_iff:* ‹x $\in$ *Insts* U $\longleftrightarrow$ $(\exists w.\ x ::_w U)$›
by *blast*

lemma *Insts_mono[intro]:*
assumes ‹u_1 $\sqsubseteq$ u_2›
shows ‹*Insts* u_1 $\subseteq$ *Insts* u_2›
using *assms* **by** *fastforce*

lemma *rigidI[intro!]:*
assumes ‹u $\in$ $\mathcal{U}$› ‹$\bigwedge x\ w_1\ w_2.\ [\![\ x ::_{w_1} u\ ;\ w_2 \in \mathcal{W}\ ;\ x \in w_2\ ]\!] \Longrightarrow x ::_{w_2} u$›
shows ‹*rigid* u›
using *assms* **by** *(auto simp: rigid_def)*

lemma *rigidE[elim!]:*
assumes ‹*rigid* u›
obtains ‹u $\in$ $\mathcal{U}$› ‹$\bigwedge x\ w_1\ w_2.\ [\![\ x ::_{w_1} u\ ;\ w_2 \in \mathcal{W}\ ;\ x \in w_2\ ]\!] \Longrightarrow x ::_{w_2} u$›
using *assms* **by** *(auto simp: rigid_def)*

lemma $\mathcal{U}_{\mathcal{M}}$*_I[intro]:*
assumes ‹x $::_w$ u› ‹x $\in$ $\mathcal{M}$›
shows ‹u $\in$ $\mathcal{U}_{\mathcal{M}}$›
using *assms* **by** *(auto simp:* $\mathcal{U}_{\mathcal{M}}$*_def)*

lemma $\mathcal{U}_{\mathcal{M}}$*_E[elim]:*
assumes ‹u $\in$ $\mathcal{U}_{\mathcal{M}}$›
obtains x w **where** ‹x $::_w$ u› ‹x $\in$ $\mathcal{M}$›
using *assms* **by** *(auto simp:* $\mathcal{U}_{\mathcal{M}}$*_def)*

lemma $\mathcal{U}_{\mathcal{M}}$_$\mathcal{U}$: ‹$u \in \mathcal{U}_{\mathcal{M}} \Longrightarrow u \in \mathcal{U}$›
  by *auto*

lemma $\mathcal{U}_{\mathcal{M}}$_subset_$\mathcal{U}$: ‹$\mathcal{U}_{\mathcal{M}} \subseteq \mathcal{U}$›
  by *auto*

lemma $\mathcal{U}_{\mathcal{M}}$_iff: ‹$u \in \mathcal{U}_{\mathcal{M}} \longleftrightarrow (\exists x\ w.\ x ::_w u \wedge x \in \mathcal{M})$›
  by *(auto simp:* $\mathcal{U}_{\mathcal{M}}$*_def)*

lemma $\mathcal{U}_{\mathcal{S}}$*_I[intro]:*
  assumes ‹$x ::_w u$› ‹$x \in \mathcal{S}$›
  shows ‹$u \in \mathcal{U}_{\mathcal{S}}$›
  using *assms* **by** *(auto simp:* $\mathcal{U}_{\mathcal{S}}$*_def)*

lemma $\mathcal{U}_{\mathcal{S}}$*_E[elim]:*
  assumes ‹$u \in \mathcal{U}_{\mathcal{S}}$›
  obtains x w **where** ‹$x ::_w u$› ‹$x \in \mathcal{S}$›
  using *assms* **by** *(auto simp:* $\mathcal{U}_{\mathcal{S}}$*_def)*

lemma $\mathcal{U}_{\mathcal{S}}$_$\mathcal{U}$: ‹$u \in \mathcal{U}_{\mathcal{S}} \Longrightarrow u \in \mathcal{U}$›
  by *auto*

lemma $\mathcal{U}_{\mathcal{S}}$_subset_$\mathcal{U}$: ‹$\mathcal{U}_{\mathcal{S}} \subseteq \mathcal{U}$›
  by *auto*

lemma $\mathcal{U}_{\mathcal{S}}$_iff: ‹$u \in \mathcal{U}_{\mathcal{S}} \longleftrightarrow (\exists x\ w.\ x ::_w u \wedge x \in \mathcal{S})$›
  by *(auto simp:* $\mathcal{U}_{\mathcal{S}}$*_def)*

end

context *instantiation_thy*
begin

lemma *iof_scope_E:*
  assumes ‹$x ::_w U$›
  obtains ‹$x \in \mathcal{P}$› ‹$w \in \mathcal{W}$› ‹$x \in w$› ‹$U \in \mathcal{U}$›
  using *assms iof_imp_existence*
    worlds_are_made_of_particulars **by** *blast*

lemma *iof_scope:*

```
  assumes ‹x ::w U›
  shows ‹x ∈ 𝒫› ‹w ∈ 𝒲› ‹x ∈ w› ‹U ∈ 𝒰›
  using assms iof_scope_E by metis+


end

context universals
begin

lemma 𝒰𝒮_insts[intro]:
  assumes ‹U ∈ 𝒰𝒮› ‹x ::w U›
  shows ‹x ∈ 𝒮›
proof -
  have A: ‹x ∈ ℰ› using assms(2)
    using iof_scope(1) by auto
  have B: ‹x ∉ ℳ› using 𝒰ℳ_I substantial_moment_univs_separate
    assms substantial_moment_univs_separate by blast
  then show ?thesis using A by blast
qed

lemma 𝒰ℳ_insts[intro]:
  assumes ‹U ∈ 𝒰ℳ› ‹x ::w U›
  shows ‹x ∈ ℳ›
proof -
  have A: ‹x ∈ ℰ› using assms(2)
    using iof_scope(1) by auto
  have B: ‹x ∉ 𝒮› using 𝒰𝒮_I substantial_moment_univs_separate
    assms substantial_moment_univs_separate by blast
  then show ?thesis using A by blast
qed

lemma endurant_universal_subsumption_cases:
  assumes ‹U1 ⊑ U2›
  obtains (substantial_universals) ‹U1 ∈ 𝒰𝒮› ‹U2 ∈ 𝒰𝒮› |
          (moment_universals) ‹U1 ∈ 𝒰ℳ› ‹U2 ∈ 𝒰ℳ›
proof -
  obtain A: ‹U1 ∈ 𝒰› ‹U2 ∈ 𝒰› ‹⋀x w. x ::w U1 ⟹ x ::w U2›
    using assms subsumesE by blast
  obtain x w where B: ‹x ::w U1› ‹x ::w U2› using A(1,3) by blast
  then have ‹x ∈ ℰ› using iof_scope by metis
  then show ?thesis
```

```
    apply (cases x rule: endurant_cases)
    subgoal using B substantial_universals by blast
    using B moment_universals by blast
qed

lemma substantial_universal_subsumes[iff]:
  assumes ‹U1 ⊑ U2›
  shows ‹U1 ∈ 𝒰𝒮 ⟷ U2 ∈ 𝒰𝒮› (is ‹?A›) ‹U1 ∈ 𝒰ℳ ⟷ U2 ∈ 𝒰ℳ› (is ‹?B›)
proof -
  show ?A
    using assms
    apply (cases rule: endurant_universal_subsumption_cases ; simp?)
    using substantial_moment_univs_separate by blast
  show ?B
    using assms
    apply (cases rule: endurant_universal_subsumption_cases ; simp?)
    using substantial_moment_univs_separate by blast
qed

end

end
```

APPENDIX C – UFO Particular Structures and Morphisms

C.1 Particular Structures

```
theory ParticularStructure
  imports "../Particulars/Particulars"
beginrecord ('p,'q) particular_struct =
  ps_quality_spaces :: ‹'q set set›
  ps_worlds :: ‹'p set set›
  ps_inheres_in :: ‹'p ⇒ 'p ⇒ bool›
  ps_assoc_quale :: ‹'p ⇒ 'q ⇒ bool›
  ps_towards :: ‹'p ⇒ 'p ⇒ bool›lemma particular_struct_eqI[intro!]:
  fixes M1 M2 :: ‹('p,'q) particular_struct›
  assumes
    ‹ps_quality_spaces M1 = ps_quality_spaces M2›
    ‹ps_worlds M1 = ps_worlds M2›
    ‹ps_inheres_in M1 = ps_inheres_in M2›
    ‹ps_assoc_quale M1 = ps_assoc_quale M2›
    ‹ps_towards M1 = ps_towards M2›
  shows ‹M1 = M2›
  using assms by auto

locale particular_struct_sig =
  fixes
    Γ :: ‹('p,'q) particular_struct› and
    Typp :: ‹'p itself› and
    Typq :: ‹'q itself›

context particular_struct_sig
begin

abbreviation ‹𝒲 ≡ ps_worlds Γ›

abbreviation inheresIn (infix ‹◁› 75) where
  ‹inheresIn ≡ ps_inheres_in Γ›

abbreviation assoc_quale (infix ‹⇝› 75) where
  ‹assoc_quale ≡ ps_assoc_quale Γ›
```

```
abbreviation towards (infix ‹⟶› 75) where
  ‹(⟶) ≡ ps_towards Γ›

abbreviation ‹𝒬𝒮 ≡ ps_quality_spaces Γ›

end

definition (in ufo_particular_theory_sig) Γ
  :: ‹('p,'q) particular_struct› where
  ‹Γ ≡ ⦇
      ps_quality_spaces = 𝒬𝒮,
      ps_worlds = 𝒲,
      ps_inheres_in = inheresIn,
      ps_assoc_quale = assoc_quale,
      ps_towards = towards
    ⦈›

lemma (in ufo_particular_theory_sig)
    Γ_simps[simp]:
  ‹ps_quality_spaces Γ = 𝒬𝒮›
  ‹ps_worlds Γ = 𝒲›
  ‹ps_inheres_in Γ = inheresIn›
  ‹ps_assoc_quale Γ = assoc_quale›
  ‹ps_towards Γ = towards›
  using Γ_def by auto


locale particular_struct_defs =
    particular_struct_sig
      where Typ_p = ‹Typ_p› and Typ_q = ‹Typ_q› +
    ufo_particular_theory_sig where
      𝒬𝒮 = ‹𝒬𝒮› and
      𝒲 = ‹𝒲› and
      inheresIn = ‹inheresIn› and
      assoc_quale = ‹assoc_quale› and
      towards = ‹towards› and
      Typ_p = ‹Typ_p› and Typ_q = ‹Typ_q›
  for
    Typ_p :: ‹'p itself› and
    Typ_q :: ‹'q itself›
```

```
locale particular_struct =
    particular_struct_defs where
      Typ_p = ‹Typ_p› and Typ_q = ‹Typ_q›  +
    ufo_particular_theory where
      QS = ‹QS› and
      W = ‹W› and
      inheresIn = ‹inheresIn› and
      assoc_quale = ‹assoc_quale› and
      towards = ‹towards› and
      Typ_p = ‹Typ_p› and Typ_q = ‹Typ_q›
  for
    Typ_p :: ‹'p itself› and
    Typ_q :: ‹'q itself›

lemma ufo_theory_and_particular_struct_eq[simp]:
  fixes
    W QS assoc_quale
    universal_qualia_set inheresIn
    charSet towards
  shows
    ‹particular_struct (
        ufo_particular_theory_sig.Γ
          W inheresIn QS assoc_quale towards) ⟷
     ufo_particular_theory
          W inheresIn QS assoc_quale towards
     ›
    (is ‹?B ⟷ ?A›)
proof -
  note S = ufo_particular_theory_sig.Γ_simps
  show ‹?thesis›
  proof
    assume ‹?A›
    then interpret
        ufo_particular_theory ‹W› ‹inheresIn›
           ‹QS› ‹assoc_quale› ‹towards› by simp
    show ‹?B›
      apply (intro_locales ; simp only: S)
      subgoal
        by (simp add: possible_worlds_axioms)
      subgoal
        by (simp add: inherence_base.axioms(2)
                  inherence_base_axioms)
```

```
      subgoal
        by (simp add: noetherian_inherence_axioms_def)
      subgoal
        by (simp add: inherence_axioms_def)
      subgoal
        by (simp add: quality_space_axioms)
      subgoal
        by (simp add: qualified_particulars.axioms(3)
                qualified_particulars_axioms)
      subgoal
        using towardness_axioms towardness_def
        by blast
      using just_ufo_particular_theory_axioms
            ufo_particular_theory_axioms_def
      by blast
  next
    assume ‹?B›
    then interpret particular_struct
      ‹ufo_particular_theory_sig.Γ
        𝒲 inheresIn 𝒬𝒮 assoc_quale towards›
      by simp
    show ‹?A›
      using ufo_particular_theory_axioms
      by (simp only: S)
  qed
qed

context ufo_particular_theory
begin

lemma particular_struct_valid[intro!,simp]: ‹particular_struct Γ›
  using ufo_theory_and_particular_struct_eq
        ufo_particular_theory_axioms
  by simp

end

abbreviation ‹particulars Γ ≡
  possible_worlds_sig.𝒫 (ps_worlds Γ)›

end
```

C.2 Particular Structures Homomorphisms

C.2.1 Definitions

```
theory ParticularStructureMorphisms
 imports ParticularStructure "../Misc/WellfoundedExtra"
begin

no_notation converse (‹(_⁻¹)› [1000] 999)

locale ufo_src_tgt_particular_models_sig =
    src: particular_struct_defs
    where Γ = ‹Γ₁› and Typ_p = ‹Typ_p1›
      and Typ_q = ‹Typ_q›  +
    tgt: particular_struct_defs
    where Γ = ‹Γ₂› and Typ_p = ‹Typ_p2›
      and Typ_q = ‹Typ_q›
  for
    Γ₁ :: ‹('p₁,'q) particular_struct› and
    Γ₂ :: ‹('p₂,'q) particular_struct› and
    Typ_p1 :: ‹'p₁ itself› and
    Typ_p2 :: ‹'p₂ itself› and
    Typ_q :: ‹'q itself›context ufo_src_tgt_particular_models_sig
begin abbreviation src_inheres_in (infix ‹◁ₛ› 75) where
  ‹x ◁ₛ y ≡ src.inheresIn x y›

abbreviation tgt_inheres_in (infix ‹◁ₜ› 75) where
  ‹x ◁ₜ y ≡ tgt.inheresIn x y›abbreviation src_assoc_quale (infix ‹⇝ₛ› 75) where
  ‹x ⇝ₛ y ≡ src.assoc_quale x y›

abbreviation tgt_assoc_quale (infix ‹⇝ₜ› 75) where
  ‹x ⇝ₜ y ≡ tgt.assoc_quale x y›abbreviation src_towards (infix ‹⟶ₛ› 75) where
  ‹x ⟶ₛ y ≡ src.towards x  y›

abbreviation tgt_towards (infix ‹⟶ₜ› 75) where
  ‹x ⟶ₜ y ≡ tgt.towards x  y›abbreviation ‹𝒬ₛ ≡ src.qualia›

abbreviation ‹𝒬ₜ ≡ tgt.qualia›end

locale particular_struct_morphism_sig =
  ufo_src_tgt_particular_models_sig
  where Typ_p1 = ‹Typ_p1› and Typ_p2 = ‹Typ_p2›
```

```
    and Typ_q = ‹Typ_q›
  for
    φ :: ‹'p_1 ⇒ 'p_2› and
    Typ_p1 :: ‹'p_1 itself› and
    Typ_p2 :: ‹'p_2 itself› and
    Typ_q :: ‹'q itself›

context particular_struct_morphism_sig
begin definition morph_image (‹𝒫_img›) where
  ‹𝒫_img ≡ φ ` src.𝒫›definition inv_morph (‹φ^-1›) where
  ‹φ^-1 = inv_into src.𝒫 φ›

lemma morph_image_I[intro]:
  ‹x ∈ src.𝒫 ⟹ φ x ∈ 𝒫_img›
  by (auto simp: morph_image_def)

lemma morph_image_E[elim!]:
  assumes ‹x ∈ 𝒫_img›
  obtains y where ‹y ∈ src.𝒫› ‹x = φ y›
  using assms by (auto simp: morph_image_def)

lemma morph_image_iff[simp]:
  ‹x ∈ 𝒫_img ⟷ (∃y ∈ src.𝒫. x = φ y)›
  by (auto simp: morph_image_def)

end
```

C.2.2 Axiomatization

```
locale pre_particular_struct_morphism =
    particular_struct_morphism_sig
    where Typ_p1 = ‹Typ_p1› and Typ_p2 = ‹Typ_p2›
      and Typ_q = ‹Typ_q› +
    src: particular_struct
    where Γ = ‹Γ_1› and Typ_p = ‹Typ_p1›
        and Typ_q = ‹Typ_q›  +
    tgt: particular_struct
    where Γ = ‹Γ_2› and Typ_p = ‹Typ_p2›
        and Typ_q = ‹Typ_q›
  for
    Typ_p1 :: ‹'p_1 itself› and
    Typ_p2 :: ‹'p_2 itself› and
```

```
    Typ_q :: ‹'q itself› +
  assumes
    quality_space_subset:
      ‹src.𝒬𝒮 ⊆ tgt.𝒬𝒮› and
    morph_preserves_particulars[intro]:
      ‹⋀x. x ∈ src.𝒫 ⟹ φ x ∈ tgt.𝒫› and
    morph_reflects_inherence[simp]:
      ‹⋀x y. ⟦ x ∈ src.𝒫 ; y ∈ src.𝒫 ⟧ ⟹
              φ x ◁t φ y ⟷ x ◁s y› and
    morph_does_not_add_bearers:
      ‹⋀x z. ⟦ x ∈ src.𝒫 ; φ x ◁t z ⟧ ⟹
        ∃y ∈ src.𝒫. z = φ y› and
    morph_reflects_towardness[simp]:
      ‹⋀x y. ⟦ x ∈ src.𝒫 ; y ∈ src.𝒫 ⟧ ⟹
          φ x ⟶t φ y ⟷ x ⟶s y› and
    morph_does_not_add_towards:
      ‹⋀x z. ⟦ x ∈ src.𝒫 ; φ x ⟶t z ⟧ ⟹
          ∃y ∈ src.𝒫. z = φ y› and
    morph_reflects_quale_assoc[simp]:
      ‹⋀x q. x ∈ src.𝒫 ⟹ x ⇝s q ⟷ φ x ⇝t q›

begin endcontext pre_particular_struct_morphism
begin

lemma morph_scope: ‹𝒫img ⊆ tgt.𝒫›
  using morph_preserves_particulars by blast

lemma morph_preserves_inherence_1[intro]:
  assumes ‹x ◁s y›
  shows ‹φ x ◁t φ y›
  using assms src.inherence_scope morph_reflects_inherence by auto

lemma morph_preserves_substantials[simp]:
  assumes ‹x ∈ src.𝒫›
  shows ‹φ x ∈ tgt.𝒮 ⟷ x ∈ src.𝒮›
proof -
  have ‹(∃y. φ x ◁t y) ⟷ (∃y. x ◁s y)›
    using assms morph_does_not_add_bearers morph_reflects_inherence by blast
  then show ‹?thesis›
    using assms by blast
qedlemma morph_preserves_moments[intro]: ‹x ∈ src.ℳ ⟹ φ x ∈ tgt.ℳ›
  using morph_reflects_inherence by blast
```

```
lemma morph_preserves_moments_simp[simp]:
  assumes ‹x ∈ src.𝒫›
  shows ‹φ x ∈ tgt.ℳ ⟷ x ∈ src.ℳ›
  using morph_preserves_moments assms
  using morph_preserves_substantials
  by blastlemma morph_reflects_bearers[simp]:
  assumes assms[simp,intro!]: ‹x ∈ src.ℳ›
  shows ‹tgt.bearer (φ x) = φ (src.bearer x)›
proof -
  have A[simp,intro!]: ‹φ x ∈ tgt.ℳ› using assms by blast
  then obtain y where B: ‹y ∈ src.𝒫› ‹φ x ◁t φ y› ‹x ◁s y›
    by (meson assms morph_preserves_inherence_1 src.bearer_ex1 src.endurantI1 src.endurantI2)
  then obtain ‹tgt.bearer (φ x)  = φ y› ‹src.bearer x = y›
    using tgt.bearer_eqI assms
    by (simp add: src.bearer_eqI)
  then show ‹?thesis› using B by blast
qedlemma morph_reflects_ultimate_bearers[simp]:
  assumes ‹x ∈ src.𝒫›
  shows ‹tgt.ultimateBearer (φ x) = φ (src.ultimateBearer x)›
  using assms
proof (induct ‹x› rule: wfP_induct[OF src.inherence_is_noetherian] ; simp)
  fix x
  assume as: ‹∀y. x ◁s y ⟶ y ∈ src.𝒫 ⟶ tgt.ultimateBearer (φ y) = φ (src.ultimateBearer
y)›
             ‹x ∈ src.𝒫›
  have A[simp]: ‹tgt.ultimateBearer (φ y) = φ (src.ultimateBearer y)› if ‹x ◁s y›
‹y ∈ src.𝒫› for y
    using that as(1) by blast
  show ‹tgt.ultimateBearer (φ x) = φ (src.ultimateBearer x)›
  proof (cases ‹x ∈ src.𝒮›)
    assume ‹x ∈ src.𝒮›
    then have B: ‹src.ultimateBearer x = x›
      using src.ultimate_bearer_eq_simp by auto
    have ‹φ x ∈ tgt.𝒮› using ‹x ∈ src.𝒮›
      by (simp add: as(2))
    then have ‹tgt.ultimateBearer (φ x) = φ x›
      using tgt.ultimate_bearer_eq_simp by auto
    then show ‹?thesis› using B by simp
  next
    assume as1: ‹x ∉ src.𝒮›
```

```
    then obtain B: ‹x ∈ src.𝓜› ‹φ x ∈ tgt.𝓜›
      using as(2) by blast
    then obtain y where C: ‹y ∈ src.𝓟› ‹x ◁s y› ‹φ x ◁t φ y›
      using morph_reflects_inherence morph_does_not_add_bearers src.inherence_scope
by auto
    then have D[simp]: ‹tgt.ultimateBearer (φ y) = φ (src.ultimateBearer y)›
      using A by blast
    have Esrc[simp]: ‹src.ultimateBearer x = src.ultimateBearer y›
      apply (rule src.ultimate_bearer_eqI1)
      using C(2) by blast
    have Etgt[simp]: ‹tgt.ultimateBearer (φ x) = tgt.ultimateBearer (φ y)›
      apply (rule tgt.ultimate_bearer_eqI1)
      using C(3) by blast
    show ‹?thesis› by simp
  qed
qedendcontext particular_struct_morphism_sig
begin

definition world_corresp (infix ‹⇔› 75) where
  ‹ws ⇔ wt ≡ ws ∈ src.𝓦 ∧ wt ∈ tgt.𝓦 ∧
              (∀x ∈ src.𝓟. (x ∈ ws ⟷ φ x ∈ wt))›

lemma world_corresp_I[intro!]:
  assumes
    ‹ws ∈ src.𝓦› ‹wt ∈ tgt.𝓦›
    ‹⋀x. x ∈ src.𝓟 ⟹ x ∈ ws ⟷ φ x ∈ wt›
  shows ‹ws ⇔ wt›
  using assms by (auto simp: world_corresp_def)

lemma world_corresp_E[elim!]:
  assumes ‹ws ⇔ wt›
  obtains ‹ws ∈ src.𝓦› ‹wt ∈ tgt.𝓦›
      ‹⋀x. x ∈ src.𝓟 ⟹ x ∈ ws ⟷ φ x ∈ wt›
  using assms by (auto simp: world_corresp_def)

end

locale particular_struct_morphism =
    pre_particular_struct_morphism
    where Typp1 = ‹Typp1› and Typp2 = ‹Typp2›
      and Typq = ‹Typq›
  for
```

```
      Typ_p1 :: ‹'p1 itself› and
      Typ_p2 :: ‹'p2 itself› and
      Typ_q :: ‹'q itself›  +
  assumes
    morph_worlds_correspond_src_tgt:
      ‹w_s ∈ src.𝒲 ⟹ ∃w_t. w_s ⇔ w_t› and
    morph_worlds_correspond_tgt_src:
      ‹w_t ∈ tgt.𝒲 ⟹ ∃w_s. w_s ⇔ w_t›
begin lemma morph_reflects_ed:
  ‹src.ed x y ⟷
      x ∈ src.𝒫 ∧ y ∈ src.𝒫
      ∧ tgt.ed (φ x) (φ y)›
proof (intro iffI conjI ; (elim conjE)?)
  assume ‹src.ed x y›
  then obtain A: ‹x ∈ src.𝒫› ‹y ∈ src.𝒫› ‹⋀w. ⟦ w ∈ src.𝒲 ; x ∈ w ⟧ ⟹ y ∈ w›
    by blast
  note A(1,2)[simp,intro!]
  show ‹x ∈ src.𝒫› ‹y ∈ src.𝒫› by auto
  obtain B[simp,intro!]: ‹φ x ∈ tgt.𝒫› ‹φ y ∈ tgt.𝒫› by auto
  show ‹tgt.ed (φ x) (φ y)›
  proof (intro tgt.edI ; simp?)
    fix w_t
    assume C[simp,intro!]: ‹w_t ∈ tgt.𝒲› ‹φ x ∈ w_t›
    obtain w_s where D: ‹w_s ⇔ w_t› using C morph_worlds_correspond_tgt_src by metis
    then obtain E[intro!,simp]: ‹w_s ∈ src.𝒲› ‹x ∈ w_s›
      using C(2) world_corresp_E by blast
    then have F[simp,intro!]: ‹y ∈ w_s› using A(3) by blast
    then show ‹φ y ∈ w_t› using D world_corresp_E by blast
  qed
next
  assume A[simp,intro!]: ‹x ∈ src.𝒫› ‹y ∈ src.𝒫› and B: ‹tgt.ed (φ x) (φ y)›
  then obtain A: ‹⋀w. ⟦ w ∈ tgt.𝒲 ; φ x ∈ w ⟧ ⟹ φ y ∈ w›
    by blast
  obtain B[simp,intro!]: ‹φ x ∈ tgt.𝒫› ‹φ y ∈ tgt.𝒫› by auto
  show ‹src.ed x y›
  proof (intro src.edI ; simp?)
    fix w_s
    assume C[simp,intro!]: ‹w_s ∈ src.𝒲› ‹x ∈ w_s›
    obtain w_t where D: ‹w_s ⇔ w_t› using C morph_worlds_correspond_src_tgt by metis
    then obtain E[intro!,simp]: ‹w_t ∈ tgt.𝒲› ‹φ x ∈ w_t›
      using C(2) world_corresp_E by blast
    then have F[simp,intro!]: ‹φ y ∈ w_t› using A by blast
```

```
    then show ‹y ∈ w_s› using D world_corresp_E by blast
  qed
qed

lemma morph_reflects_ed_simp[simp]:
  assumes ‹x ∈ src.𝒫› ‹y ∈ src.𝒫›
  shows ‹src.ed x y ⟷  tgt.ed (φ x) (φ y)›
  using assms morph_reflects_ed by blastlemma morph_reflects_src_indep: ‹src.indep
x y ⟷ x ∈ src.𝒫 ∧ y ∈ src.𝒫 ∧ tgt.indep (φ x) (φ y)›
  apply (auto simp add: src.indep_def tgt.indep_def)
  subgoal G1 for w_1 w_2
    by (meson morph_worlds_correspond_src_tgt world_corresp_def)
  subgoal G2 for w_1 w_2
    by (meson morph_worlds_correspond_src_tgt world_corresp_def)
  subgoal G3 for w_1 w_2
    by (meson morph_worlds_correspond_tgt_src world_corresp_def)
  subgoal G4 for w_1 w_2
    by (meson morph_worlds_correspond_tgt_src world_corresp_def)
  done


lemma morph_reflects_src_indep_simp[simp]:
  assumes ‹x ∈ src.𝒫› ‹y ∈ src.𝒫›
  shows ‹tgt.indep (φ x) (φ y) ⟷ src.indep x y›
  using assms morph_reflects_src_indep by autoend
```

C.2.3 The Category of Particular Structures

```
lemma particular_struct_morphism_comp:
  fixes
    Γ_1 :: ‹('p_1,'q) particular_struct› and
    Γ_2 :: ‹('p_2,'q) particular_struct› and
    Γ_3 :: ‹('p_3,'q) particular_struct›
  assumes
    ‹particular_struct_morphism Γ_1 Γ_2 φ_12›
    ‹particular_struct_morphism Γ_2 Γ_3 φ_23›
  shows
    ‹particular_struct_morphism Γ_1 Γ_3 (φ_23 ∘ φ_12)›
proof -
  interpret M1: particular_struct_morphism ‹Γ_1› ‹Γ_2› ‹φ_12› using assms by simp
  interpret M2: particular_struct_morphism ‹Γ_2› ‹Γ_3› ‹φ_23› using assms by simp
```

```
interpret M12: pre_particular_struct_morphism Γ1 Γ3 ‹φ23 ∘ φ12›
  apply (unfold_locales ; simp?)
  subgoal quality_space_subset
    using M1.quality_space_subset M2.quality_space_subset by auto
  subgoal morph_preserves_particulars
    by (simp add: M1.morph_preserves_particulars M2.morph_preserves_particulars)
  subgoal morph_reflects_inherence
    using M1.morph_preserves_particulars by auto
  subgoal morph_does_not_add_bearers
    by (metis M1.morph_does_not_add_bearers M1.morph_preserves_particulars M2.morph_does_not
M2.morph_reflects_inherence)
  subgoal morph_reflects_towardness
    using M1.morph_preserves_particulars by auto
  subgoal morph_does_not_add_towards
    by (metis M1.morph_does_not_add_towards M1.morph_preserves_particulars M2.morph_does_not
          M2.morph_reflects_towardness)
  subgoal morph_reflects_quale_assoc
    by (simp add: M1.morph_preserves_particulars)
  done

have m12_morph_worlds_correspond_src_tgt:
   ‹∃ wt. M12.world_corresp ws wt› if A[simp]: ‹ws ∈ M1.src.𝒲› for ws
proof -
  obtain w2 where ‹M1.world_corresp ws w2›
    using A M1.morph_worlds_correspond_src_tgt by fastforce
  then obtain
      B[simp]: ‹w2 ∈ M2.src.𝒲›
        ‹⋀x. x ∈ M1.src.𝒫 ⟹ φ12 x ∈ w2 ⟷ x ∈ ws›
    using M1.world_corresp_E by metis
  obtain wt where ‹M2.world_corresp w2 wt›
    using M2.morph_worlds_correspond_src_tgt[OF B(1)] by metis
  then  obtain C[simp]: ‹wt ∈ M2.tgt.𝒲›
      ‹⋀x. x ∈ M2.src.𝒫 ⟹ φ23 x ∈ wt ⟷ x ∈ w2›
    using M2.world_corresp_E by blast
  show ?thesis
    apply (intro exI[of _ wt] M12.world_corresp_I ; simp?)
    by (simp add: M1.morph_preserves_particulars)
qed

have m12_morph_worlds_correspond_tgt_src:
    ‹∃ ws. M12.world_corresp ws wt› if A[simp]: ‹wt ∈ M2.tgt.𝒲› for wt
proof -
```

```
    obtain w₂ where ‹M2.world_corresp w₂ wₜ›
      using A M2.morph_worlds_correspond_tgt_src by fastforce
    then obtain
        B[simp]: ‹w₂ ∈ M2.src.𝒲›
          ‹⋀x. x ∈ M2.src.𝒫 ⟹ φ₂₃ x ∈ wₜ ⟷ x ∈ w₂›
      using M2.world_corresp_E by metis
    obtain wₛ where ‹M1.world_corresp wₛ w₂›
      using M1.morph_worlds_correspond_tgt_src[OF B(1)] by metis
    then obtain C[simp]: ‹wₛ ∈ M1.src.𝒲›
        ‹⋀x. x ∈ M1.src.𝒫 ⟹ φ₁₂ x ∈ w₂ ⟷ x ∈ wₛ›
      using M1.world_corresp_E by blast
    show ?thesis
      apply (intro exI[of _ wₛ] M12.world_corresp_I ; simp?)
      by (simp add: M1.morph_preserves_particulars)
  qed

  show ?thesis
    apply (unfold_locales)
    subgoal using m12_morph_worlds_correspond_src_tgt .
    subgoal using m12_morph_worlds_correspond_tgt_src .
    done
qedcontext particular_struct
begin

lemma (in particular_struct) id_is_a_morphism[intro!]:
  ‹particular_struct_morphism Γ Γ id›
proof -
  show ?thesis
    apply (unfold_locales ; auto?)
    subgoal for w
      by (intro exI[of _ w] ; auto simp: particular_struct_morphism_sig.world_corresp_def
    subgoal for w
      by (intro exI[of _ w] ; auto simp: particular_struct_morphism_sig.world_corresp_def
    done
qedend
```

C.2.4 Classes of Particular Structure Morphisms

```
locale particular_struct_injection =
    particular_struct_morphism where Typ_p1 = ‹Typ_p1› and Typ_p2 = ‹Typ_p2› and Typ_q
= ‹Typ_q›
  for
```

```
      Typ_p1 :: ‹'p1 itself› and
      Typ_p2 :: ‹'p2 itself› and
      Typ_q :: ‹'q itself›  +
  assumes
    morph_is_injective[intro!,simp]: ‹inj_on φ src.𝒫›

begin end


locale particular_struct_surjection =
    particular_struct_morphism where Typ_p1 = ‹Typ_p1› and Typ_p2 = ‹Typ_p2› and Typ_q
= ‹Typ_q›
  for
      Typ_p1 :: ‹'p1 itself› and
      Typ_p2 :: ‹'p2 itself› and
      Typ_q :: ‹'q itself›  +
  assumes
    morph_is_surjective[simp]: ‹φ ` src.𝒫 = tgt.𝒫›

begin end

locale  particular_struct_bijection =
    particular_struct_injection where Typ_p1 = ‹Typ_p1› and Typ_p2 = ‹Typ_p2› and Typ_q
= ‹Typ_q› +
    particular_struct_surjection where Typ_p1 = ‹Typ_p1› and Typ_p2 = ‹Typ_p2› and Typ_q
= ‹Typ_q›
  for
      Typ_p1 :: ‹'p1 itself› and
      Typ_p2 :: ‹'p2 itself› and
      Typ_q :: ‹'q itself›
beginend


locale particular_struct_endomorphism_sig =
    particular_struct_morphism_sig where Γ1 = ‹Γ› and Γ2 = ‹Γ› and φ = ‹φ›
        and Typ_p1 = ‹Typ_p› and Typ_p2 = ‹Typ_p› and Typ_q = ‹Typ_q› +
    particular_struct_defs where Typ_p = ‹Typ_p› and Typ_q = ‹Typ_q›
  for
    φ :: ‹'p ⇒ 'p› and
    Typ_p :: ‹'p itself› and
    Typ_q :: ‹'q itself›context particular_struct_endomorphism_sig
begin
```

notation *inv_morph* (‹φ^{-1}›)

abbreviation *endurants* (‹$\mathcal{P}$›)
  where ‹$\mathcal{P}$ $\equiv$ *src.endurants*›

notation *world_corresp* (**infix** ‹$\Leftrightarrow$› *75*)
notation *tgt_inheres_in* (**infix** ‹$\lhd$› *75*)
notation *tgt_towards* (**infix** ‹$\longrightarrow$› *75*)
notation *tgt_assoc_quale* (**infix** ‹$\rightsquigarrow$› *75*)

abbreviation *qualifiedParticulars* (‹$\mathcal{P}_q$›)
  where ‹$\mathcal{P}_q$ $\equiv$ *src.qualifiedParticulars*›

abbreviation ‹$\mathcal{P}_{img}$ $\equiv$ φ ‘ $\mathcal{P}$›

notation *src.*$\mathcal{S}$ (‹$\mathcal{S}$›)
notation *src.*$\mathcal{M}$ (‹$\mathcal{M}$›)**end**

locale *particular_struct_endomorphism* =
    particular_struct_endomorphism_sig **where** Typ_p = ‹Typ_p› **and** Typ_q = ‹Typ_q› +
    particular_struct **where** Typ_p = ‹Typ_p› **and** Typ_q = ‹Typ_q› +
    particular_struct_morphism **where** Γ_1 = ‹Γ› **and** Γ_2 = ‹Γ› **and** Typ_{p1} = ‹Typ_p› **and** Typ_{p2} = ‹Typ_p› **and** Typ_q = ‹Typ_q›
  for
    Typ_p :: ‹’p itself› **and**
    Typ_q :: ‹’q itself›**locale** *particular_struct_permutation* =
    particular_struct_endomorphism **where** Typ_p = ‹Typ_p› **and** Typ_q = ‹Typ_q› +
    particular_struct_bijection **where** Γ_1 = ‹Γ› **and** Γ_2 = ‹Γ› **and** Typ_{p1} = ‹Typ_p› **and** Typ_{p2} = ‹Typ_p› **and** Typ_q = ‹Typ_q›
  for
    Typ_p :: ‹’p itself› **and**
    Typ_q :: ‹’q itself›

lemma *particular_struct_injection_comp:*
  fixes
    Γ_1 :: ‹(’p_1,’q) particular_struct› **and**
    Γ_2 :: ‹(’p_2,’q) particular_struct› **and**
    Γ_3 :: ‹(’p_3,’q) particular_struct›
  assumes
    ‹particular_struct_injection Γ_1 Γ_2 φ_{12}›
    ‹particular_struct_injection Γ_2 Γ_3 φ_{23}›

```
  shows
    ‹particular_struct_injection Γ1 Γ3 (φ23 ∘ φ12)›
proof -
  interpret M1: particular_struct_injection ‹Γ1› ‹Γ2› ‹φ12› using assms by simp
  interpret M2: particular_struct_injection ‹Γ2› ‹Γ3› ‹φ23› using assms by simp

  interpret particular_struct_morphism ‹Γ1› ‹Γ3› ‹φ23 ∘ φ12›
    using particular_struct_morphism_comp
      M1.particular_struct_morphism_axioms M2.particular_struct_morphism_axioms
    by metis
  show ‹?thesis›
    apply (unfold_locales)
    using M1.morph_is_injective M2.morph_is_injective
    by (metis M1.morph_image_def M1.morph_scope comp_inj_on inj_on_subset)
qed

lemma particular_struct_surjection_comp:
  fixes
    Γ1 :: ‹('p1,'q) particular_struct› and
    Γ2 :: ‹('p2,'q) particular_struct› and
    Γ3 :: ‹('p3,'q) particular_struct›
  assumes
    ‹particular_struct_surjection Γ1 Γ2 φ12›
    ‹particular_struct_surjection Γ2 Γ3 φ23›
  shows
    ‹particular_struct_surjection Γ1 Γ3 (φ23 ∘ φ12)›
proof -
  interpret M1: particular_struct_surjection ‹Γ1› ‹Γ2› ‹φ12› using assms by simp
  interpret M2: particular_struct_surjection ‹Γ2› ‹Γ3› ‹φ23› using assms by simp

  interpret particular_struct_morphism ‹Γ1› ‹Γ3› ‹φ23 ∘ φ12›
    using particular_struct_morphism_comp
      M1.particular_struct_morphism_axioms M2.particular_struct_morphism_axioms
    by metis
  show ‹?thesis›
    apply (unfold_locales)
    using M1.morph_is_surjective M2.morph_is_surjective
    by (metis image_comp)
qed

lemma particular_struct_bijection_comp:
  fixes
```

```
    Γ1 :: ‹('p1,'q) particular_struct› and
    Γ2 :: ‹('p2,'q) particular_struct› and
    Γ3 :: ‹('p3,'q) particular_struct›
  assumes
    ‹particular_struct_bijection Γ1 Γ2 φ12›
    ‹particular_struct_bijection Γ2 Γ3 φ23›
  shows
    ‹particular_struct_bijection Γ1 Γ3 (φ23 ∘ φ12)›
  using assms particular_struct_bijection_def
    particular_struct_injection_comp
    particular_struct_surjection_comp
  by smt

lemma particular_struct_endomorphism_comp:
  fixes
    Γ1 :: ‹('p1,'q) particular_struct› and
    Γ2 :: ‹('p2,'q) particular_struct› and
    Γ3 :: ‹('p3,'q) particular_struct›
  assumes
    ‹particular_struct_endomorphism Γ φ1›
    ‹particular_struct_endomorphism Γ φ2›
  shows
    ‹particular_struct_endomorphism Γ (φ2 ∘ φ1)›
  by (meson assms particular_struct_endomorphism_def
            particular_struct_morphism_comp)

lemma particular_struct_permutation_comp:
  fixes
    Γ1 :: ‹('p1,'q) particular_struct› and
    Γ2 :: ‹('p2,'q) particular_struct› and
    Γ3 :: ‹('p3,'q) particular_struct›
  assumes
    ‹particular_struct_permutation Γ φ1›
    ‹particular_struct_permutation Γ φ2›
  shows
    ‹particular_struct_permutation Γ (φ2 ∘ φ1)›
  by (meson assms particular_struct_permutation_def
            particular_struct_bijection_comp
            particular_struct_endomorphism_comp)definition morphs (‹Morphs _,_› [999,999]
1000) where
‹MorphsΓ,Γ' ≡ {φ . particular_struct_morphism Γ Γ' φ }›
```

definition *injectives* (‹*InjMorphs_,_*› *[999,999] 1000)* **where**
‹$InjMorphs_{\Gamma,\Gamma'}$ ≡ {φ . *particular_struct_injection* Γ Γ' φ }›

definition *surjectives* (‹*SurjMorphs_,_*› *[999,999] 1000)* **where**
‹$SurjMorphs_{\Gamma,\Gamma'}$ ≡ {φ . *particular_struct_surjection* Γ Γ' φ }›

definition *bijectives* (‹*BijMorphs_,_*› *[999,999] 1000)* **where**
‹$BijMorphs_{\Gamma,\Gamma'}$ ≡ {φ . *particular_struct_bijection* Γ Γ' φ }›

definition *endomorphisms* (‹*EndoMorphs_*› *[999] 1000)* **where**
‹$EndoMorphs_{\Gamma}$ ≡ {φ . *particular_struct_endomorphism* Γ φ }›

definition *permutations* (‹*Perms_*› *[999] 1000)* **where**
‹$Perms_{\Gamma}$ ≡ {φ . *particular_struct_permutation* Γ φ }›

lemma *morphs_I[intro!]:*
‹*particular_struct_morphism* Γ Γ' φ ⟹ φ ∈ $Morphs_{\Gamma,\Gamma'}$›
by *(auto simp: morphs_def)*

lemma *morphs_D[dest!]:* ‹φ ∈ $Morphs_{\Gamma,\Gamma'}$ ⟹ *particular_struct_morphism* Γ Γ' φ›
by *(auto simp: morphs_def)*

lemma *morphs_iff[simp]:* ‹φ ∈ $Morphs_{\Gamma,\Gamma'}$ ⟷ *particular_struct_morphism* Γ Γ' φ›
by *(auto simp: morphs_def)*

lemma *injectives_I[intro!]:*
‹*particular_struct_injection* Γ Γ' φ ⟹ φ ∈ $InjMorphs_{\Gamma,\Gamma'}$›
by *(auto simp: injectives_def)*

lemma *injectives_D[dest!]:*
‹φ ∈ $InjMorphs_{\Gamma,\Gamma'}$ ⟹ *particular_struct_injection* Γ Γ' φ›
by *(auto simp: injectives_def)*

lemma *injectives_iff[simp]:* ‹φ ∈ $InjMorphs_{\Gamma,\Gamma'}$ ⟷ *particular_struct_injection* Γ Γ' φ›
by *(auto simp: injectives_def)*

lemma *surjectives_I[intro!]:*
‹*particular_struct_surjection* Γ Γ' φ ⟹ φ ∈ $SurjMorphs_{\Gamma,\Gamma'}$›
by *(auto simp: surjectives_def)*

lemma *surjectives_D[dest!]:*

‹$\varphi \in$ *SurjMorphs*$_{\Gamma,\Gamma'}$ $\Longrightarrow$ *particular_struct_surjection* Γ Γ' φ›
by *(auto simp: surjectives_def)*

lemma *surjectives_iff[simp]:*
‹$\varphi \in$ *SurjMorphs*$_{\Gamma,\Gamma'}$ $\longleftrightarrow$ *particular_struct_surjection* Γ Γ' φ›
by *(auto simp: surjectives_def)*

lemma *bijectives_I[intro!]:*
‹*particular_struct_bijection* Γ Γ' φ $\Longrightarrow$ $\varphi \in$ *BijMorphs*$_{\Gamma,\Gamma'}$›
by *(auto simp: bijectives_def)*

lemma *bijectives_D[dest!]:*
‹$\varphi \in$ *BijMorphs*$_{\Gamma,\Gamma'}$ $\Longrightarrow$ *particular_struct_bijection* Γ Γ' φ›
by *(auto simp: bijectives_def)*

lemma *bijectives_iff[simp]:*
‹$\varphi \in$ *BijMorphs*$_{\Gamma,\Gamma'}$ $\longleftrightarrow$ *particular_struct_bijection* Γ Γ' φ›
by *(auto simp: bijectives_def)*

lemma *endomorphisms_I[intro!]:* ‹*particular_struct_endomorphism* Γ φ $\Longrightarrow$ $\varphi \in$ *EndoMorphs*$_{\Gamma}$›
by *(auto simp: endomorphisms_def)*

lemma *endomorphisms_D[dest!]:* ‹$\varphi \in$ *EndoMorphs*$_{\Gamma}$ $\Longrightarrow$ *particular_struct_endomorphism* Γ φ›
by *(auto simp: endomorphisms_def)*

lemma *endomorphisms_iff[simp]:* ‹$\varphi \in$ *EndoMorphs*$_{\Gamma}$ $\longleftrightarrow$ *particular_struct_endomorphism* Γ φ›
by *(auto simp: endomorphisms_def)*

lemma *permutations_I[intro!]:* ‹*particular_struct_permutation* Γ φ $\Longrightarrow$ $\varphi \in$ *Perms*$_{\Gamma}$›

by *(auto simp: permutations_def)*

lemma *permutations_D[dest!]:* ‹$\varphi \in$ *Perms*$_{\Gamma}$ $\Longrightarrow$ *particular_struct_permutation* Γ φ›

by *(auto simp: permutations_def)*

lemma *permutations_iff[simp]:* ‹$\varphi \in$ *Perms*$_{\Gamma}$ $\longleftrightarrow$ *particular_struct_permutation* Γ φ›
by *(auto simp: permutations_def)*

abbreviation ‹*invMorph* ≡ *particular_struct_morphism_sig.inv_morph*›

lemma *morphisms_are_closed_under_comp[intro]:*
‹$\llbracket\ \varphi_a \in Morphs_{\Gamma_1,\Gamma_2}\ ;\ \varphi_b \in Morphs_{\Gamma_2,\Gamma_3}\ \rrbracket \Longrightarrow \varphi_b \circ \varphi_a \in Morphs_{\Gamma_1,\Gamma_3}$›
by *(simp add: particular_struct_morphism_comp)*

lemma *injective_morphisms_are_closed_under_comp[intro]:*
‹$\llbracket\ \varphi_a \in InjMorphs_{\Gamma_1,\Gamma_2}\ ;\ \varphi_b \in InjMorphs_{\Gamma_2,\Gamma_3}\ \rrbracket \Longrightarrow \varphi_b \circ \varphi_a \in InjMorphs_{\Gamma_1,\Gamma_3}$›
by *(simp add: particular_struct_injection_comp)*

lemma *surjective_morphisms_are_closed_under_comp[intro]:*
‹$\llbracket\ \varphi_a \in SurjMorphs_{\Gamma_1,\Gamma_2}\ ;\ \varphi_b \in SurjMorphs_{\Gamma_2,\Gamma_3}\ \rrbracket \Longrightarrow \varphi_b \circ \varphi_a \in SurjMorphs_{\Gamma_1,\Gamma_3}$›

by *(simp add: particular_struct_surjection_comp)*

lemma *bijective_morphisms_are_closed_under_comp[intro]:*
‹$\llbracket\ \varphi_a \in BijMorphs_{\Gamma_1,\Gamma_2}\ ;\ \varphi_b \in BijMorphs_{\Gamma_2,\Gamma_3}\ \rrbracket \Longrightarrow \varphi_b \circ \varphi_a \in BijMorphs_{\Gamma_1,\Gamma_3}$›
by *(simp add: particular_struct_bijection_comp)*

lemma *endomorphisms_are_closed_under_comp[intro]:*
‹$\llbracket\ \varphi_a \in EndoMorphs_{\Gamma}\ ;\ \varphi_b \in EndoMorphs_{\Gamma}\ \rrbracket \Longrightarrow \varphi_b \circ \varphi_a \in EndoMorphs_{\Gamma}$›
by *(simp add: particular_struct_endomorphism_comp)*

lemma *permutations_are_closed_under_comp[intro]:*
‹$\llbracket\ \varphi_a \in Perms_{\Gamma}\ ;\ \varphi_b \in Perms_{\Gamma}\ \rrbracket \Longrightarrow \varphi_b \circ \varphi_a \in Perms_{\Gamma}$›
by *(simp add: particular_struct_permutation_comp)*

lemma *injections_are_morphisms:* ‹$\varphi \in InjMorphs_{\Gamma_1,\Gamma_2} \Longrightarrow \varphi \in Morphs_{\Gamma_1,\Gamma_2}$›
by *(simp add: particular_struct_injection_def)*

lemma *surjections_are_morphisms:* ‹$\varphi \in SurjMorphs_{\Gamma_1,\Gamma_2} \Longrightarrow \varphi \in Morphs_{\Gamma_1,\Gamma_2}$›
by *(simp add: particular_struct_surjection_def)*

lemma *bijections_are_injections_and_surjections:*
‹$\varphi \in BijMorphs_{\Gamma_1,\Gamma_2} \longleftrightarrow \varphi \in InjMorphs_{\Gamma_1,\Gamma_2} \wedge \varphi \in SurjMorphs_{\Gamma_1,\Gamma_2}$›
by *(simp add: particular_struct_bijection_def)*

lemma *bijections_eq_injections_int_surjections:*
‹$BijMorphs_{\Gamma_1,\Gamma_2} = InjMorphs_{\Gamma_1,\Gamma_2} \cap SurjMorphs_{\Gamma_1,\Gamma_2}$›
by *(intro set_eqI ; simp only: Int_iff bijections_are_injections_and_surjections*
)

lemma *bijections_are_injections:* ‹$\varphi \in BijMorphs_{\Gamma_1,\Gamma_2} \Longrightarrow \varphi \in InjMorphs_{\Gamma_1,\Gamma_2}$›
by *(simp add: bijections_are_injections_and_surjections particular_struct_bijection_def*

lemma *bijections_are_surjections:* ‹$\varphi \in BijMorphs_{\Gamma_1,\Gamma_2} \Longrightarrow \varphi \in SurjMorphs_{\Gamma_1,\Gamma_2}$›
by *(simp add: bijections_are_injections_and_surjections particular_struct_bijection_def*

lemma *bijections_are_morphisms:* ‹$\varphi \in BijMorphs_{\Gamma_1,\Gamma_2} \Longrightarrow \varphi \in Morphs_{\Gamma_1,\Gamma_2}$›
using *bijections_are_injections*
injections_are_morphisms
by *metis*

lemma *endormorphisms_are_morphisms:* ‹$Morphs_{\Gamma,\Gamma}$ = $EndoMorphs_{\Gamma}$›
apply *(intro set_eqI)*
by *(simp add: endomorphisms_def particular_struct_endomorphism_def*
particular_struct_morphism_def
pre_particular_struct_morphism_def)

lemma *permutations_are_endomorphisms:* ‹$\varphi \in Perms_{\Gamma} \Longrightarrow \varphi \in EndoMorphs_{\Gamma}$›
by *(simp add: particular_struct_permutation_def)*

lemma *permutations_are_bijections:* ‹$\varphi \in Perms_{\Gamma} \Longrightarrow \varphi \in BijMorphs_{\Gamma,\Gamma}$›
by *(simp add: particular_struct_permutation_def)***context** *particular_struct_injection*
begin lemma *inv_morph_morph[simp]:*
assumes ‹$x \in src.\mathcal{P}$›
shows ‹φ^{-1} $(\varphi\ x) = x$›
apply *(simp add: inv_morph_def)*
using *inv_into_f_f[OF morph_is_injective] assms* **by** *simp***lemma** *morph_bij_on_img[intro!,*
‹*bij_betw* φ $src.\mathcal{P}$ $\mathcal{P}_{img}$›
apply *(simp only: morph_image_def)*
by *(intro inj_on_imp_bij_betw ; simp)***lemma** *phi_inv_img[simp]:* ‹φ^{-1} ‘ $\mathcal{P}_{img}$ = $src.\mathcal{P}$›

by *(simp add: inv_morph_def morph_image_def)***lemma** *phi_inv_scope[intro]:* ‹$x \in \mathcal{P}_{img} \Longrightarrow \varphi^{-1}\ x \in src.\mathcal{P}$›
by *auto*

lemma *phi_inv_inj_on_I_img[intro!,simp]:* ‹*inj_on* φ^{-1} $\mathcal{P}_{img}$›
by *(simp add: inj_on_def morph_image_def)***lemma** *phi_inv_bij_on_src_I[intro!,simp]:*
‹*bij_betw* φ^{-1} $\mathcal{P}_{img}$ $src.\mathcal{P}$ ›
apply *(simp only: phi_inv_img[symmetric])*
by *(intro inj_on_imp_bij_betw ; simp)***lemma** *morph_inv_morph_img:*
assumes ‹$x \in \mathcal{P}_{img}$›

```
  shows ‹φ (φ⁻¹ x) = x›
  apply (simp add: inv_morph_def)
  apply (intro f_inv_into_f)
  using assms by blastlemma inv_inheres_in_reflects_on_image[simp]:
  assumes ‹x ∈ 𝒫_img› ‹y ∈ 𝒫_img›
  shows  ‹x ◁_t y ⟷ φ⁻¹ x ◁_s φ⁻¹ y›
proof -
  obtain A: ‹φ⁻¹ x ∈ src.𝒫› ‹φ⁻¹ y ∈ src.𝒫›
    using phi_inv_scope assms by blast
  obtain B[simp]: ‹φ (φ⁻¹ x) = x› ‹φ (φ⁻¹ y) = y›
    using assms morph_inv_morph_img by simp
  have C[simp]: ‹(∃y'. φ⁻¹ x ◁_s y' ∧ y = φ y') ⟷ φ⁻¹ x ◁_s φ⁻¹ y›
    by (metis B(2) inv_morph_morph src.inherence_scope)
  show ‹?thesis›
    using morph_reflects_inherence[of ‹φ⁻¹ x› ‹φ⁻¹ y›] A B C by metis
qedlemma inv_towardness_reflects_on_image[simp]:
  assumes ‹x ∈ 𝒫_img› ‹y ∈ 𝒫_img›
  shows  ‹x ⟶_t y ⟷ φ⁻¹ x ⟶_s φ⁻¹ y›
proof -
  obtain A: ‹φ⁻¹ x ∈ src.𝒫› ‹φ⁻¹ y ∈ src.𝒫›
    using phi_inv_scope assms by blast
  obtain B[simp]: ‹φ (φ⁻¹ x) = x› ‹φ (φ⁻¹ y) = y›
    using assms  morph_inv_morph_img morph_image_def by simp
  have C[simp]: ‹(∃z. φ⁻¹ x ⟶_s z ∧ y = φ z) ⟷ φ⁻¹ x ⟶_s φ⁻¹ y›
  proof
    assume ‹∃z. φ⁻¹ x ⟶_s z ∧ y = φ z›
    then obtain z where z: ‹φ⁻¹ x ⟶_s z› ‹y = φ z› by blast
    have ‹z ∈ src.𝒫› using src.towardness_scope z(1) by blast
    then have ‹φ⁻¹ x ⟶_s φ⁻¹ (φ z)›
      using inv_morph_morph z(1) by metis
    then show ‹φ⁻¹ x ⟶_s φ⁻¹ y› using z(2) by simp
  next
    assume as: ‹φ⁻¹ x ⟶_s φ⁻¹ y›
    show ‹∃z. φ⁻¹ x ⟶_s z ∧ y = φ z›
      apply (intro exI[of _ ‹φ⁻¹ y›] conjI as)
      using morph_inv_morph_img assms(2) by simp
  qed
  then show ‹?thesis›
    using morph_reflects_towardness A(1) morph_inv_morph_img assms(1)
    by (metis A(2) B(2))
qedlemma world_preserve_inv_img:
  assumes ‹w_t ∈ tgt.𝒲›
```

```
  shows ‹φ⁻¹ ` (w_t ∩ 𝒫_img) ∈ src.𝒲›
proof -
  obtain w_s where A: ‹w_s ⇔ w_t›
    using morph_worlds_correspond_tgt_src[OF assms] by blast
  obtain B: ‹w_s ∈ src.𝒲›
    ‹⋀x. x ∈ src.𝒫 ⟹ x ∈ w_s ⟷ φ x ∈ w_t›
    using world_corresp_E[OF A] by metis
  have C: ‹φ⁻¹ ` (w_t ∩ 𝒫_img) = w_s›
    apply (intro set_eqI ; simp add: image_iff B Bex_def ; intro iffI ; (elim exE
conjE)? ; hypsubst? ; simp?)
    subgoal for z
      using B(2) by blast
    subgoal premises P for z
      apply (rule exI[of _ ‹φ z›] ; intro conjI exI[of _ ‹z›])
      subgoal G1 using B P by blast
      subgoal G2 using B P by blast
      subgoal G3 using B P by blast
      using B P G1 G2 G3 by auto
    done
  then show ‹?thesis›
    using B(1) by simp
qedend

context particular_struct_surjection
begin lemma I_img_eq_tgt_I[simp]: ‹𝒫_img = tgt.𝒫›
  by (simp only: morph_image_def morph_is_surjective)lemma inv_morph_morph[simp]:
  assumes ‹x ∈ tgt.𝒫›
  shows ‹φ (φ⁻¹ x) = x›
  apply (simp add: inv_morph_def)
  using f_inv_into_f morph_is_surjective assms
  by metislemma world_preserve_img[intro!]:
  assumes ‹w_s ∈ src.𝒲›
  shows ‹φ ` w_s ∈ tgt.𝒲›
proof -
  obtain w_t where A: ‹w_s ⇔ w_t›
    using morph_worlds_correspond_src_tgt[OF assms] by blast
  obtain B: ‹w_t ∈ tgt.𝒲›
    ‹⋀x. x ∈ src.𝒫 ⟹ x ∈ w_s ⟷ φ x ∈ w_t›
    using world_corresp_E[OF A] by metis
  have C: ‹φ ` w_s = w_t›
    apply (intro set_eqI ; simp add: image_iff B Bex_def ; intro iffI ; (elim exE
conjE)? ; hypsubst? ; simp?)
```

```
    subgoal for z
      using B(2) assms by blast
    subgoal premises P for z
      apply (rule exI[of _ ‹φ⁻¹ z›])
      apply (intro conjI exI[of _ ‹φ⁻¹ z›])
      supply B1 = ‹⋀P. (⟦w_t ∈ tgt.𝒲; ⋀x. x ∈ src.endurants ⟹ (x ∈ w_s) = (φ x
∈ w_t)⟧ ⟹ P) ⟹ P›
      subgoal by (metis I_img_eq_tgt_I P B1 inv_into_into inv_morph_morph inv_morph_def
morph_image_def tgt.𝒫_I)
      by (metis P B1 inv_morph_morph tgt.𝒫_I)
    done
  then show ‹?thesis›
    using B(1) by simp
qedlemma tgt_quality_spaces_eq_src[simp]: ‹tgt.𝒬𝒮 = src.𝒬𝒮›
proof (rule ; (intro quality_space_subset subsetI)?)
  fix Q
  assume ‹Q ∈ tgt.𝒬𝒮›
  then obtain x q where A: ‹x ⇝_t q› ‹q ∈ Q›
    using tgt.every_quality_space_is_used by blast
  then obtain B: ‹x ∈ tgt.𝒫› ‹x ∈ tgt.ℳ›
    using tgt.assoc_quale_scopeD by blast
  then obtain w_t where C: ‹w_t ∈ tgt.𝒲› ‹x ∈ w_t›
    by blast
  then obtain w_s where C1: ‹w_s ⇔ w_t›
    using morph_worlds_correspond_tgt_src by fastforce
  then obtain w_s where D: ‹w_s ∈ src.𝒲›
    ‹⋀x. x ∈ src.𝒫 ⟹ x ∈ w_s⟷ φ x ∈ w_t›
    using world_corresp_E[OF C1] B by metis
  obtain y where ‹x = φ y› ‹y ∈ src.𝒫›
    using morph_is_surjective B(1) by blast
  then have ‹y ⇝_s q› using A(1) morph_reflects_quale_assoc by blast
  then show ‹Q ∈ src.𝒬𝒮›
    using A(2) quality_space_subset
    by (smt ‹Q ∈ tgt.𝒬𝒮› quality_space.qspace_eq_I src.𝒬_E src.assoc_quale_scopeD(2)
subsetD tgt.quality_space_axioms)
qedend

context particular_struct_bijection
begin

lemma morph_bijective[intro!,simp]: ‹bij_betw φ src.𝒫 tgt.𝒫›
  using morph_bij_on_img by simplemma inv_morph_bijective[intro!,simp]: ‹bij_betw
```

```
φ⁻¹ tgt.𝒫 src.𝒫›
  using phi_inv_bij_on_src_I by simplemma inv_morph_injective[intro!,simp]: ‹inj_on
φ⁻¹ tgt.𝒫›
  using phi_inv_inj_on_I_img I_img_eq_tgt_I by simp

declare
    inv_towardness_reflects_on_image[simp del]
    inv_inheres_in_reflects_on_image[simp del]lemma inv_inheres_in_reflects[simp]:
  assumes ‹x ∈ tgt.𝒫› ‹y ∈ tgt.𝒫›
  shows  ‹φ⁻¹ x ◁s φ⁻¹ y ⟷ x ◁t y›
  using inv_inheres_in_reflects_on_image assms by simplemma inv_towardness_reflects[sim
  assumes ‹x ∈ tgt.𝒫› ‹y ∈ tgt.𝒫›
  shows  ‹φ⁻¹ x ⟶s φ⁻¹ y ⟷ x ⟶t y›
  using assms inv_towardness_reflects_on_image
  by simplemma inv_assoc_quale_reflects[simp]:
  assumes ‹x ∈ tgt.𝒫› ‹y ∈ tgt.𝒫›
  shows  ‹φ⁻¹ x ⇝s q ⟷ x ⇝t q›
  using assms
  by (simp add: phi_inv_scope)lemma world_preserve_inv_img1[intro!]:
  assumes ‹wt ∈ tgt.𝒲›
  shows ‹φ⁻¹ ‘ wt ∈ src.𝒲›
  using assms world_preserve_inv_img assms
  by (metis I_img_eq_tgt_I inf.orderE tgt.worlds_are_made_of_particulars)lemma
phi_phi_inv_world[simp]:
  assumes ‹w ∈ tgt.𝒲›
  shows ‹φ ‘ φ⁻¹ ‘ w = w›
  by (simp add: assms image_inv_into_cancel inv_morph_def
      possible_worlds.worlds_are_made_of_particulars tgt.possible_worlds_axioms)lemma
phi_inv_phi_world[simp]:
  assumes ‹w ∈ src.𝒲›
  shows ‹φ⁻¹ ‘ φ ‘ w = w›
proof -
  have ‹x ∈ w ⟹ x ∈ src.𝒫› for x using assms by blast
  then have ‹φ⁻¹ (φ x) = x› if ‹x ∈ w› for x by (simp add: that)
  then show ‹?thesis›
    by (simp add: assms particular_struct_morphism_sig.inv_morph_def src.worlds_are_made_

qedlemma src_world_corresp_unique:
  assumes ‹w ⇔ w1› ‹w ⇔ w2›
  shows ‹w1 = w2›
proof -
  obtain A: ‹w ∈ src.𝒲› ‹w1 ∈ tgt.𝒲›
```

```
      ‹⋀x. x ∈ src.𝒫 ⟹ x ∈ w ⟷ φ x ∈ w₁›
    using assms(1) world_corresp_E by blast
  obtain B: ‹w₂ ∈ tgt.𝒲›
      ‹⋀x. x ∈ src.𝒫 ⟹ x ∈ w ⟷ φ x ∈ w₂›
    using assms(2) world_corresp_E by blast
  obtain C: ‹∀x. x ∈ src.𝒫 ⟶ (φ x ∈ w₁ ⟷ φ x ∈ w₂)›
            ‹∀x. x ∈ src.𝒫 ⟶ (φ x ∈ w₂ ⟷ φ x ∈ w₁)›
    using A B by metis
  have D: ‹x ∈ w₁› if as: ‹w₁ ∈ tgt.𝒲› ‹w₂ ∈ tgt.𝒲› ‹∀x. x ∈ src.𝒫 ⟶ (φ x ∈ w₁
⟷ φ x ∈ w₂)› ‹x ∈ w₂› for x w₁ w₂
  proof -
    have ‹x ∈ tgt.𝒫› using as by blast
    then obtain y where BB: ‹x = φ y› ‹y ∈ src.𝒫›
      using I_img_eq_tgt_I by blast
    then show ‹?thesis› using BB as by metis
  qed
  show ‹?thesis›
    using D[OF A(2) B(1) C(1)] D[OF B(1) A(2) C(2)] by blast
qedlemma tgt_world_corresp_unique:
  assumes ‹w₁ ⇔ w› ‹w₂ ⇔ w›
  shows ‹w₁ = w₂›
proof -
  obtain A: ‹w ∈ tgt.𝒲› ‹w₁ ∈ src.𝒲›
      ‹⋀x. x ∈ src.𝒫 ⟹ x ∈ w₁ ⟷ φ x ∈ w›
    using assms(1) world_corresp_E by blast
  obtain B: ‹w₂ ∈ src.𝒲›
      ‹⋀x. x ∈ src.𝒫 ⟹ x ∈ w₂ ⟷ φ x ∈ w›
    using assms(2) world_corresp_E by blast
  obtain C: ‹⋀x. x ∈ src.𝒫 ⟹ x ∈ w₁ ⟷ x ∈ w₂›
    using A B by metis
  then have D: ‹⋀x. x ∈ w₁ ⟷ x ∈ w₂›
    using A(2) B(1) src.worlds_are_made_of_particulars by blast
  then show ‹?thesis›
    by auto
qedlemma src_world_corresp_image:
  assumes ‹w ∈ src.𝒲›
  shows ‹w ⇔ φ ‘ w›
proof -
  obtain wₜ where A: ‹w ⇔ wₜ›
    using morph_worlds_correspond_src_tgt assms
    by fastforce
  then obtain B: ‹wₜ ∈ tgt.𝒲›
```

```
      ‹⋀x. x ∈ src.𝒫 ⟹ x ∈ w ⟷ φ x ∈ w_t›
    using world_corresp_E by metis
  obtain C: ‹x ∈ w ⟹ x ∈ src.𝒫› for x using assms
    by (simp add: src.𝒫_I)
  obtain D: ‹x ∈ w_t ⟹ x ∈ tgt.𝒫› for x using B(1)
    by auto
  have E: ‹φ ` w = w_t›
  proof (intro set_eqI iffI ; (elim imageE)? ; simp add: image_def Bex_def ; hypsubst_thi
    show G1: ‹φ x ∈ w_t› if ‹x ∈ w› for x
      using B(2) C that by metis
    show ‹∃y. y ∈ w ∧ x = φ y› if ‹x ∈ w_t› for x
      supply x_in_tgt_I[simp,intro!] = D[OF that]
      supply phi_phi_inv[simp] = inv_morph_morph[OF x_in_tgt_I]
      apply (intro exI[of _ ‹φ⁻¹ x›] conjI ; simp?)
      apply (intro B(2)[of ‹φ⁻¹ x›,simplified,THEN iffD2] that)
      by (simp add: phi_inv_scope)
  qed
  then show ‹?thesis›
    using A B by simp
qed

lemma tgt_world_corresp_inv_image[intro!]:
  assumes ‹w ∈ tgt.𝒲›
  shows ‹φ⁻¹ ` w ⇔ w›
proof -
  obtain w_s where A: ‹w_s ⇔ w›
    using morph_worlds_correspond_tgt_src assms
    by fastforce
  then obtain B: ‹w_s ∈ src.𝒲›
      ‹⋀x. x ∈ src.𝒫 ⟹ x ∈ w_s ⟷ φ x ∈ w›
    using world_corresp_E by metis
  obtain C: ‹x ∈ w_s ⟹ x ∈ src.𝒫› for x using assms
    using B(1) by blast
  obtain D: ‹x ∈ w ⟹ x ∈ tgt.𝒫› for x using assms
    by auto
  have E: ‹φ⁻¹ ` w = w_s›
  proof (intro set_eqI iffI ; (elim imageE)? ; simp add: image_def Bex_def ; hypsubst_thi
    show G1: ‹φ⁻¹ x ∈ w_s› if ‹x ∈ w› for x
      using B(2) assms phi_inv_scope tgt.𝒫_I that by auto
    show ‹∃y. y ∈ w ∧ x = φ⁻¹ y› if ‹x ∈ w_s› for x
      apply (intro exI[of _ ‹φ x›] conjI)
      subgoal G1 using A that by blast
```

```
      subgoal using that C by auto
      done
  qed
  then show ‹?thesis›
    using A B by simp
qed

lemma src_world_corresp_image_iff:
  assumes ‹w ∈ src.𝒲›
  shows ‹w ⇔ w' ⟷ w' = φ ` w›
  by (meson assms src_world_corresp_image src_world_corresp_unique)lemma tgt_world_corresp_i
  assumes ‹w ∈ tgt.𝒲›
  shows ‹w' ⇔ w ⟷ w' = φ⁻¹ ` w›
  by (meson assms tgt_world_corresp_inv_image tgt_world_corresp_unique)lemma inv_is_bijectiv
‹particular_struct_bijection Γ₂ Γ₁ (φ⁻¹)›
proof -
  interpret I: pre_particular_struct_morphism Γ₂ Γ₁ ‹φ⁻¹› ‹TYPE('p₂)› ‹TYPE('p₁)›
    apply (unfold_locales)
    subgoal G1 by simp
    subgoal G2 by (simp add: phi_inv_scope)
    subgoal G5 by (metis inv_inheres_in_reflects)
    subgoal G6 using phi_inv_img by auto
    subgoal G7 by simp
    subgoal G8 by blast
    by (simp add: G2)

  interpret I: particular_struct_morphism Γ₂ Γ₁ ‹φ⁻¹› ‹TYPE('p₂)› ‹TYPE('p₁)›
    apply (unfold_locales)
    subgoal morph_worlds_correspond_src_tgt for wₜ
      apply (intro exI[of _ ‹φ⁻¹ ` wₜ›] ; simp ; intro particular_struct_morphism_sig.world_co
; simp?)
      subgoal G3_1 by blast
      subgoal G3_2 by (metis image_eqI inv_morph_morph phi_phi_inv_world)
      done
    subgoal morph_worlds_correspond_tgt_src for wₛ
      apply (intro exI[of _ ‹φ ` wₛ›] ; simp ; intro particular_struct_morphism_sig.world_corr
; simp?)
      subgoal G3_1 by blast
      subgoal G3_2 by (metis image_eqI morph_inv_morph_img phi_inv_phi_world I_img_eq_tgt_I)

      done
    done
```

```
  interpret I: particular_struct_injection Γ2 Γ1 ‹φ⁻¹› ‹TYPE('p2)› ‹TYPE('p1)›
    by (unfold_locales ; simp)

  interpret I: particular_struct_surjection Γ2 Γ1 ‹φ⁻¹› ‹TYPE('p2)› ‹TYPE('p1)›
    apply (unfold_locales ; simp)
    using phi_inv_img by auto

  show ?thesis
    by (unfold_locales)
qedend

lemma permutations_are_inj_comp[intro]:
  assumes
    ‹φ1 ∈ Morphs_{Γ1,Γ2}›
    ‹φ2 ∈ Morphs_{Γ2,Γ3}›
    ‹φ2 ∘ φ1 ∈ BijMorphs_{Γ1,Γ3}›
  shows ‹φ1 ∈ InjMorphs_{Γ1,Γ2}›
proof -
  interpret phi1: particular_struct_morphism Γ1 Γ2 φ1
    using assms(1) by simp
  interpret phi2: particular_struct_morphism Γ2 Γ3 φ2
    using assms(2) by simp
  interpret phi21: particular_struct_bijection Γ1 Γ3 ‹φ2 ∘ φ1›
    using assms(3) by simp
  have ‹inj_on φ1 phi1.src.𝒫›
    using phi21.morph_is_injective inj_on_imageI2 by blast
  then show ?thesis
    by (simp ; unfold_locales ; simp)
qed

lemma permutations_are_surj_comp[intro]:
  assumes
    ‹φ1 ∈ Morphs_{Γ1,Γ2}›
    ‹φ2 ∈ Morphs_{Γ2,Γ3}›
    ‹φ2 ∘ φ1 ∈ BijMorphs_{Γ1,Γ3}›
  shows ‹φ2 ∈ SurjMorphs_{Γ2,Γ3}›
proof -
  interpret phi1: particular_struct_morphism Γ1 Γ2 φ1
    using assms(1) by simp
  interpret phi2: particular_struct_morphism Γ2 Γ3 φ2
    using assms(2) by simp
```

```
  interpret phi21: particular_struct_bijection Γ1 Γ3 ‹φ2 ∘ φ1›
    using assms(3) by simp
  have ‹phi2.tgt.𝒫 ⊆ φ2 ` phi2.src.𝒫›
    apply (simp add: image_def Bex_def ; safe)
    subgoal for x
      using [[show_sorts]]
      using exI[of _ ‹invMorph Γ1 (φ2 ∘ φ1) x›]
    using phi21.morph_is_surjective
    by (metis comp_def phi1.morph_preserves_particulars
        phi21.inv_morph_morph phi21.morph_image_def phi21.phi_inv_scope)
    done
  then show ?thesis
    apply (simp ;  unfold_locales)
    by blast
qed
```

C.2.5 Morphism image structure

```
abbreviation lift_morph_1  where
  ‹lift_morph_1 Γ φ p x ≡ ∃y. p Γ y ∧ x = φ y›

abbreviation lift_morph_2 where
  ‹lift_morph_2 Γ φ p x y ≡ ∃x1 y1. p Γ x1 y1 ∧ x = φ x1 ∧ y = φ y1›

abbreviation lift_morph_2_1 ::
  ‹('p1,'a) particular_struct ⇒
   ('p1 ⇒ 'p2) ⇒
   (('p1,'a) particular_struct ⇒ 'p1 ⇒ 'b ⇒ bool) ⇒
   'p2 ⇒
   'b ⇒
   bool› where
  ‹lift_morph_2_1 Γ φ p x z ≡ lift_morph_1 Γ φ (λΓ x. p Γ x z) x›

abbreviation lift_world where
  ‹lift_world φ w ≡ φ ` w›

definition MorphImg :: ‹('p1 ⇒ 'p2) ⇒ ('p1,'q) particular_struct ⇒ ('p2,'q) particular_struc

  where ‹MorphImg φ Γ ≡
  (|
    ps_quality_spaces = ps_quality_spaces Γ,
    ps_worlds = lift_world φ ` ps_worlds Γ,
```

```
    ps_inheres_in = lift_morph_2 Γ φ ps_inheres_in,
    ps_assoc_quale = lift_morph_2_1 Γ φ ps_assoc_quale,
    ps_towards = lift_morph_2 Γ φ ps_towards
  ⦈›

lemma particular_struct_morphism_image_simps[simp]:
  ‹ps_quality_spaces (MorphImg φ Γ) =
    ps_quality_spaces Γ›
  ‹ps_worlds (MorphImg φ Γ) =
    { φ ‘ w | w . w ∈ ps_worlds Γ }›
  ‹ps_inheres_in (MorphImg φ Γ) =
     (λx y. ∃x1 y1. ps_inheres_in Γ x1 y1
          ∧ x = φ x1 ∧ y = φ y1)›
  ‹ps_assoc_quale (MorphImg φ Γ) =
     (λx q. ∃x1. ps_assoc_quale Γ x1 q
          ∧ x = φ x1)›
  ‹ps_towards (MorphImg φ Γ) =
    (λx y. ∃x1 y1. ps_towards Γ x1 y1
          ∧ x = φ x1 ∧ y = φ y1)›
  by (auto simp: MorphImg_def)


context particular_struct_bijection
begin

lemma tgt_is_morph_img: ‹MorphImg φ Γ1 = Γ2›
proof (rule sym)
  show ‹Γ2 = MorphImg φ Γ1›
    apply (auto ; (intro ext iffI conjI)? ; (elim conjE exE)?)
    subgoal G1 using phi_phi_inv_world by blast
    subgoal G2 by (metis inv_inheres_in_reflects inv_morph_morph tgt.inherence_scope)
    subgoal G3 using src.inherence_scope by auto
    subgoal G4 by (metis I_img_eq_tgt_I inv_morph_morph morph_reflects_quale_assoc
particular_struct_injection.phi_inv_scope particular_struct_injection_axioms tgt.assoc_qu
    subgoal G5 using morph_reflects_quale_assoc src.assoc_quale_scopeD(1) by blast

    subgoal G6 by (metis inv_towardness_reflects_on_image morph_inv_morph_img particular_
particular_struct_surjection_axioms tgt.towardness_scopeE)
    using morph_reflects_towardness by blast
qed

declare particular_struct_morphism_image_simps[simp del]
```

```
end

context particular_struct_permutation
begin

lemma morph_img_phi_eq_itself[simp]: ‹MorphImg φ Γ = Γ›
  apply (intro particular_struct_eqI
        ; simp only: particular_struct_morphism_image_simps
        ; (intro ext)?
        ; auto?)
  subgoal G1 using phi_phi_inv_world by blast
  subgoal G2 by (metis I_img_eq_tgt_I inherence.all_inherence_axioms(3) inherence_axioms
inv_morph_morph morph_reflects_inherence phi_inv_scope)
  subgoal G3 using assoc_quale_scopeD(1) morph_reflects_quale_assoc by blast
  subgoal G4 by (metis particular_struct_morphism_image_simps(4) tgt_is_morph_img)
  subgoal G5 using morph_reflects_towardness by blast
  by (metis I_img_eq_tgt_I inv_towardness_reflects morph_inv_morph_img towardness_scopeD(2)
towardness_scopeD(3))

end


lemma isomorphism_tgt_unique:
  fixes
    Γ :: ‹('p1,'q) particular_struct› and
    Γ1 :: ‹('p2,'q) particular_struct› and
    Γ2 :: ‹('p2,'q) particular_struct›
  assumes
    ‹particular_struct_bijection Γ Γ1 φ›
    ‹particular_struct_bijection Γ Γ2 φ›
  shows ‹Γ1 = Γ2›
  using
    assms[THEN particular_struct_bijection.tgt_is_morph_img]
  by simp

lemma isomorphism_iff_isomorphism_to_morphimg:
  ‹particular_struct_bijection Γ1 Γ2 φ ⟷
    particular_struct_bijection Γ1 (MorphImg φ Γ1) φ ∧
    Γ2 =  MorphImg φ Γ1›
  using particular_struct_bijection.tgt_is_morph_img by blast
```

```
locale particular_struct_bijection_1 =
    particular_struct_injection where Γ1 = ‹Γ1›
          and φ = ‹φ› and Γ2 = ‹MorphImg φ Γ1›
          and Typp1 = ‹Typp1› and Typp2 = ‹Typp2› and Typq = ‹Typq› +
    particular_struct_surjection where Γ1 = ‹Γ1›
          and φ = ‹φ› and Γ2 = ‹MorphImg φ Γ1›
          and Typp1 = ‹Typp1› and Typp2 = ‹Typp2› and Typq = ‹Typq›
  for
    Γ1 :: ‹('p1,'q) particular_struct› and
    φ :: ‹'p1 ⇒ 'p2› and
    Typp1 :: ‹'p1 itself› and
    Typp2 :: ‹'p2 itself› and
    Typq :: ‹'q itself›

lemma particular_struct_bijection_iff_particular_struct_bijection_1:
  fixes
    Γ1 :: ‹('p1,'q) particular_struct› and
    Γ2 :: ‹('p2,'q) particular_struct› and
    φ :: ‹'p1 ⇒ 'p2›
  shows
    ‹particular_struct_bijection Γ1 Γ2 φ ⟷
      particular_struct_bijection_1 Γ1 φ ∧
      Γ2 = MorphImg φ Γ1›
  using particular_struct_bijection.tgt_is_morph_img
        particular_struct_bijection_1_def particular_struct_bijection_def
  by blast

lemma particular_struct_bijection_iff_particular_struct_bijection_1_2:
  ‹particular_struct_bijection_1 Γ φ ⟷
    particular_struct_bijection Γ (MorphImg φ Γ) φ›
  supply R =
    particular_struct_bijection_iff_particular_struct_bijection_1[
        of ‹Γ› ‹MorphImg φ Γ› ‹φ›]
  supply P = R[THEN iffD1,THEN conjunct1] R[THEN iffD2,simplified]
  using P by metis

sublocale particular_struct_bijection_1 ⊆
  particular_struct_bijection where  Γ1 = ‹Γ1›
    and φ = ‹φ› and Γ2 = ‹MorphImg φ Γ1›
    and Typp1 = ‹Typp1› and Typp2 = ‹Typp2› and Typq = ‹Typq›
  using particular_struct_bijection_iff_particular_struct_bijection_1
    particular_struct_bijection_1_axioms
```

```
  by metis

lemma particular_struct_bijection_1_comp:
  fixes
    Γ1 :: ‹('p1,'q) particular_struct› and
    Γ2 :: ‹('p2,'q) particular_struct› and
    Γ3 :: ‹('p3,'q) particular_struct›
  assumes
    ‹particular_struct_bijection_1 Γ1 φ12›
    ‹particular_struct_bijection_1 (MorphImg φ12 Γ1) φ23›
  shows
    ‹particular_struct_bijection_1 Γ1 (φ23 ∘ φ12)›
  using assms
    particular_struct_bijection_comp
    particular_struct_bijection_iff_particular_struct_bijection_1
  by metis

lemma (in particular_struct_bijection) is_a_particular_struct_bijection_1:
  ‹particular_struct_bijection_1 Γ1 φ›
proof -
  note tgt_is_morph_img[simp]
  interpret S: particular_struct ‹MorphImg φ Γ1›
    by (simp add: tgt.particular_struct_axioms)
  interpret M: particular_struct_injection Γ1 ‹MorphImg φ Γ1› φ
    apply (simp)
    by (unfold_locales)
  show ?thesis
    by (unfold_locales ; simp)
qed

context particular_struct_permutation
begin

lemma particular_struct_permutation_to_isomorphism_1[intro,simp]:
  ‹particular_struct_bijection_1 Γ φ›
  apply (simp add: particular_struct_bijection_1_def ; intro conjI)
  using particular_struct_injection_axioms particular_struct_surjection_axioms by
auto

end

definition bijections1 :: ‹('p,'q) particular_struct ⇒ 'p1 itself ⇒  ('p ⇒ 'p1)
```

```
set⟩
  (⟨BijMorphs1_,_⟩ [999] 1000)
  where ⟨BijMorphs1_Γ,_ ≡ { φ . particular_struct_bijection_1 Γ φ }⟩

lemma bijections_I[intro!]:
  ⟨particular_struct_bijection_1 Γ φ ⟹ φ ∈ BijMorphs1_Γ,X⟩
  by (auto simp: bijections1_def)

lemma bijections_D[dest!]:
  ⟨φ ∈ BijMorphs1_Γ,X ⟹ particular_struct_bijection_1 Γ φ⟩
  by (auto simp: bijections1_def)

lemma bijections_iff[simp]:
  ⟨φ ∈ BijMorphs1_Γ,X ⟷ particular_struct_bijection_1 Γ φ⟩
  by (auto simp: bijections1_def)

lemma bijections1_iff_bijections_to_morph_img:
  ⟨φ ∈ BijMorphs1_Γ,X ⟷ φ ∈ BijMorphs_Γ,MorphImg φ Γ⟩
  by (intro iffI ; simp add: particular_struct_bijection_iff_particular_struct_bijection_

lemma bijections1_are_morphisms:
  ⟨φ ∈ BijMorphs1_Γ,X ⟹ φ ∈ Morphs_Γ, MorphImg φ Γ⟩
  by (meson bijections1_iff_bijections_to_morph_img bijections_are_morphisms)

lemma permutations_are_bijections1: ⟨φ ∈ Perms_Γ ⟹ φ ∈ BijMorphs1_Γ,X⟩
  by (simp add: particular_struct_permutation.particular_struct_permutation_to_isomorphis


definition isomorphic_models
  :: ⟨('p,'q) particular_struct ⇒ 'p1 itself ⇒
      ('p1,'q) particular_struct set⟩
      (⟨IsoModels_,_⟩ [999,999] 1000) where
  ⟨IsoModels_Γ,X ≡ { MorphImg φ Γ | φ . φ ∈ BijMorphs1_Γ,X }⟩

lemma isomorphic_models_I[intro]:
  assumes ⟨φ ∈ BijMorphs1_Γ,X⟩ ⟨Γ1 = MorphImg φ Γ⟩
  shows ⟨Γ1 ∈ IsoModels_Γ,X⟩
  using assms
  by (auto simp: isomorphic_models_def)

lemma isomorphic_models_E[elim!]:
  assumes ⟨Γ1 ∈ IsoModels_Γ,X⟩
```

```
  obtains φ where ‹φ ∈ BijMorphs1_{Γ,X}› ‹Γ1 = MorphImg φ Γ›
  using assms
  by (auto simp: isomorphic_models_def)

lemma isomorphic_models_iff[simp]:
  ‹Γ' ∈ IsoModels_{Γ,X} ⟷ (∃φ ∈ BijMorphs1_{Γ,X}. Γ' = MorphImg φ Γ)›
  by (auto simp: isomorphic_models_def)

lemma isomorphic_models_sym[sym]:
  assumes ‹Γ1 ∈ IsoModels_{Γ2,TYPE('p1)}›
  shows ‹Γ2 ∈ IsoModels_{Γ1,TYPE('p2)}›
  using assms
  by (metis isomorphic_models_iff bijections1_def mem_Collect_eq
      particular_struct_bijection.inv_is_bijective_morphism
      particular_struct_bijection_iff_particular_struct_bijection_1)

context ufo_particular_theory
begin

lemma MorphImg_of_id[simp]: ‹MorphImg id Γ = Γ›
  by (rule ; simp)

lemma id_is_isomorphism[intro!,simp]: ‹particular_struct_bijection_1 Γ id›
proof -
  interpret particular_struct_morphism Γ Γ id
    apply (simp add:
        particular_struct_bijection_1_def
        particular_struct_injection_def
        particular_struct_morphism_def
        pre_particular_struct_morphism_def
        particular_struct_surjection_def ;
        intro conjI ; unfold_locales ; simp)
    subgoal using inherence_scope by blast
    subgoal using towardness_scope by blast
    subgoal G1 for w
      by (intro exI[of _ ‹w›]
          particular_struct_morphism_sig.world_corresp_I ; simp)
    subgoal G2 for w
      by (intro exI[of _ ‹w›]
          particular_struct_morphism_sig.world_corresp_I ; simp)
    done
  show ?thesis
```

```
    apply (simp add:
      particular_struct_bijection_1_def
      particular_struct_injection_def
      particular_struct_surjection_def ; intro conjI ; unfold_locales)
    by auto
qed

lemma id_is_a_permutation[intro!,simp]: ‹particular_struct_permutation Γ id›
proof -
  interpret id: particular_struct_bijection_1 ‹Γ› ‹id› by simp
  show ‹?thesis›
    by (simp add: particular_struct_permutation_def
          id.particular_struct_bijection_axioms[simplified]
          particular_struct_endomorphism_def
          id.particular_struct_morphism_axioms[simplified]
          ufo_particular_theory_axioms)
qed

lemma id_in_isomorphs[intro!,simp]:
  fixes X
  shows ‹id ∈ BijMorphs1_{Γ,X}›
  by (intro bijections_I id_is_isomorphism)

lemma itself_in_isomodels[intro!,simp]:
  fixes X
  shows ‹Γ ∈ IsoModels_{Γ,X}›
  by (intro isomorphic_models_I[of ‹id›] id_in_isomorphs ; simp)

lemma id_in_permutations[intro!,simp]: ‹id ∈ Perms_Γ›
  by (intro permutations_I id_is_a_permutation)

end

context particular_struct_permutation
begin

lemma inv_morph_permutation[simp,intro!]: ‹particular_struct_permutation Γ φ⁻¹›
proof -
  interpret iso: particular_struct_bijection ‹Γ› ‹Γ› ‹φ⁻¹› by simp
  show ‹?thesis›
    by (intro_locales)
qed
```

```
end

context
begin

lemma inv_morph_in_BijMorphs[intro!,simp]:
  ‹invMorph Γ φ ∈ BijMorphs1MorphImg φ Γ,X›
  if A: ‹φ ∈ BijMorphs1Γ,X›
proof -
  let ‹?invmorph› = ‹invMorph Γ›
  interpret phi: particular_struct_bijection_1 ‹Γ› ‹φ›
    using A by blast
  interpret inv: particular_struct_bijection_1 ‹MorphImg φ Γ› ‹?invmorph φ›
    using particular_struct_bijection_iff_particular_struct_bijection_1 by blast
  show ‹?thesis›
    using inv.particular_struct_bijection_1_axioms
    by blast
qed

lemma inv_morph_in_Perms[intro!,simp]: ‹invMorph Γ φ ∈ PermsΓ› if A: ‹φ ∈ PermsΓ›
proof -
  let ‹?invmorph› = ‹invMorph Γ›
  interpret perm: particular_struct_permutation ‹Γ› ‹φ› using A by simp

  interpret inv: particular_struct_permutation ‹Γ› ‹?invmorph φ›
    by simp
  show ‹?thesis›
    by simp
qed

end

lemma morph_img_comp[simp]: ‹MorphImg (φ1 ∘ φ2) Γ = MorphImg φ1 (MorphImg φ2 Γ)›
  apply (subst eq_commute)
  apply (auto ; (intro ext)?)
  subgoal for w1
    by (intro exI[of _ ‹w1›] ; simp ; blast)
  subgoal for w1
    by (intro exI[of _ ‹φ2 ‘ w1›] ; simp ; blast)
  subgoal for x y
```

```
    apply (intro iffI ; elim exE conjE ; hypsubst_thin)
    subgoal for _ _ x1 y1
      by (rule exI[of _ ‹x1›] ; rule exI[of _ ‹y1›] ; simp)
    subgoal for x1 y1
      apply (rule exI[of _ ‹φ2 x1›] ;
          rule exI[of _ ‹φ2 y1›] ;
          intro conjI ; simp?)
      by (rule exI[of _ ‹x1›] ; rule exI[of _ ‹y1›] ; simp)
    done
  subgoal for x q
    apply (intro iffI ; elim exE conjE ; hypsubst_thin)
    subgoal for x1 x1'
      by (rule exI[of _ ‹x1'›] ;  simp ; blast)
    subgoal for x1
      by (rule exI[of _ ‹φ2 x1›] ;
          intro conjI ; simp? ; blast)
    done
  subgoal for x y
    apply (intro iffI ; elim exE conjE ; hypsubst_thin)
    subgoal for _ _ x1 y1
      by (rule exI[of _ ‹x1›] ; rule exI[of _ ‹y1›] ; simp)
    subgoal for x1 y1
      apply (rule exI[of _ ‹φ2 x1›] ;
          rule exI[of _ ‹φ2 y1›] ;
          intro conjI ; simp?)
      by (rule exI[of _ ‹x1›] ; rule exI[of _ ‹y1›] ; simp)
    done
  done


context ufo_particular_theory
begin

lemma inj_morph_img_valid_structure:
  fixes φ :: ‹'p ⇒ 'p1›
  assumes ‹inj_on φ 𝒫› ‹∃(f :: 'p1 ⇒ ZF). inj f›
  shows ‹particular_struct (MorphImg φ Γ)›
proof -
  define phi_inv (‹φ⁻¹›) where ‹φ⁻¹ ≡ inv_into 𝒫 φ›

  obtain phi_inv:
```

```
    ‹⋀x. x ∈ 𝒫 ⟹ φ⁻¹ (φ x) = x›
    ‹⋀x. x ∈ φ ` 𝒫 ⟹ φ (φ⁻¹ x) = x›
    ‹⋀X. X ⊆ 𝒫 ⟹ φ⁻¹ ` φ ` X = X›
    ‹⋀X. X ⊆ φ ` 𝒫 ⟹ φ ` φ⁻¹ ` X = X›
    ‹⋀X. φ⁻¹ ` φ ` (X ∩ 𝒫) = X ∩ 𝒫›
    ‹⋀X. φ ` φ⁻¹ ` (X ∩ φ ` 𝒫) = X ∩ φ ` 𝒫›
  using assms(1) that
  by (simp add: f_inv_into_f image_inv_into_cancel phi_inv_def)

have same_worlds: ‹w₁ = w₂› if as: ‹w₁ ∈ 𝒲› ‹w₂ ∈ 𝒲› ‹φ ` w₁ = φ ` w₂› for w₁ w₂
  using as worlds_are_made_of_particulars assms(1)
  by (metis phi_inv(3))

have phi_img_inj: ‹X = Y› if ‹X ⊆ 𝒫› ‹Y ⊆ 𝒫› ‹φ ` X = φ ` Y› for X Y
  using that assms(1) by (meson inj_on_image_eq_iff)

let ‹?Γ'› = ‹MorphImg φ Γ›

have morphI_eq[simp]: ‹possible_worlds_sig.𝒫 {φ ` w |w. w ∈ 𝒲} = φ ` 𝒫›
  by (auto simp: possible_worlds_sig.𝒫_def)

have A1: ‹∃w∈{φ ` w |w. w ∈ 𝒲}. x ∉ w›
  if as: ‹x ∈ possible_worlds_sig.𝒫 {φ ` w |w. w ∈ 𝒲}› for x
proof -
  obtain w where AA: ‹w ∈ 𝒲› ‹x ∈ φ ` w›
    apply (rule as[THEN possible_worlds_sig.𝒫_E] ; simp
          ; elim exE conjE ; hypsubst
          ; elim imageE ; hypsubst_thin)
    subgoal premises P for w₁ w₂ y
      by (rule P(1)[of ‹w₂›] ; simp add: P)
    done
  obtain BB: ‹w ⊆ 𝒫› ‹x ∈ φ ` 𝒫› using AA
    worlds_are_made_of_particulars by blast
  obtain y where CC: ‹x = φ y› ‹y ∈ w› ‹y ∈ 𝒫›
    using AA(2) BB by blast
  obtain w' where DD: ‹w' ∈ 𝒲› ‹y ∉ w'›
    using CC(3) particulars_do_not_exist_in_some_world by blast
  have EE: ‹x ∉ φ ` w'›
      by (metis CC(1) CC(3) DD(1) DD(2) image_eqI phi_inv(1) phi_inv(3) worlds_are_made_of_p
  show ‹?thesis›
    apply (intro bexI[of _ ‹φ ` w'›] EE)
    using DD by auto
```

```
  qed

  let ‹?𝒲› = ‹{φ ` w |w. w ∈ 𝒲}›
  let ‹?inheresIn› = ‹λx y. ∃x₁ y₁. x₁ ◁ y₁ ∧ x = φ x₁ ∧ y = φ y₁›
  let ‹?assocQuale› = ‹λx q. ∃x₁. x₁ ⇝ q ∧ x = φ x₁›
  let ‹?towards› = ‹λx y. ∃x₁ y₁. x₁ ⟶ y₁ ∧ x = φ x₁ ∧ y = φ y₁›

  interpret M: possible_worlds ‹?𝒲› ‹TYPE('p₁)›
    apply (unfold_locales)
    subgoal has_inj using assms by blast
    subgoal using at_least_one_possible_world by auto
    subgoal using A1 by blast
    done

  have Med_simp[simp]: ‹M.ed x y ⟷ (∃x₁ y₁. ed x₁ y₁ ∧ x = φ x₁ ∧ y = φ y₁)›
for x y
    apply (simp only: possible_worlds_sig.ed_def ; simp)
    apply (safe ; hypsubst_thin?)
    subgoal G1 premises P for x₁ x₂
      apply (rule exI[of _ ‹x₁›] ; intro conjI P
            ; rule exI[of _ ‹x₂›] ; intro conjI P ; simp?
            ; intro ballI impI)
      subgoal premises Q for w
        supply R1 = P(2)[rule_format,of ‹φ ` w›,
            simplified image_def Bex_def,simplified,
            OF exI[of _ ‹w›],
            simplified,
            OF _ exI[of _ ‹x₂›],
            simplified,OF _ conjI]
        by (meson P Q assms(1) image_eqI inj_on_image_mem_iff worlds_are_made_of_particul
      done
    subgoal G2 premises P for x₁ x₂
      using P by blast
    subgoal G3 premises P for x₁ x₂
      using P by blast
    subgoal premises P for x₁ x₂ w₁ w₂ x₃
      using P apply (simp add: image_def)
      apply (rule bexI[of _ ‹x₂›] ; simp?)
      by (metis 𝒫_I phi_inv(1))
    done
```

```
interpret M: inherence_base ‹?𝒲› ‹?inheresIn› ‹TYPE('p1)›
  apply (unfold_locales ; simp?)
  subgoal G1 by auto
  subgoal G2 by (metis inherence_imp_ed)
  subgoal G3 by (metis bearer_eqI inherence_scope phi_inv(1))
  done

interpret M: noetherian_inherence ‹?𝒲› ‹?inheresIn› ‹TYPE('p1)›
proof (unfold_locales ; simp?)
  have AA: ‹(λx y. ∃x1 y1. x1 ◁ y1 ∧ x = φ x1 ∧ y = φ y1)^-1-1 =
        (λx y. ∃x1 y1. (◁)^-1-1 x1 y1 ∧ x = φ x1 ∧ y = φ y1)›
    by (intro ext ; simp ; blast)
  show ‹wfP (λx y. ∃x1 y1. x1 ◁ y1 ∧ x = φ x1 ∧ y = φ y1)^-1-1›
    apply (subst AA)
    apply (intro wfI[to_pred,of _ ‹φ ` 𝒫› ‹φ ` 𝒫›] subsetI ; safe)
    subgoal using inherence_scope by auto
    subgoal using inherence_scope by auto
    subgoal premises P for x1 P x2 x3
      using P
      apply (induct arbitrary: ‹x2› rule: inherence_is_noetherian[THEN wfP_induct])
      apply simp
      subgoal premises Q for x4 x5
        apply (rule Q(2)[rule_format] ; elim exE conjE ; simp)
        subgoal premises T for x6 x7 x8
          apply (rule Q(1)[rule_format,of ‹x7› ‹x7›,simplified])
          supply R1 = inj_onD[OF assms(1),OF T(3) Q(5),
                          OF inherence_scope[OF T(1),THEN conjunct1]]
          subgoal using T(1) R1 by simp
          using T(1) inherence_scope by simp
        done
      done
    done
qed

interpret M: inherence ‹?𝒲› ‹?inheresIn› ‹TYPE('p1)›
  apply (unfold_locales ; intro wfI[to_pred,of _ ‹φ ` 𝒫› ‹φ ` 𝒫›] ; safe)
  subgoal G1 by blast
  subgoal G2 by blast
  subgoal G3 premises P for x1 P x2 x3
    using P
    apply (induct arbitrary: ‹x2› rule: inherence_is_wf[THEN wfP_induct])
```

```
      apply simp
      subgoal G3_1 premises Q for x4 x5
        apply (rule Q(2)[rule_format] ; elim exE conjE ; simp)
        subgoal G3_1_1 premises T for x6 x7 x8
          apply (rule Q(1)[rule_format,of ‹x7› ‹x7›,simplified])
          supply R1 = inj_onD[OF assms(1),OF T(3) Q(5),
                OF inherence_scope[OF T(1),THEN conjunct2]]
          subgoal G3_1_1_1 using T(1) R1 by simp
          using T(1) inherence_scope by simp
        done
      done
    done

  have M_qual_particular[simp]:
    ‹qualified_particulars_sig.qualifiedParticulars (λx q. ∃x1 w1. x1 ⇝ q ∧ x =
φ x1) =
      φ ` 𝒫q›
    by (auto simp: qualified_particulars_sig.qualifiedParticulars_def)

  interpret M: qualified_particulars ‹?𝒲› ‹?inheresIn› ‹𝒬𝒮› ‹?assocQuale› ‹TYPE('p1)›
    apply (unfold_locales)
    subgoal G1 for x q
      using assoc_quale_scopeD inheres_in_bearerI by blast
    subgoal G2 for x q1 q2
      apply (elim exE conjE ; hypsubst_thin)
      subgoal premises P for x1 x2
        supply S1 = assoc_quale_scopeD[OF P(1)]
        supply S2 = assoc_quale_scopeD[OF P(2)]
        supply q1q2 = inj_onD[OF assms(1),OF P(3) S1(1) S2(1)]
        using P(1,2)[simplified  q1q2] assoc_quale_unique
        by blast
      done
    subgoal G3 for w y1 y2 x q1 q2 Q
      apply (elim imageE exE conjE ; simp ; elim exE conjE imageE ; hypsubst_thin
            ; elim imageE ; simp)
      by (metis 𝒫_I assoc_quale_scopeD(1) inherence_scope phi_inv(1) quality_moment_uniq
    subgoal G4 for Q
      using every_quality_space_is_used by blast
    subgoal G5 using quale_determines_moment
      by (metis assoc_quale_scopeD(3) endurantI1 inherence_scope phi_inv(1))
    done
```

```
  have M_M_eq[simp]: ‹M.𝓜 = φ ` 𝓜›
    by (auto simp: inherence_sig.𝓜_def)

  have trans_inheres_in_scopeD: ‹x ∈ 𝓜› ‹x ∈ 𝒫› ‹y ∈ 𝒫› if ‹x ◁◁ y› for x y
    using that trans_inheres_in_scope by blast+

  have M_inheres_in_trancl[simp]:
    ‹?inheresIn⁺⁺ x y ⟷ (∃x₁ y₁. (◁)⁺⁺ x₁ y₁ ∧ x = φ x₁ ∧ y = φ y₁)› for x y
    apply (intro iffI)
    subgoal G1
      apply (induct rule: tranclp.induct)
      subgoal G1_1 for x₁ y₁
        apply (elim exE conjE ; hypsubst_thin)
        subgoal G1_1_1 for x₂ y₂
          by (rule exI[of _ ‹x₂›] ; rule exI[of _ ‹y₂›] ; simp)
        done
      subgoal G1_2 for x₁ y₁ z₁
        apply (elim exE conjE ; hypsubst_thin)
        subgoal G1_2_1 premises P for x₂ x₃ y₂ y₃
          supply S1 = inherence_scope[OF P(3),THEN conjunct1]
                      inherence_scope[OF P(3),THEN conjunct2]
          supply S2 = trans_inheres_in_scopeD[OF P(2)]
          supply y2x3 = inj_onD[OF assms(1),OF P(4) _ S1(1),OF S2(3)]
          supply P1 = P(1,2,3)[simplified y2x3]
          supply R1 = tranclp.intros(2)[of ‹(◁)›,OF P(2),simplified y2x3,OF P(3)]

          apply (rule exI[of _ ‹x₂›] ; rule exI[of _ ‹y₃›])
          using R1 by simp
        done
      done
    subgoal G2
      apply (elim exE conjE ; hypsubst_thin)
      subgoal G2_1 for x₁ y₁
        apply (induct rule: tranclp.induct)
        subgoal G2_1_1 by auto
        subgoal G2_1_2 for x₂ y₂ z₂
          apply (rule tranclp.intros(2)[of ‹?inheresIn›, of ‹φ x₂› ‹φ y₂› ‹φ z₂›] ;
simp?)
          subgoal premises P
            by (rule exI[of _ ‹y₂›] ; rule exI[of _ ‹z₂›] ; simp add: P)
          done
        done
```

```
      done
    done

  have M_inheres_in_rtranclp[simp]:
    ‹?inheresIn** x y ⟷ x = y ∨ (∃x1 y1. (◁)++ x1 y1 ∧ x = φ x1 ∧ y = φ y1)›
for x y
    by (simp add: Nitpick.rtranclp_unfold)

  have ed_scope: ‹x ∈ ℰ› ‹y ∈ ℰ› if ‹ed x y› for x y
    using that edE by blast+


  have M_ultimateBearer[simp]:
    ‹M.ultimateBearer (φ x) = φ (ultimateBearer x)› if as: ‹x ∈ 𝒫› for x
    using as
    apply (subst Inherence.noetherian_inherence.ultimate_bearer_eq_simp[
        of ‹?𝒲› ‹?inheresIn›,
          simplified,OF M.noetherian_inherence_axioms])
    apply auto
    subgoal G1 by (meson 𝒮_E image_eqI ultimate_bearer_substantial)
    subgoal G2 by (metis endurantI1 inherence_sig.𝒮_E phi_inv(1) ultimate_bearer_substan

    by (metis relpowp_imp_rtranclp rtranclpD ultimate_bearer_and_order)

  have M_directed_moments[simp]: ‹towardness_sig.directed_moments ?towards = φ ‘
ℳ→›
    by (auto simp: towardness_sig.directed_moments_def image_def Bex_def)

  interpret M: towardness ‹?𝒲› ‹?inheresIn› ‹?towards› ‹TYPE('p1)›
    apply (unfold_locales ; simp?)
    subgoal G1 for x y
      apply (elim exE conjE ; hypsubst_thin)
      apply (intro conjI ; simp add: image_def inherence_sig.𝒮_def possible_worlds_sig.𝒫
      subgoal G1_1 by blast
      subgoal G1_2 for x1 y1
        apply (intro conjI ballI)
        subgoal G1_2_1 by (metis 𝒫_E image_def image_eqI towardness_scopeD(3))
        by (metis 𝒮_E assms(1) endurantI1 inj_on_contraD towardness_scope)
      done
    subgoal G2 for x y
      apply (elim exE conjE ; hypsubst_thin)
      subgoal for x1 y1
```

```
        apply (rule exI[of _ ‹x₁›] ; rule exI[of _ ‹y₁›] ; simp)
        by (simp add: towardness_imp_ed)
      done
    subgoal G3 using towardness_diff_ultimate_bearers
      by (smt M_ultimateBearer endurantI1 inherence_sig.𝒮_E noetherian_inherence.ultimate_bea
noetherian_inherence_axioms phi_inv(1) towardness_scope)
    subgoal G4 using towardness_single by (metis endurantI1 phi_inv(1) towardness_apply_to_mo
    done

  interpret M: ufo_particular_theory_sig ‹?𝒲› ‹?inheresIn› ‹𝒬𝒮› ‹?assocQuale› ‹?towards›
‹TYPE('p₁)› .

  interpret M: ufo_particular_theory ‹?𝒲› ‹?inheresIn› ‹𝒬𝒮› ‹?assocQuale› ‹?towards›
‹TYPE('p₁)›
  proof (unfold_locales ; simp ; intro allI impI ; hypsubst_thin
          ; elim M.qualifiedParticularsE exE conjE ; hypsubst_thin)
    fix x₁ y₁ q x₂
    assume as: ‹x₁ ◁ y₁› ‹x₂ ⇝ q›
    show ‹φ x₂ ≠ φ y₁›
    proof
      obtain A: ‹x₁ ∈ 𝒫› ‹y₁ ∈ 𝒫› ‹x₂ ∈ 𝒫›
        using as(1,2) inherence_scope
        by (simp add: assoc_quale_scopeD(1))
      assume ‹φ x₂ = φ y₁›
      then have ‹x₂ = y₁› using ‹inj_on φ 𝒫›[THEN inj_onD] A by blast
      then have ‹y₁ ⇝ q› using as(2) by simp
      then show False using as(1)
        using qualified_particulars_are_not_bearers by blast
    qed
  qed

  show ‹?thesis›
    apply (simp add: particular_struct_def)
    using M.ufo_particular_theory_axioms by simp
qed


lemma inj_morph_img_isomorphism[intro]:
  fixes φ :: ‹'p ⇒ 'p₁›
  assumes ‹inj_on φ 𝒫› ‹∃(f :: 'p₁ ⇒ ZF). inj f›
  shows ‹particular_struct_bijection_1 Γ φ›
proof -
```

```
  note assms[simp]
  interpret M: particular_struct ‹MorphImg φ Γ› ‹TYPE('p1)›
    using inj_morph_img_valid_structure[OF assms] .

  interpret I: pre_particular_struct_morphism Γ ‹MorphImg φ Γ› φ
    apply (simp add:
        pre_particular_struct_morphism_def
        M.ufo_particular_theory_axioms
        M.particular_struct_axioms
        ufo_particular_theory_axioms)
    apply (unfold_locales ; simp add: possible_worlds_sig.𝒫_def
          ; (intro iffI)? ; elim bexE conjE exE ; hypsubst_thin?)
    subgoal G1 by blast
    subgoal G2 by (metis 𝒫_I assms(1) inherence_scope inj_onD)
    subgoal G3 by blast
    subgoal G4 by (metis 𝒫_E inherence_scope)
    subgoal G5 by (metis 𝒫_I assms(1) inj_on_contraD towardness_scopeD(2) towardness_sco
    subgoal G6 by blast
    subgoal G6 by (metis 𝒫_E towardness_scopeE)
    subgoal G7 by blast
    by (metis 𝒫_I assms(1) assoc_quale_scopeD(1) inj_onD)

  interpret I: particular_struct_morphism Γ ‹MorphImg φ Γ› φ
    apply (unfold_locales ; simp add: I.world_corresp_def
        ; (elim exE conjE)? ; hypsubst_thin?
        ; (elim imageE)? ; hypsubst_thin?)
    by (metis assms(1) inj_on_image_mem_iff worlds_are_made_of_particulars)+

  interpret I: particular_struct_injection Γ ‹MorphImg φ Γ› φ
    by (unfold_locales ; simp)

  interpret I: particular_struct_surjection Γ ‹MorphImg φ Γ› φ
    by (unfold_locales ; auto simp: possible_worlds_sig.𝒫_def)

  show ‹?thesis›
    by (unfold_locales)
qed

lemma inj_morph_img_BijMorphs:
  fixes φ :: ‹'p ⇒ 'p1›
  assumes ‹inj_on φ 𝒫› ‹∃(f :: 'p1 ⇒ ZF). inj f›
  shows ‹φ ∈ BijMorphs1Γ,X›
```

```
  apply (intro bijections_I)
  using assms inj_morph_img_isomorphism by metis

lemma BijMorphs_iff_inj[simp]: ‹(φ :: 'p ⇒ 'p1) ∈ BijMorphs1Γ,X ⟷ inj_on φ
𝒫 ∧ (∃(f :: 'p1 ⇒ ZF). inj f)›
proof (intro iffI ; (elim conjE)?)
  show ‹φ ∈ BijMorphs1Γ,X› if as: ‹inj_on φ ℰ› ‹∃f::'p1 ⇒ ZF. inj f›
    using inj_morph_img_BijMorphs[OF as] by simp
  show ‹inj_on φ 𝒫 ∧  (∃(f :: 'p1 ⇒ ZF). inj f)›
    if as: ‹φ ∈ BijMorphs1Γ,X› for φ :: ‹'p ⇒ 'p1› and X
  proof
    interpret I1: particular_struct_bijection_1 ‹Γ› ‹φ›
      using as by blast
    show ‹inj_on φ ℰ› using I1.morph_is_injective by simp
    show ‹∃(f :: 'p1 ⇒ ZF). inj f› using I1.tgt.injection_to_ZF_exist .
  qed
qed

lemma isomorphism_1_iff_inj[simp]:
  ‹particular_struct_bijection_1 Γ (φ :: 'p ⇒ 'p1) ⟷
      inj_on φ 𝒫 ∧ (∃(f :: 'p1 ⇒ ZF). inj f)›
  using BijMorphs_iff_inj
  apply (simp only: bijections1_def)
  by blast

lemma Perms_iff_inj[simp]: ‹φ ∈ PermsΓ ⟷ inj_on φ 𝒫 ∧ MorphImg φ Γ = Γ›
proof -
  have A: ‹φ ∈ BijMorphs1Γ,TYPE('p)› if as: ‹inj_on φ 𝒫›
    using inj_morph_img_BijMorphs[OF as] injection_to_ZF_exist by simp
  have B: ‹φ ∈ PermsΓ› if as: ‹inj_on φ 𝒫› and as1[simp]: ‹MorphImg φ Γ = Γ›
  proof -
    interpret I: particular_struct_bijection_1 ‹Γ› ‹φ› using A as by blast
    interpret I1: particular_struct_permutation ‹Γ› ‹φ›
      apply (simp add: particular_struct_permutation_def
              I.particular_struct_bijection_axioms[simplified as1])
      apply intro_locales
      using I.particular_struct_morphism_axioms[simplified as1]
        I.pre_particular_struct_morphism_axioms[simplified as1]
      by (simp add: particular_struct_morphism_def
                    pre_particular_struct_morphism_def)+
    show ‹?thesis›
      using I1.particular_struct_permutation_axioms by blast
```

```
  qed
  have C: ‹inj_on φ' 𝒫 ∧ MorphImg φ' Γ = Γ› if ‹φ' ∈ Perms_Γ› for φ'
  proof -
    interpret I1: particular_struct_permutation ‹Γ› ‹φ'›
      using that by blast
    interpret I: particular_struct_bijection_1 ‹Γ› ‹φ'›
      using I1.particular_struct_permutation_to_isomorphism_1 by simp
    show ‹inj_on φ' 𝒫 ∧ MorphImg φ' Γ = Γ›
      using I.morph_is_injective by auto
  qed
  show ‹?thesis›
    apply (intro iffI ; (elim conjE)?)
    subgoal by (rule C ; simp)
    subgoal by (rule B ; simp)
    done
qed

end

context particular_struct_bijection_1
begin

lemma phi_in_iso_morphs[intro]: ‹φ ∈ BijMorphs1_src.Γ,X›
  apply simp
  using tgt.injection_to_ZF_exist by blast

lemma tgt_in_src_iso_models[intro]: ‹MorphImg φ src.Γ ∈ IsoModels_src.Γ,X›
  by (intro isomorphic_models_I[of ‹φ›] phi_in_iso_morphs ; simp )

lemma tgt_Gamma_eq_Morph_img[simp]: ‹tgt.Γ = MorphImg φ src.Γ›
  apply (simp add: MorphImg_def)
  by (intro particular_struct_eqI ext ; simp add: ufo_particular_theory_sig.Γ_def)

interpretation inv_morph: particular_struct_bijection_1 ‹MorphImg φ src.Γ› ‹φ⁻¹› ‹TYPE('p2)
‹TYPE('p1)›
  apply (intro tgt.inj_morph_img_isomorphism[simplified tgt_Gamma_eq_Morph_img])

  using src.injection_to_ZF_exist
  by auto

lemma preserves_morphisms_src_tgt:
  fixes σ :: ‹'p1 ⇒ 'p3›
```

```
  assumes ‹particular_struct_morphism src.Γ Γ' σ›
  shows ‹particular_struct_morphism tgt.Γ Γ' (σ ∘ φ⁻¹)›
  apply (intro particular_struct_morphism_comp[OF _ assms])
  by (metis inv_is_bijective_morphism inv_morph.particular_struct_morphism_axioms
particular_struct_bijection_1.tgt_Gamma_eq_Morph_img particular_struct_bijection_iff_particula
tgt_Gamma_eq_Morph_img)

end

lemma isomorphisms_respect_morphisms:
  fixes σ   :: ‹'p₁ ⇒ 'p₃› and φ :: ‹'p₁ ⇒ 'p₂›
    and Γ   :: ‹('p₁,'q) particular_struct›
    and Γσ :: ‹('p₃,'q) particular_struct›
  assumes ‹φ ∈ BijMorphs1Γ,X› ‹σ ∈ MorphsΓ,Γσ›
  shows ‹σ ∘ (invMorph Γ φ) ∈ MorphsMorphImg φ Γ,Γσ›
proof -
  interpret I1: particular_struct_bijection_1 ‹Γ› ‹φ›
    using assms(1) by blast
  interpret I2: particular_struct_morphism ‹Γ› ‹Γσ› ‹σ›
    using assms by auto
  have S1: ‹I1.tgt.Γ = MorphImg φ Γ› using I1.tgt.Γ_simps by blast
  interpret I3: particular_struct_bijection_1 ‹MorphImg φ Γ› ‹I1.inv_morph›
    apply (simp only: I1.tgt.isomorphism_1_iff_inj[simplified S1]
          ; intro conjI ; simp?)
    using I1.src.injection_to_ZF_exist by blast
  have A: ‹I1.src.Γ = Γ› by auto
  have B: ‹I3.src.Γ = MorphImg φ Γ›
    by (intro particular_struct_eqI ; simp only: I3.src.Γ_simps)
  have C: ‹I3.tgt.endurants = I1.src.endurants›
    apply (auto simp: possible_worlds_sig.𝒫_def)
    subgoal for x w
      apply (intro bexI[of _ ‹w›] ; simp?)
      using particular_struct_bijection.tgt_is_morph_img by force
    subgoal for x w
      apply (intro bexI[of _ ‹w›] ; simp?)
      using particular_struct_bijection.tgt_is_morph_img by force
    done
  have D: ‹MorphImg I1.inv_morph (MorphImg φ Γ) = Γ›
    apply (intro particular_struct_eqI ; simp?)
    subgoal using particular_struct_bijection_iff_particular_struct_bijection_1
by force
    subgoal by (metis I1.inv_is_bijective_morphism particular_struct_bijection_iff_particular_
```

```
    subgoal using particular_struct_bijection.tgt_is_morph_img by fastforce
    subgoal by (metis I1.inv_is_bijective_morphism I3.particular_struct_bijection_axioms
isomorphism_tgt_unique)
    done
  show ‹?thesis›
    apply (intro morphs_I particular_struct_morphism_comp[of _ ‹Γ›]
            I2.particular_struct_morphism_axioms)
    using I3.particular_struct_morphism_axioms[simplified D] .
qed

context ufo_particular_theory
begin

lemma isomorphs_to_zf_non_empty[simp]: ‹BijMorphs1Γ,TYPE(ZF) ≠ ∅›
proof -
  obtain σ :: ‹'p ⇒ ZF› where ‹inj σ›  using injection_to_ZF_exist by blast
  have ‹particular_struct_bijection_1 Γ σ›
    apply simp
    using ‹inj σ› inj_on_id inj_on_subset by blast
  then have ‹σ ∈ BijMorphs1Γ,TYPE(ZF)› by blast
  then show ?thesis by blast
qed

end

end
```

C.3 Properties of UFO Particular Structures and Morphisms

C.3.1 Sub-Structures

```
theory SubStructures
  imports ParticularStructureMorphisms
beginabbreviation sub_structure_by ::
  ‹('i1,'q) particular_struct ⇒ ('i1 ⇒ 'i2) ⇒ ('i2,'q) particular_struct ⇒ bool›

  (‹_ ≪_ _› [74,1,74] 75) where
  ‹Γ1 ≪φ Γ2 ≡ particular_struct_injection Γ1 Γ2 φ›lemma sub_structure_by_refl:
  assumes ‹particular_struct Γ›
  shows ‹Γ ≪id Γ›
proof -
  interpret particular_struct Γ using assms .
```

```
  show ?thesis
    by (metis Γ_simps particular_struct_eqI particular_struct_bijection_1_def
          ufo_particular_theory.MorphImg_of_id
          ufo_particular_theory.id_is_isomorphism ufo_particular_theory_axioms)
qedlemma sub_structure_by_trans:
  assumes ‹Γ1 ≪φ1 Γ2› ‹Γ2 ≪φ2 Γ3›
  shows ‹Γ1 ≪φ2 ∘ φ1 Γ3›
  by (intro particular_struct_injection_comp[OF assms])abbreviation isomorphic_by
::
  ‹('i1,'q) particular_struct ⇒ ('i1 ⇒ 'i2) ⇒ ('i2,'q) particular_struct ⇒ bool›

  (‹_ ∼_ _› [74,1,74] 75) where
  ‹Γ1 ∼φ Γ2 ≡ particular_struct_bijection Γ1 Γ2 φ›lemma  isomorphic_by_imp_sub_structure[des
  assumes ‹Γ1 ∼φ Γ2›
  shows ‹Γ1 ≪φ Γ2›
  using assms
  by (simp add: particular_struct_bijection.axioms(1))lemma isomorphic_by_refl:

  assumes ‹particular_struct Γ›
  shows ‹Γ ∼id Γ›
proof -
  interpret particular_struct Γ using assms.
  show ?thesis
    by (metis Γ_simps id_is_a_permutation
          particular_struct_eqI particular_struct_permutation.axioms(2))
qedlemma isomorphic_by_sym[intro!]:
  assumes ‹Γ1 ∼φ Γ2›
  shows ‹Γ2 ∼particular_struct_morphism_sig.inv_morph Γ1 φ Γ1›
  using particular_struct_bijection.inv_is_bijective_morphism assms by metislemma
isomorphic_by_trans[trans]:
  assumes ‹Γ1 ∼φ1 Γ2› ‹Γ2 ∼φ2 Γ3›
  shows ‹Γ1 ∼φ2 ∘ φ1 Γ3›
  using assms particular_struct_bijection_comp by blast

lemma isomorphic_by_sub_structure_transf[simp]:
  assumes ‹Γ1 ∼φ Γ2›
  shows ‹Γ ≪φ ∘ φ1 Γ2 ∧ φ1 ` particulars Γ ⊆ particulars Γ1 ⟷ Γ ≪φ1 Γ1›
proof -
  interpret I1: particular_struct_bijection Γ1 Γ2 φ using assms .
  interpret I2: particular_struct_bijection_1 Γ1 φ
    using assms particular_struct_bijection_iff_particular_struct_bijection_1
    by blast
```

```
have A[simp]: ‹Γ2 = MorphImg φ Γ1›
  by (simp add: I1.tgt_is_morph_img)

show ?thesis
proof (cases ‹particular_struct Γ ∧ inj_on φ1 (particulars Γ) ∧ Γ ≪φ ∘ φ1 Γ2
                ∧ φ1 ‘ particulars Γ ⊆ particulars Γ1›
        ; (elim conjE)?)
  case False
  note AA = this
  then consider (C1) ‹¬ particular_struct Γ› |
                (C2) ‹¬ inj_on φ1 (particulars Γ)› |
                (C3) ‹¬ Γ ≪φ ∘ φ1 Γ2› |
                (C4) ‹Γ ≪φ ∘ φ1 Γ2› ‹¬ φ1 ‘ particulars Γ ⊆ particulars Γ1›
    by metis
  then show ?thesis
  proof cases
    case C1
    then show ?thesis
    by (simp only: particular_struct_injection_def
          particular_struct_morphism_def
          pre_particular_struct_morphism_def ; simp)
  next
    case C2
    then have B: ‹¬ inj_on (φ ∘ φ1) (particulars Γ)›
      using inj_on_imageI2 by blast
    show ?thesis
      by (simp only: particular_struct_injection_def
              particular_struct_injection_axioms_def C2 B ; simp)
  next
    case C3
    then have B: ‹¬ Γ ≪φ1 Γ1›
      using I1.particular_struct_injection_axioms
          particular_struct_injection_comp by metis
    show ?thesis
      by (intro iffI conjI ; (elim conjE)? ; (simp only: C3 B)?)
  next
    case C4
    then  interpret phi_phi1: particular_struct_injection Γ Γ2 ‹φ ∘ φ1› by simp
    have B: ‹(∃x. x ∈ phi_phi1.src.endurants ∧ φ1 x ∉ I1.src.endurants)›
      using C4 by blast
    show ?thesis
      by (simp add: C4 particular_struct_injection_def
```

```
            particular_struct_morphism_def
            pre_particular_struct_morphism_def
            pre_particular_struct_morphism_axioms_def
            B)
  qed
next
  assume assms1: ‹particular_struct Γ›
      ‹inj_on φ1 (particulars Γ)› ‹Γ ≪φ ∘ φ1 Γ2›
      ‹φ1 ` particulars Γ ⊆ particulars Γ1›
  interpret gamma: particular_struct Γ using assms1(1) .
  interpret phi_phi1: particular_struct_injection Γ Γ2 ‹φ ∘ φ1›
    using assms1(3) .

  have phi1_img[intro!]: ‹φ1 x ∈  particulars Γ1 › if ‹x ∈ particulars Γ› for x

      using assms1(4) that  by blast

  have B[simp]: ‹φ (φ1 x) = φ (φ1 y) ⟷ x = y› if ‹x ∈ phi_phi1.src.𝒫› ‹y ∈ phi_phi1.src.𝒫›
for x y
    using phi_phi1.morph_is_injective[THEN inj_onD,OF _ that,simplified] by metis
  have C[simp]: ‹φ1 x = φ1 y ⟷ x = y› if ‹x ∈ phi_phi1.src.𝒫› ‹y ∈ phi_phi1.src.𝒫›
for x y
    using assms1(2)[THEN inj_onD,OF _ that,simplified] by metis
  have D[simp]: ‹φ x = φ y ⟷ x = y› if ‹x ∈ φ1 ` phi_phi1.src.𝒫› ‹y ∈ φ1 ` phi_phi1.src.𝒫›
for x y
    using that apply (elim imageE ; hypsubst_thin)
    by simp
  note simps = B C D
  have E: ‹φ1 x = I2.inv_morph ((φ ∘ φ1) x)› if ‹x ∈ phi_phi1.src.𝒫› for x
    using that assms assms1 by auto

  { fix x y z q
    note phi_phi1.morph_reflects_inherence[of x y]
         phi_phi1.morph_does_not_add_bearers[of x z]
         phi_phi1.morph_reflects_towardness[of x y]
         phi_phi1.morph_does_not_add_towards[of x z]
         phi_phi1.morph_reflects_quale_assoc[of x q] }
  note R1 = this
  { fix x y z q
    note I1.morph_reflects_inherence[of ‹φ1 x› ‹φ1 y›]
         I1.morph_does_not_add_bearers[of ‹φ1 x› z]
         I1.morph_reflects_towardness[of ‹φ1 x› ‹φ1 y›]
```

```
        I1.morph_does_not_add_towards[of ‹φ1 x› z]
        I1.morph_reflects_quale_assoc[of ‹φ1 x› q] }
  note R2 = this

  interpret phi1: pre_particular_struct_morphism Γ Γ1 φ1
    apply (unfold_locales ; simp? ; (simp only: E)?)
    subgoal G1  using phi_phi1.quality_space_subset by auto
    subgoal G2  using assms1 by (simp ; blast)
    subgoal G3 for x y
      using phi_phi1.morph_preserves_particulars phi_phi1.morph_reflects_inherence
by auto
    subgoal G4 for x z
      by (metis (no_types, lifting) E G2 I1.inv_morph_morph I1.morph_reflects_inherence
I1.src.endurantI2 I1.src.moment_non_migration phi_phi1.morph_does_not_add_bearers
phi_phi1.morph_preserves_particulars)
    subgoal G5 for x y
      using phi_phi1.morph_preserves_particulars phi_phi1.morph_reflects_towardness
by auto
    subgoal G6 for x y
      by (metis E G2 I1.inv_morph_morph I1.morph_reflects_towardness
            I1.src.towardness_scopeE I2.morph_is_injective
            phi_phi1.morph_does_not_add_towards
            phi_phi1.morph_preserves_particulars inj_onD)
    subgoal for x q
      using G2 by auto
    done

  have phi1_w_c: ‹phi1.world_corresp w w2 ⟷ phi_phi1.world_corresp w (φ ‘ w2)
∧ w2 ∈ I1.src.𝒲› for w w2
    apply (simp add: particular_struct_morphism_sig.world_corresp_def
        ; intro iffI conjI impI ballI ; (elim imageE conjE disjE exE)? ; simp?)
    subgoal G1 by blast
    subgoal G2 for x y by (metis I2.morph_is_injective inj_onD phi1.tgt.𝒫_I phi1_img)
    using I2.src_world_corresp_image by blast

  have R4: ‹phi1.inv_morph ‘ (w1 ∩ φ1 ‘ phi_phi1.src.𝒫) ∈ phi_phi1.src.𝒲›
    if as: ‹w1 ∈ phi1.tgt.𝒲› for w1
  proof -
    obtain w2 where AA: ‹I1.world_corresp w1 w2›
      using as I1.morph_worlds_correspond_src_tgt by metis
    obtain BB: ‹w2 ∈ I1.tgt.𝒲›
              ‹⋀x. x ∈ phi1.tgt.𝒫 ⟹ x ∈ w1 ⟷ φ x ∈ w2›
```

```
    using I1.world_corresp_E AA by metis
  obtain w3 where CC: ‹phi_phi1.world_corresp w3 w2›
    using BB phi_phi1.morph_worlds_correspond_tgt_src by metis
  obtain DD: ‹w3 ∈ phi_phi1.src.𝒲›
      ‹⋀x. x ∈ phi_phi1.src.𝒫 ⟹ x ∈ w3 ⟷φ (φ1 x) ∈ w2›
    using phi_phi1.world_corresp_E[OF CC] comp_apply by metis
  { fix x
    assume ‹x ∈ w3›
    then have EE: ‹x ∈ phi_phi1.src.𝒫› using DD(1) by blast
    then have FF: ‹φ (φ1 x) ∈ w2 ⟷ x ∈ w3› using DD(2) by simp
    have ‹φ1 x ∈ phi1.tgt.𝒫› using EE by blast
    then have ‹φ1 x ∈ w1 ⟷ x ∈ w3› using BB(2) FF by metis }
  note simp1[simp] = this

  { fix x
    assume EE: ‹x ∈ phi_phi1.src.𝒫›
    then have FF: ‹φ (φ1 x) ∈ w2 ⟷ x ∈ w3› using DD(2) by simp
    have ‹φ1 x ∈ phi1.tgt.𝒫› using EE by blast
    then have ‹φ1 x ∈ w1 ⟷ x ∈ w3› using BB(2) FF by metis }
  note simp2[simp] = this

  have GG: ‹φ1 ` w3 = w1 ∩ φ1 ` phi_phi1.src.𝒫›
    apply (auto  ; (intro imageI)? ; simp?)
    using DD(1) by blast
  have HH: ‹phi1.inv_morph (φ1 x) = x› if ‹x ∈ phi_phi1.src.𝒫› for x
    by (simp add: assms1(2) phi1.inv_morph_def that)
  have II: ‹phi1.inv_morph ` φ1 ` w3 = w3›
    supply T = subsetD[OF phi_phi1.src.worlds_are_made_of_particulars,OF DD(1)]
    using T by (auto simp: image_iff HH)
  show ?thesis
    by (simp add: GG[symmetric] II DD(1))
qed

have ex_ex_1: ‹(∃y1∈ws. phi1.src_inheres_in y1 x ∧ φ y = (φ ∘ φ1) y1)
    ⟷
      (∃y1∈ws. phi1.src_inheres_in y1 x ∧ y = φ1 y1)›
  if ‹y ∈ wt› ‹wt ∈ phi1.tgt.𝒲› for y wt ws x
proof -
  have AA[intro!,simp]: ‹y ∈ phi1.tgt.𝒫› using that by blast

  show ?thesis
    apply (intro iffI ; elim bexE conjE)
```

```
    subgoal for y1
      apply (intro bexI[of _ y1] conjI)
      subgoal using that AA by simp
      subgoal using that AA
        apply simp
        by (meson AA I2.morph_is_injective inj_onD phi1_img phi_phi1.src.inherence_sc
      subgoal using that AA by simp
      done
    subgoal for y1
      by (intro bexI[of _ y1] conjI ; simp add: that)
    done
qed


interpret phi1: particular_struct_morphism Γ Γ1 φ1
  apply (unfold_locales ; simp only: phi1_w_c simp_thms)
  subgoal G1 premises P for w1
    using phi_phi1.morph_worlds_correspond_src_tgt[OF P]
    apply (elim exE)
    subgoal premises Q for w2
      apply (rule phi_phi1.world_corresp_E[OF Q])
      subgoal premises T
        apply (intro exI[of _ ‹I1.inv_morph ` w2›] conjI)
        subgoal using Q T(2) by simp
        using T(2) by (simp add: I1.world_preserve_inv_img1)
      done
    done
  subgoal G2 premises P for w1
    apply (intro exI[of _ ‹phi1.inv_morph ` (w1 ∩ φ1 ` phi_phi1.src.𝒫)›] conjI)

    apply (intro phi1_w_c[THEN iffD1,THEN conjunct1]
            phi1.world_corresp_I P iffI ; simp)
    using P apply auto
    subgoal G2_1 using R4 by blast
    subgoal G2_2 by (simp add: assms1(2) phi1.inv_morph_def)
    by (metis IntI assms1(2) image_iff inv_into_f_f phi1.inv_morph_def)
  done

interpret phi1: particular_struct_injection Γ Γ1 φ1
  using assms1(2) by unfold_locales

show ?thesis
```

```
      apply (intro conjI iffI ; (elim conjE)?)
      subgoal using phi1.particular_struct_injection_axioms by blast
      subgoal using phi_phi1.particular_struct_injection_axioms by blast
      by blast
  qed
qed


lemma finite_card_sub_structure_by_lteq:
  fixes Γ1 :: ‹('p1,'q) particular_struct› and
        Γ2 :: ‹('p2,'q) particular_struct›
  assumes ‹finite (particulars Γ2)› ‹Γ1 ≪φ Γ2›
  shows ‹card (particulars Γ1) ≤ card (particulars Γ2)›
proof -
  interpret phi: particular_struct_injection Γ1 Γ2 φ ‹TYPE('p1)› ‹TYPE('p2)› ‹TYPE('q)›

    using assms by simp
  have A: ‹finite (particulars Γ1)›
    using finite_image_iff[OF phi.morph_is_injective,THEN iffD1,
          OF finite_subset,of ‹particulars Γ2›] assms(1) by blast
  show ?thesis
    using A assms(1) phi.morph_is_injective
    by (metis card_image card_mono phi.morph_image_def phi.morph_scope)
qed

lemma finite_card_substruct_lt:
  fixes Γ1 :: ‹('p1,'q) particular_struct› and
        Γ2 :: ‹('p2,'q) particular_struct›
  assumes ‹finite (particulars Γ2)› ‹Γ1 ≪φ Γ2› ‹∀φ. ¬ Γ2 ≪φ Γ1›
  shows ‹card (particulars Γ1) < card (particulars Γ2)›
proof -
  interpret phi: particular_struct_injection Γ1 Γ2 φ ‹TYPE('p1)› ‹TYPE('p2)› ‹TYPE('q)›

    using assms by simp
  have A: ‹finite (particulars Γ1)›
    using finite_image_iff[OF phi.morph_is_injective,THEN iffD1,
          OF finite_subset,of ‹particulars Γ2›] assms(1) by blast
  have B: ‹card (particulars Γ1) ≤ card (particulars Γ2)›
    using A assms(1) phi.morph_is_injective
    by (metis card_image card_mono phi.morph_image_def phi.morph_scope)

  have C: ‹card (particulars Γ1) ≠ card (particulars Γ2)›
```

```
  proof
    assume AA: ‹card (particulars Γ1) = card (particulars Γ2)›
    then have ‹bij_betw φ (particulars Γ1) (particulars Γ2)›
      using assms(1) A phi.morph_is_injective
      by (metis B card_image card_seteq phi.morph_bij_on_img phi.morph_image_def
phi.morph_scope)
    then interpret phi: particular_struct_bijection Γ1 Γ2 φ ‹TYPE('p1)› ‹TYPE('p2)›
‹TYPE('q)›
      apply unfold_locales
      by (simp add: bij_betw_def)
    interpret phi_inv:  particular_struct_bijection Γ2 Γ1 ‹phi.inv_morph› ‹TYPE('p2)›
‹TYPE('p1)› ‹TYPE('q)›
      by simp
    show False using assms
      phi_inv.particular_struct_injection_axioms by simp
  qed
  then show ?thesis using B by auto
qed

lemma finite_card_sub_structure_by_finite:
  fixes Γ1 :: ‹('p1,'q) particular_struct› and
        Γ2 :: ‹('p2,'q) particular_struct›
  assumes ‹finite (particulars Γ2)› ‹Γ1 ≪φ Γ2›
  shows ‹finite (particulars Γ1)›
proof -
  interpret phi: particular_struct_injection Γ1 Γ2 φ using assms by simp
  show ?thesis
    using assms phi.morph_is_injective
          finite_image_iff finite_subset phi.morph_scope
    by fastforce
qed

lemma isomorphic_by_sub_structure_transf_inv:
  assumes ‹Γ1 ∼φ Γ2›
  shows ‹Γ ≪invMorph Γ1 φ ∘ φ1 Γ1 ∧ φ1 ` particulars Γ ⊆ particulars Γ2 ⟷
         Γ ≪φ1 Γ2›
  using isomorphic_by_sub_structure_transf[OF isomorphic_by_sym] assms by metis

lemma sub_structure_by_finite_weak_antisym[intro]:
  fixes Γ1 :: ‹('p1,'q) particular_struct› and
        Γ2 :: ‹('p2,'q) particular_struct›
  assumes ‹Γ1 ≪φ1 Γ2› ‹Γ2 ≪φ2 Γ1› ‹finite (particulars Γ1)›
```

```
  shows ‹Γ1 ∼φ1 Γ2›
proof -
  interpret phi1: particular_struct_injection Γ1 Γ2 φ1 using assms(1) .
  interpret phi2: particular_struct_injection Γ2 Γ1 φ2 using assms(2) .
  have A: ‹inj_on (φ2 ∘ φ1) phi1.src.𝒫›
    using comp_inj_on phi1.morph_is_injective phi2.morph_is_injective
      phi1.morph_scope
    by (metis phi1.morph_image_def subset_inj_on)
  then have A1: ‹inj_on φ1 phi1.src.𝒫› by blast
  have A2: ‹inj_on φ2 (φ1 ` phi1.src.𝒫)›
    using A A1 inj_on_imageI by blast
  have B: ‹f ` A = B› if
      ‹inj_on f A› ‹finite A› ‹f ` A ⊆ B›
      ‹inj_on g B› ‹g ` B ⊆ A›
    for A :: ‹'p1 set› and B :: ‹'p2 set› and f g
    using that
    by (metis card_bij_eq card_image card_subset_eq inj_on_finite)
  have bij: ‹bij_betw φ1 phi1.src.𝒫 phi1.tgt.𝒫›
    apply (simp add: bij_betw_def)
    apply (intro B[of _ _ _ φ2] ; simp?)
    using assms(3) by blast+
  then show ?thesis
    apply (unfold_locales)
    by (simp add: bij_betw_def)
qed

end
theory MorphismImage
  imports ParticularStructureMorphisms
begin

context inherence_base
begin

lemma moments_eq_particulars_minus_subst: ‹𝒩 = 𝒫 - 𝒮› by blast

lemma substantials_eq_particulars_minus_moments: ‹𝒮 = 𝒫 - 𝒩› by blast

end

context particular_struct_morphism
begin
```

```
abbreviation ‹𝒲φ ≡ ps_worlds (MorphImg φ Γ1)›
abbreviation img_inheres_in (infix ‹◁φ› 75)
  where ‹(◁φ) ≡ ps_inheres_in (MorphImg φ Γ1)›
abbreviation img_assoc_quale (infix ‹⇝φ› 75)
  where ‹(⇝φ) ≡ ps_assoc_quale (MorphImg φ Γ1)›
abbreviation img_towards (infix ‹⟶φ› 75)
  where ‹(⟶φ) ≡ ps_towards (MorphImg φ Γ1)›

interpretation img: ufo_particular_theory_sig ‹𝒲φ› ‹(◁φ)› src.𝒬𝒮 ‹(⇝φ)› ‹(⟶φ)›
.

lemma morph_image_particulars: ‹img.𝒫 = φ ` src.𝒫›
  by (auto simp: possible_worlds_sig.𝒫_def)

lemma morph_image_worlds_tgt:
  ‹𝒲φ = { w ∩ φ ` src.𝒫 | w . w ∈ tgt.𝒲}›

proof (auto)
  fix ws
  assume A[simp]: ‹ws ∈ src.𝒲›
  then obtain wt where B: ‹ws ⇔ wt› using morph_worlds_correspond_src_tgt by metis
  obtain C[simp]: ‹wt ∈ tgt.𝒲› ‹⋀x. x ∈ src.𝒫 ⟹ x ∈ ws ⟷ φ x ∈ wt›
    using world_corresp_E[OF B] by metis
  have ‹φ ` ws = wt ∩ φ ` src.endurants›
    apply auto
    using A C(2) by blast+
  then show ‹∃wt. φ ` ws = wt ∩ φ ` src.endurants ∧ wt ∈ tgt.𝒲›
    using C(1) by blast
next
  fix wt
  assume A[simp]: ‹wt ∈ tgt.𝒲›
  then obtain ws where B: ‹ws ⇔ wt› using morph_worlds_correspond_tgt_src by metis
  obtain C[simp]: ‹ws ∈ src.𝒲› ‹⋀x. x ∈ src.𝒫 ⟹ x ∈ ws ⟷ φ x ∈ wt›
    using world_corresp_E[OF B] by metis
  have ‹wt ∩ φ ` src.endurants = φ ` ws ∧ ws ∈ src.𝒲›
    apply auto
    using A C by blast+
  then show ‹∃ws. wt ∩ φ ` src.endurants = φ ` ws ∧ ws ∈ src.𝒲› by blast
qed

lemma morph_image_worlds_src:‹𝒲φ = ((`) φ) ` src.𝒲›
```

```
  by auto

lemma morph_image_ed_tgt: ‹img.ed x y ⟷ x ∈ φ ‘ src.𝒫 ∧ y ∈ φ ‘ src.𝒫 ∧ tgt.ed
x y›
proof (intro iffI conjI ; (elim conjE)?)
  assume A: ‹possible_worlds_sig.ed (ps_worlds (MorphImg φ Γ1)) x y›
  obtain B[simp]: ‹x ∈ φ ‘ src.𝒫› ‹y ∈ φ ‘ src.𝒫› and C: ‹⋀w. w ∈ ps_worlds (MorphImg
φ Γ1) ⟹ x ∈ w ⟹ y ∈ w›
    using possible_worlds_sig.edE[OF A,simplified morph_image_particulars ] by blast
  show D: ‹x ∈ φ ‘ src.𝒫› ‹y ∈ φ ‘ src.𝒫› by simp_all
  have E: ‹y ∈ w1› if ‹w1 = w2 ∩ φ ‘ src.𝒫› ‹w2 ∈ tgt.𝒲› ‹x ∈ w1› for w1 w2
    using C[simplified morph_image_worlds_tgt, OF CollectI] that by metis
  obtain F[simp]: ‹x ∈ tgt.𝒫› ‹y ∈ tgt.𝒫› using D by blast
  show ‹tgt.ed x y›
  proof (intro tgt.edI F)
    fix wt
    assume AA[simp]: ‹wt ∈ tgt.𝒲› ‹x ∈ wt›
    show ‹y ∈ wt›
      using E[of ‹wt ∩ φ ‘ src.𝒫› wt,THEN IntD1,simplified] by simp
  qed
next
  assume A: ‹x ∈ φ ‘ src.𝒫› ‹y ∈ φ ‘ src.𝒫› ‹tgt.ed x y›
  then obtain xs ys where B[simp]: ‹x = φ xs› ‹y = φ ys› ‹xs ∈ src.𝒫› ‹ys ∈ src.𝒫›
by blast
  obtain C[simp]: ‹φ xs ∈ tgt.𝒫› ‹φ ys ∈ tgt.𝒫› using A B by blast
  have D: ‹φ ys ∈ w› if ‹w ∈ tgt.𝒲› ‹φ xs ∈ w› for w
    using tgt.edE[OF A(3),simplified] that by blast
  obtain E: ‹φ xs ∈ particulars (MorphImg φ Γ1)› ‹φ ys ∈ particulars (MorphImg φ
Γ1)›
    by (simp only: morph_image_particulars A B(1,2)[symmetric])

  show ‹possible_worlds_sig.ed (ps_worlds (MorphImg φ Γ1)) x y›
  proof (simp only: B ; intro possible_worlds_sig.edI E ; simp ; elim exE conjE ;
simp)
    fix w :: "'p2 set" and wa :: "'p1 set"
    assume a1: "φ xs ∈ φ ‘ wa"
    assume a2: "w = φ ‘ wa"
    assume "wa ∈ src.𝒲"
    then obtain PP :: "'p1 set ⇒ 'p2 set" where
      f3: "wa ⇔ PP wa"
      by (meson morph_worlds_correspond_src_tgt)
    have "x ∈ w"
```

```
      using a2 a1 by fastforce
    then show "φ y_s ∈ φ ` wa"
      using f3 a2 A(3) B(2) B(4) by blast
  qed
qed

lemma morph_image_edI:
  assumes ‹x ∈ φ ` src.𝒫› ‹y ∈ φ ` src.𝒫› ‹tgt.ed x y›
  shows ‹img.ed x y›
  using assms by (simp only: morph_image_ed_tgt)

lemma morph_image_edD:
  assumes ‹img.ed x y›
  shows ‹tgt.ed x y›
  using assms by (simp only: morph_image_ed_tgt)

lemma morph_image_edE:
  assumes ‹img.ed x y›
  obtains ‹x ∈ φ ` src.𝒫› ‹y ∈ φ ` src.𝒫› ‹tgt.ed x y›
  using assms by (simp only: morph_image_ed_tgt)

lemma morph_image_indep_iff: ‹img.indep x y ⟷ x ∈ φ ` src.𝒫 ∧ y ∈ φ ` src.𝒫
∧ tgt.indep x y›
proof (intro iffI conjI ; (elim conjE)?)
  assume A: ‹img.indep x y›
  obtain B: ‹x ∈ φ ` src.𝒫› ‹y ∈ φ ` src.𝒫› ‹¬ img.ed x y› ‹¬ img.ed y x›
    using img.indepE[OF A,simplified morph_image_particulars] by metis
  then show ‹x ∈ φ ` src.𝒫› ‹y ∈ φ ` src.𝒫› by simp+
  obtain C: ‹x ∈ tgt.𝒫› ‹y ∈ tgt.𝒫› using B(1,2) by blast
  have D: ‹¬ tgt.ed x y› ‹¬ tgt.ed y x›
    using B(3,4)[THEN notE,OF morph_image_edI] B(2,1) by metis+
  then show ‹tgt.indep x y›
    by (intro tgt.indepI C D)
next
  assume A: ‹x ∈ φ ` src.𝒫› ‹y ∈ φ ` src.𝒫› and B: ‹tgt.indep x y›
  show ‹img.indep x y›
    apply (intro img.indepI notI ; (elim notE morph_image_edE)?
          ; (simp only: morph_image_particulars A) ; simp)
    using B tgt.indepE by blast+
qed

lemma morph_image_indepI:
```

```
  assumes ‹x ∈ φ ‘ src.𝒫› ‹y ∈ φ ‘ src.𝒫› ‹tgt.indep x y›
  shows ‹img.indep x y›
  using assms by (simp only: morph_image_indep_iff)

lemma morph_image_indepE:
  assumes ‹img.indep x y›
  obtains ‹x ∈ φ ‘ src.𝒫› ‹y ∈ φ ‘ src.𝒫› ‹tgt.indep x y›
  using assms by (simp only: morph_image_indep_iff)

lemma morph_image_indepD:
  assumes ‹img.indep x y›
  shows ‹x ∈ φ ‘ src.𝒫› ‹y ∈ φ ‘ src.𝒫› ‹tgt.indep x y›
  using assms by (simp only: morph_image_indep_iff)+

lemma morph_image_inheres_in: ‹x ◁φ y ⟷ x ∈ φ ‘ src.𝒫 ∧ y ∈ φ ‘ src.𝒫 ∧ x
◁t y›
  apply (auto)
  by blast

lemma morph_image_inheres_in_I:
  assumes ‹x ∈ φ ‘ src.𝒫› ‹y ∈ φ ‘ src.𝒫› ‹x ◁t y›
  shows ‹x ◁φ y›
  using assms by (simp only: morph_image_inheres_in)

lemma morph_image_inheres_in_E:
  assumes ‹x ◁φ y›
  obtains ‹x ∈ φ ‘ src.𝒫› ‹y ∈ φ ‘ src.𝒫› ‹x ◁t y›
  using assms by (simp only: morph_image_inheres_in)

lemma morph_image_inheres_in_D:
  assumes ‹x ◁φ y›
  shows ‹x ∈ φ ‘ src.𝒫› ‹y ∈ φ ‘ src.𝒫› ‹x ◁t y›
  using assms by (simp only: morph_image_inheres_in)+

lemma morph_image_trans_inheres_in: ‹(◁φ)++ x y ⟷ x ∈ φ ‘ src.𝒫 ∧ y ∈ φ ‘
src.𝒫 ∧ (◁t)++ x y›
proof (intro iffI conjI ; (elim conjE)?)
  assume A: ‹(◁φ)++ x y›
  show G1: ‹x ∈ φ ‘ src.𝒫› ‹y ∈ φ ‘ src.𝒫›
  proof -
    obtain x1 where ‹x ◁φ x1›
      using A tranclp_induct by metis
```

```
    obtain y1 where ‹y1 ◁φ y›
      using A by (metis tranclp.cases)
    then show ‹x ∈ φ ` src.𝒫› ‹y ∈ φ ` src.𝒫›
      using ‹x ◁φ x1› morph_image_inheres_in_D(1,2) by metis+
  qed
  from A
  show ‹(◁t)++ x y›
  proof (induct)
    fix y
    assume AA: ‹x ◁φ y›
    show ‹(◁t)++ x y›
      using AA morph_image_inheres_in_D(3) tranclp.intros(1) by metis
  next
    fix y z
    assume AA:  ‹y ◁φ z› ‹(◁t)++ x y›
    then show ‹(◁t)++ x z›
      using AA morph_image_inheres_in_D(3) tranclp.intros(2) by metis
  qed
next
  assume A: ‹x ∈ φ ` src.𝒫› ‹y ∈ φ ` src.𝒫› and B: ‹(◁t)++ x y›
  from B
  show ‹(◁φ)++ x y›
  proof (induct)
    fix y
    assume AA: ‹x ◁t y›
    then show ‹(◁φ)++ x y›
      using A morph_image_inheres_in_I tranclp.intros(1)
      by (metis image_iff morph_does_not_add_bearers)
  next
    fix y z
    assume AA: ‹(◁t)++ x y› ‹y ◁t z› ‹(◁φ)++ x y›
    have BB: ‹y ∈ φ ` src.𝒫›
      by (metis AA(3) converse_tranclp_induct morph_image_inheres_in)
    then have CC: ‹z ∈ φ ` src.𝒫› using morph_does_not_add_bearers AA(2) by blast
    then have BB: ‹y ◁φ z›
      using morph_image_inheres_in_I BB AA(2) by blast
    then show ‹(◁φ)++ x z›
      using AA(3) tranclp.intros(2) by metis
  qed
qed

lemma morph_image_moments: ‹img.ℳ = φ ` src.ℳ›
```

by *(auto simp: inherence_sig.$\mathcal{M}$_def)*

lemma *morph_image_assoc_quale:* ‹$x \rightsquigarrow_\varphi q \longleftrightarrow x \in \varphi$ ‘ *src.$\mathcal{P}$* $\wedge$ $x \rightsquigarrow_t q$›
apply *(auto)*
subgoal by *(meson imageI src.assoc_quale_scopeD(1))*
subgoal using *morph_reflects_quale_assoc src.assoc_quale_scopeD(1)* **by** *blast*
using *morph_reflects_quale_assoc* **by** *blast*

lemma *morph_image_assoc_quale_I:*
assumes ‹$x \in \varphi$ ‘ *src.$\mathcal{P}$*› ‹$x \rightsquigarrow_t q$›
shows ‹$x \rightsquigarrow_\varphi q$›
using *assms* **by** *(simp only: morph_image_assoc_quale)*

lemma *morph_image_assoc_quale_E:*
assumes‹$x \rightsquigarrow_\varphi q$›
obtains ‹$x \in \varphi$ ‘ *src.$\mathcal{P}$*› ‹$x \rightsquigarrow_t q$›
using *assms* **by** *(simp only: morph_image_assoc_quale)*

lemma *morph_image_assoc_quale_D:*
assumes‹$x \rightsquigarrow_\varphi q$›
shows ‹$x \in \varphi$ ‘ *src.$\mathcal{P}$*› ‹$x \rightsquigarrow_t q$›
using *assms* **by** *(simp only: morph_image_assoc_quale)+*

lemma *morph_image_towards:* ‹$x \longrightarrow_\varphi y \longleftrightarrow x \in \varphi$ ‘ *src.$\mathcal{P}$* $\wedge$ $y \in \varphi$ ‘ *src.$\mathcal{P}$* $\wedge$ $x \longrightarrow_t y$›
apply *(auto)*
subgoal *G1* **using** *morph_reflects_towardness* **by** *blast*
subgoal *G2* **by** *blast*
done

lemma *morph_image_towards_I:*
assumes ‹$x \in \varphi$ ‘ *src.$\mathcal{P}$*› ‹$y \in \varphi$ ‘ *src.$\mathcal{P}$*› ‹$x \longrightarrow_t y$›
shows ‹$x \longrightarrow_\varphi y$›
using *assms* **by** *(simp only: morph_image_towards)*

lemma *morph_image_towards_E:*
assumes ‹$x \longrightarrow_\varphi y$›
obtains ‹$x \in \varphi$ ‘ *src.$\mathcal{P}$*› ‹$y \in \varphi$ ‘ *src.$\mathcal{P}$*› ‹$x \longrightarrow_t y$›
using *assms* **by** *(simp only: morph_image_towards)*

lemma *morph_image_towards_D:*

```
  assumes ‹x ⟶φ y›
  shows ‹x ∈ φ ` src.𝒫› ‹y ∈ φ ` src.𝒫› ‹x ⟶t y›
  using assms by (simp only: morph_image_towards)+

interpretation img: possible_worlds ‹𝒲φ›
  apply (unfold_locales)
  subgoal G1 by (simp add: tgt.injection_to_ZF_exist)
  subgoal G2 by (simp add: src.at_least_one_possible_world)
proof(simp only: morph_image_particulars ; simp only: morph_image_worlds_src
        ; auto)
  fix xa :: 'p1
  assume a1: "xa ∈ src.endurants"
  then obtain PP :: "'p1 ⇒ 'p1 set" where
    f2: "PP xa ∈ src.𝒲 ∧ xa ∉ PP xa"
    by (meson src.particulars_do_not_exist_in_some_world)
  then obtain PPa :: "'p1 set ⇒ 'p2 set" where
    "PP xa ⇔ PPa (PP xa)"
    by (meson morph_worlds_correspond_src_tgt)
  then have f3: "PP xa ∈ src.𝒲 ∧ PPa (PP xa) ∈ tgt.𝒲 ∧ (∀p. p ∉ src.endurants
∨ (p ∈ PP xa) = (φ p ∈ PPa (PP xa)))"
    by blast
  have "∀p f P. (p::'p2) ∉ f ` P ∨ (∃pa. p = f (pa::'p1) ∧ pa ∈ P)"
    by blast
  then show "∃P∈src.𝒲. φ xa ∉ φ ` P"
    using f3 f2 a1 by (metis possible_worlds_sig.𝒫_I)
qed



interpretation img: inherence_base ‹𝒲φ› ‹(◁φ)›
  apply (unfold_locales ; (simp only: morph_image_ed_tgt morph_image_particulars)?
        ; elim morph_image_inheres_in_E
        ; (intro conjI)? ; simp?)
  subgoal by (simp add: tgt.inherence_imp_ed)
  using tgt.moment_non_migration by blast

lemma morph_image_substantials: ‹img.𝒮 = φ ` src.𝒮›
  apply (simp only: inherence_sig.𝒮_def morph_image_particulars morph_image_moments)
  apply auto
  by (metis morph_preserves_moments_simp morph_preserves_moments)

interpretation img: noetherian_inherence ‹𝒲φ› ‹(◁φ)›
```

```
  apply (unfold_locales ; simp)
  apply (rule wf_subset[to_pred,OF tgt.inherence_is_noetherian] ; simp ; rule)
  apply (elim exE conjE )
  using morph_preserves_inherence_1 by metis



interpretation img: inherence ‹𝒲φ› ‹(◁φ)›
  apply (unfold_locales ; simp)
  apply (rule wf_subset[to_pred,OF tgt.inherence_is_wf] ; rule )
  apply (elim exE conjE )
  using morph_preserves_inherence_1 by metis

interpretation img: quality_space ‹ps_quality_spaces (MorphImg φ Γ1)›
  by (simp ; unfold_locales)

interpretation img:  qualified_particulars ‹𝒲φ› ‹(◁φ)›  ‹src.𝒬𝒮› ‹(⇝φ)›
  supply   ParticularStructure.particular_struct.simps[simp del]
  apply (unfold_locales ; (simp only: morph_image_moments morph_image_assoc_quale
; simp )?
          ; (elim conjE)?)
  subgoal G1 by (metis image_eqI  morph_image_def morph_image_iff  morph_reflects_quale_assoc
src.assoc_quale_scope)
  subgoal G1 by (simp add: tgt.assoc_quale_unique)
  subgoal G3 premises P for w x1 x2 y q1 q2 Q
    using P
    by (smt image_iff pre_particular_struct_morphism.morph_preserves_inherence_1

        pre_particular_struct_morphism.morph_reflects_inherence
        pre_particular_struct_morphism.morph_reflects_quale_assoc
        pre_particular_struct_morphism_axioms src.𝒫_I src.endurantI2
        src.quality_moment_unique_by_quality_space)
  subgoal G4
    by (meson morph_reflects_quale_assoc src.assoc_quale_scopeD(1) src.every_quality_space_is_
  subgoal G5
    using src.quale_determines_moment
    by (metis morph_preserves_inherence_1 tgt.quale_determines_moment)
  done

lemma rtranclp_iff_tranclp: ‹R** x y ⟷ x = y ∨ (R++ x y)› for R x y
  apply auto
  by (meson rtranclpD)
```

```
lemma morph_image_ultimate_bearer:
  assumes ‹x ∈ φ ‘ src.𝒫›
  shows ‹img.ultimateBearer x = tgt.ultimateBearer x›
  using assms
proof (induct x rule: wfP_induct[OF tgt.inherence_is_noetherian])
  fix x
  assume A: ‹x ∈ φ ‘ src.𝒫›
     and B: ‹∀y. (◁t)^-1-1 y x ⟶ y ∈ φ ‘ src.endurants ⟶ img.ultimateBearer
y = tgt.ultimateBearer y›
  have C: ‹img.ultimateBearer y = tgt.ultimateBearer y›
    if ‹y ∈ φ ‘ src.𝒫› ‹x ◁t y› for y using B that by blast
  have D: ‹x ∈ tgt.𝒫› using A by blast

  have ‹φ ‘ src.ℳ ⊆ tgt.ℳ› by blast
  then have E[simp]: ‹tgt.ultimateBearer x ∉ φ ‘ src.ℳ›
    using tgt.ultimate_bearer_is_not_a_moment
    using D by blast

  have c_moment: ‹img.ultimateBearer x = tgt.ultimateBearer x› if as: ‹x ∈ tgt.ℳ›

  proof -
    obtain y where ‹x ◁t y› using as by blast
    then have ‹y ∈ φ ‘ src.𝒫› using A morph_does_not_add_bearers by blast
    then have AA: ‹img.ultimateBearer y = tgt.ultimateBearer y› using C ‹x ◁t y›
by blast
    then obtain BB: ‹tgt.ultimateBearer y ∉ img.ℳ› ‹(◁φ)** y (tgt.ultimateBearer
y)›
      using AA[simplified img.ultimate_bearer_eq_simp]
      by (metis ‹y ∈ φ ‘ src.endurants› img.noetherian_inherence_axioms
              img.ultimate_bearer_is_not_a_moment
              morph_image_particulars noetherian_inherence.ultimate_bearer_eq_simp)
    have ‹x ◁φ y› using A ‹y ∈ φ ‘ src.𝒫› ‹x ◁t y› morph_image_inheres_in_I by blast
    then have CC: ‹(◁φ)** x (tgt.ultimateBearer y)›
      using BB(2) converse_rtranclp_into_rtranclp by metis
    have ‹x ≠ y› using ‹x ◁φ y› using img.inherence_irrefl by blast
    then have DD: ‹(◁φ)++ x (tgt.ultimateBearer y)› using CC
      by (metis ‹x ◁t y› rtranclp_iff_tranclp tgt.inherence_scopeE tgt.ultimate_bearer_is_

    then have EE: ‹x ≠ tgt.ultimateBearer y› using img.inherence_is_acyclic by blast
    have FF: ‹img.ultimateBearer x = tgt.ultimateBearer y›
      by (metis A AA CC ‹y ∈ φ ‘ src.endurants› img.ultimate_bearer_eq_simp morph_image_p
```

```
    obtain GG: ‹tgt.ultimateBearer y ∉ tgt.𝓜› ‹(◁t)** y (tgt.ultimateBearer y)›
      by (meson ‹x ◁t y› tgt.endurantI2 tgt.ultimate_bearer_eq_simp tgt.ultimate_bearer_is_not_
    have HH: ‹(◁t)** x (tgt.ultimateBearer y)›
      using ‹x ◁t y› converse_rtranclp_into_rtranclp GG(2) by metis
    then have II: ‹tgt.ultimateBearer x = tgt.ultimateBearer y›
      using D ‹x ◁t y› tgt.ultimate_bearer_eq_simp by auto
    show ?thesis
      by (simp only: FF II)
  qed

  have c_substantial: ‹img.ultimateBearer x = tgt.ultimateBearer x› if as: ‹x ∈ tgt.𝓢›
  proof -
    have AA: ‹x ∈ img.𝓢› using as morph_image_substantials A
      by auto
    then obtain BB: ‹x ∉ img.𝓜› ‹x ∉ tgt.𝓜› using as by blast
    obtain CC: ‹(◁φ)** x x› ‹(◁t)** x x› by blast
    have DD1: ‹img.ultimateBearer x = x›
        using AA img.ultimate_bearer_eq_simp by blast
      have DD2: ‹tgt.ultimateBearer x = x›
        by (simp add: D tgt.ultimate_bearer_eq_simp that)
    show ?thesis
      using DD1 DD2 by simp
  qed

  show ‹img.ultimateBearer x = tgt.ultimateBearer x›
    using D apply (cases x rule: tgt.endurant_cases)
    subgoal using c_substantial .
    using c_moment .
qed


lemma morph_image_directed_moments: ‹img.directed_moments = φ ` src.directed_moments›
  apply (simp only: towardness_sig.directed_moments_def morph_image_towards )
  apply auto
  subgoal G1 by blast
  by (meson inherence_base.endurantI1 morph_reflects_towardness src.inherence_base_axioms
src.towardness_scopeE)


interpretation img: towardness ‹𝓦φ› ‹(◁φ)› ‹(⟶φ)›
  apply (unfold_locales ; (simp only: morph_image_moments morph_image_towards
```

```
      morph_image_directed_moments)?)

  subgoal G1 for x y
    apply (intro conjI ; elim conjE )
    subgoal G1_1 by auto
    using morph_image_substantials by auto
  subgoal G2 for x y
    apply (auto simp: possible_worlds_sig.ed_def possible_worlds_sig.𝒫_def)
    using src.𝒫_I by auto
  subgoal G2 for x y using morph_image_ultimate_bearer by auto
  subgoal G3 for x y₁ y₂ using tgt.towardness_single by blast
  done

lemma morph_image_qualified_particulars: ‹img.qualifiedParticulars = φ ‘ src.qualifiedPa

  apply (simp only: qualified_particulars_sig.qualifiedParticulars_def morph_image_assoc_
  apply auto
  subgoal G1 by (metis imageI mem_Collect_eq morph_reflects_quale_assoc)
  subgoal G2 by (meson imageI src.assoc_quale_scopeD(1))
  using morph_reflects_quale_assoc src.assoc_quale_scopeD(1) by blast

interpretation img: ufo_particular_theory ‹𝒲φ› ‹(◁φ)› ‹src.𝒬𝒮› ‹(⇝φ)› ‹(⟶φ)›
  apply (intro_locales)
  apply (unfold_locales ; simp only: morph_image_qualified_particulars morph_image_inhere
        ; intro notI ; elim conjE)
  using src.qualified_particulars_are_not_bearers
        tgt.qualified_particulars_are_not_bearers
  by (metis img.qOf_assoc_quale_I morph_image_assoc_quale morph_image_qualified_particula
tgt.qualifiedParticulars_iff)

interpretation img1: particular_struct ‹MorphImg φ Γ₁›
  supply A = img.ufo_particular_theory_axioms[simplified ufo_particular_theory_def]
  supply B = A[THEN conjunct1] A[THEN conjunct2,THEN conjunct2]
  apply (intro_locales ; (simp only: particular_struct_morphism_image_simps(1))?)
  subgoal using B(1)[simplified qualified_particulars_def] by metis
  using B(2)[simplified qualified_particulars_def] by metis

interpretation src_to_img1_morph: pre_particular_struct_morphism Γ₁ ‹MorphImg φ Γ₁›

  by (unfold_locales
      ; (simp only: morph_image_particulars morph_image_inheres_in morph_image_towards
morph_image_assoc_quale)?
```

```
      ; simp ; blast)

lemma src_to_img1_world_corresp: ‹src_to_img1_morph.world_corresp w_s w_t ⟷ w_s
∈ src.𝒲 ∧ w_t = φ ` w_s›
proof -

  have A: ‹w_s ∈ src.𝒲 ⟹ w_t = φ ` w_s ⟹ ∃w. φ ` w_s = w ∩ φ ` src.endurants ∧
w ∈ tgt.𝒲›
  proof -
    assume a1: "w_s ∈ src.𝒲"
    assume a2: "w_t = φ ` w_s"
    then have "w_t ∈ {P ∩ φ ` src.endurants |P. P ∈ tgt.𝒲}"
      using a1 by (metis (no_types) image_iff morph_image_worlds_src morph_image_worlds_tgt)
    then show ?thesis
      using a2 by fastforce
  qed


  have B: ‹⋀x. w_s ∈ src.𝒲 ⟹ w_t = φ ` w_s ⟹ x ∈ src.endurants ⟹ φ x ∈ φ `
w_s ⟹ x ∈ w_s›
  proof -
    fix x :: 'p1
    assume a1: "x ∈ src.endurants"
    assume a2: "w_s ∈ src.𝒲"
    assume a3: "φ x ∈ φ ` w_s"
    obtain PP :: "'p1 set ⇒ 'p2 set" where
      "w_s ⇔ PP w_s"
      using a2 by (meson morph_worlds_correspond_src_tgt)
    then have "w_s ∈ src.𝒲 ∧ PP w_s ∈ tgt.𝒲 ∧ (∀p. p ∉ src.endurants ∨ (p ∈ w_s)
= (φ p ∈ PP w_s))"
      by blast
    then show "x ∈ w_s"
      using a3 a1 by (metis imageE possible_worlds_sig.𝒫_I)
  qed

  show ?thesis
    apply (simp only: src_to_img1_morph.world_corresp_def morph_image_worlds_tgt
; simp)
    apply (intro iffI ; (elim conjE)? ; intro conjI ballI iffI ; (elim exE conjE)?
;
            (intro set_eqI iffI conjI)? ; simp?)
    subgoal G1    by blast
```

```
    subgoal G2    by blast
    subgoal G3 using A .
    using B .
qed

interpretation src_to_img1_morph: particular_struct_morphism Γ1 ‹MorphImg φ Γ1› φ
proof -

  have A: ‹∃w. wt = w ∩ φ ‘ src.endurants ∧ w ∈ tgt.𝒲 ⟹ ∃ws. ws ∈ src.𝒲 ∧
wt = φ ‘ ws› for wt
  proof -
    assume "∃w. wt = w ∩ φ ‘ src.endurants ∧ w ∈ tgt.𝒲"
    then obtain PP :: "'p2 set" where
      f1: "wt = PP ∩ φ ‘ src.endurants ∧ PP ∈ tgt.𝒲"
      by blast
    have "PP ∩ φ ‘ src.endurants ∈ {P ∩ φ ‘ src.endurants |P. P ∈ tgt.𝒲} ⟶
(∃P. PP ∩ φ ‘ src.endurants = φ ‘ P ∧ P ∈ src.𝒲)"
      using morph_image_worlds_tgt by auto
    then show ?thesis
      using f1 by blast
  qed
  show ‹particular_struct_morphism Γ1 (MorphImg φ Γ1) φ›
    apply (unfold_locales
           ; (simp only: morph_image_worlds_tgt
                         morph_image_inheres_in
                         morph_image_assoc_quale
                         morph_image_qualified_particulars
                         src_to_img1_world_corresp)?
           ; simp?)
    using A .
qed

lemma morph_image_surjective[intro!,simp]: ‹particular_struct_surjection Γ1 (MorphImg
φ Γ1) φ›
  by (unfold_locales ; simp only: morph_image_particulars)

lemma morph_image_tgt_struct: ‹src_to_img1_morph.tgt.Γ = MorphImg φ Γ1›
  by (intro particular_struct_eqI
      ; simp add: ufo_particular_theory_sig.Γ_def)

lemmas morph_image_simps =
    morph_image_particulars
```

```
    morph_image_worlds_tgt
    morph_image_worlds_src
    morph_image_ed_tgt
    morph_image_indep_iff
    morph_image_inheres_in
    morph_image_trans_inheres_in
    morph_image_moments
    morph_image_assoc_quale
    morph_image_towards
    morph_image_substantials
    morph_image_ultimate_bearer
    morph_image_directed_moments
    morph_image_qualified_particulars
    src_to_img1_world_corresp
    morph_image_tgt_struct

lemmas morph_image_intros =
    morph_image_edI
    morph_image_indepI
    morph_image_inheres_in_I
    morph_image_assoc_quale_I
    morph_image_towards_I

lemmas morph_image_elims =
    morph_image_edE
    morph_image_indepE
    morph_image_inheres_in_E
    morph_image_assoc_quale_E
    morph_image_towards_E

lemmas morph_image_dests =
    morph_image_edD
    morph_image_indepD
    morph_image_inheres_in_D
    morph_image_assoc_quale_D
    morph_image_towards_D

end


end
```

C.3.2 Inverse Images

```
theory InverseImageMorphismChoice
  imports ParticularStructureMorphisms MorphismImage
begin

context particular_struct_morphism
begin

definition same_image (infix ‹∼› 75) where
  ‹x ∼ y ≡ x ∈ src.𝒫 ∧ y ∈ src.𝒫 ∧ φ x = φ y›

lemma same_image_I[intro!]:
  assumes ‹x ∈ src.𝒫› ‹y ∈ src.𝒫› ‹φ x = φ y›
  shows ‹x ∼ y›
  using assms by (auto simp: same_image_def)

lemma same_image_E[elim!]:
  assumes ‹x ∼ y›
  obtains ‹x ∈ src.𝒫› ‹y ∈ src.𝒫› ‹φ x = φ y›
  using assms by (auto simp: same_image_def)

lemma same_image_iff[simp]: ‹x ∼ y ⟷ x ∈ src.𝒫 ∧ y ∈ src.𝒫 ∧ φ x = φ y›
  by (auto simp: same_image_def)

lemma same_image_sym[sym]: ‹x ∼ y ⟹ y ∼ x›
  by auto

lemma same_image_trans[trans]: ‹⟦ x ∼ y ; y ∼ z ⟧ ⟹ x ∼ z›
  by auto

lemma same_image_refl[intro!]: ‹x ∈ src.𝒫 ⟹ x ∼ x›
  by auto

definition ‹eq_class x ≡ { y  . x ∼ y }›

lemma eq_class_I[intro!]: ‹x ∼ y ⟹ y ∈ eq_class x›
  by (auto simp: eq_class_def)

lemma eq_class_D[dest!]: ‹y ∈ eq_class x ⟹ x ∼ y›
  by (auto simp: eq_class_def)
```

```
lemma eq_class_swap[simp]: ‹y ∈ eq_class x ⟷ x ∈ eq_class y›
  by auto

lemma eq_class_unique[simp]: ‹⟦ x ∈ eq_class y ; x ∈ eq_class z ⟧ ⟹ eq_class
x = eq_class z›
  by auto

lemma eq_class_subset_P[intro!]: ‹eq_class x ⊆ src.𝒫›
  by auto

lemma eq_class_endurant_cases[cases set]:
  obtains
    (eq_class_subst) ‹eq_class x ⊆ src.𝒮›
  | (eq_class_moment) ‹eq_class x ⊆ src.ℳ›
proof (cases ‹x ∈ src.𝒫›)
  assume ‹x ∈ src.𝒫›
  then consider (substantial) ‹x ∈ src.𝒮›  | (moment) ‹x ∈ src.ℳ›
    by blast
  then show ?thesis
  proof (cases)
    case substantial
    then show ?thesis
      apply (intro that(1))
      using morph_preserves_substantials ‹x ∈ src.𝒫›
      by (metis eq_class_D particular_struct_morphism.same_image_E particular_struct_morphism_
subsetI)
  next
    case moment
    then show ?thesis
      apply (intro that(2))
      using morph_preserves_moments ‹x ∈ src.𝒫›
      by (metis eq_class_D morph_preserves_moments_simp particular_struct_morphism.same_image_
particular_struct_morphism_axioms subsetI)
  qed
next
  assume ‹x ∉ src.𝒫›
  then have ‹eq_class x = ∅› by auto
  then show ?thesis using that by auto
qed

lemma eq_class_moment[simp]: ‹eq_class x ⊆ src.ℳ ⟷ x ∉ src.𝒮›
  apply (cases x rule: eq_class_endurant_cases ; safe ; simp add: eq_class_def)
```

```
  subgoal by blast
  subgoal by auto
  by blast

lemma eq_class_substantial[simp]: ‹eq_class x ⊆ src.𝒮 ⟷ x ∉ src.ℳ›
  apply (cases x rule: eq_class_endurant_cases ; safe ; simp add: eq_class_def)
  subgoal by blast
  subgoal by auto
  by (simp add: subset_iff)


definition ‹eq_classes ≡ { eq_class x | x . x ∈ src.𝒫 }›

lemma eq_classes_I[intro]: ‹⟦ x ∈ src.𝒫 ; X = eq_class x ⟧ ⟹ X ∈ eq_classes ›
  by (auto simp: eq_classes_def)

lemma eq_classes_E[elim!]:
  assumes ‹X ∈ eq_classes›
  obtains x where ‹x ∈ src.𝒫› ‹X = eq_class x›
  using assms by (auto simp: eq_classes_def)

lemma eq_classes_disj: ‹⟦ X ∈ eq_classes ; Y ∈ eq_classes ; x ∈ X ; x ∈ Y ⟧ ⟹
X = Y›
  by auto

lemma eq_classes_un: ‹⋃ eq_classes = src.𝒫›
  by auto

lemma eq_classes_non_empty[dest]: ‹X ∈ eq_classes ⟹ X ≠ ∅›
  by (auto simp: eq_classes_def eq_class_def)

lemma eq_class_non_empty[simp]: ‹eq_class x ≠ ∅ ⟷ x ∈ src.𝒫›
  by (auto simp: eq_class_def)

definition ‹subst_eq_classes ≡ { X . X ∈ eq_classes ∧ X ⊆ src.𝒮}›

lemma subst_eq_classes_I: ‹⟦ X ∈ eq_classes ; X ⊆ src.𝒮 ⟧ ⟹ X ∈ subst_eq_classes›
  by (auto simp: subst_eq_classes_def)

lemma subst_eq_classes_E:
  assumes ‹X ∈ subst_eq_classes›
  obtains ‹X ∈ eq_classes› ‹X ⊆ src.𝒮›
```

```
  using assms by (auto simp: subst_eq_classes_def)

lemma subst_eq_classes_E1[elim!]:
  assumes ‹X ∈ subst_eq_classes›
  obtains x where ‹X ∈ eq_classes› ‹x ∈ X› ‹⋀x. x ∈ X ⟹ x ∈ src.𝒮›
  using assms by (auto simp: subst_eq_classes_def)


lemma subst_eq_classes_iff: ‹X ∈ subst_eq_classes ⟷ X ∈ eq_classes ∧ X ⊆ src.𝒮›
  by (auto simp: subst_eq_classes_def)

lemma subst_eq_classes_I1[intro]: ‹⟦ x ∈ src.𝒮 ; X = eq_class x ⟧ ⟹ X ∈ subst_eq_classes›
  by (auto simp: subst_eq_classes_def)

lemma subst_eq_classes_D1: ‹⟦ X ∈ subst_eq_classes ; x ∈ X ⟧ ⟹ x ∈ src.𝒮›
  by auto

lemma subst_eq_classes_D2: ‹X ∈ subst_eq_classes ⟹ X ∈ eq_classes›
  by auto

lemma subst_classes_disj[simp]:
  ‹⟦ X ∈ subst_eq_classes ; Y ∈ subst_eq_classes ; x ∈ X ; x ∈ Y ⟧ ⟹ X = Y›
  using subst_eq_classes_D2 eq_classes_disj by metis


lemma subst_eq_classes_un[simp]: ‹⋃ subst_eq_classes = src.𝒮›
  by auto

lemma subst_eq_class_non_empty[dest]: ‹X ∈ subst_eq_classes ⟹ X ≠ ∅›
  by auto

end

context particular_struct_morphism_sig
begin


end

locale choice_function =
  fixes f :: ‹'a set ⇒ 'a›
  assumes f_in_X1: ‹∀X. X ≠ ∅ ⟶ f X ∈ X›
```

locale *particular_struct_morphism_with_choice* =
  particular_struct_morphism **where** Typ_{p1} = Typ_{p1} **and** Typ_{p2} = Typ_{p2} **and** Typ_q = Typ_q +
  choice_function f
 for
  f :: ‹$'p_1$ *set* ⇒ $'p_1$› **and**
  Typ_{p1} :: ‹$'p_1$ *itself*› **and**
  Typ_{p2} :: ‹$'p_2$ *itself*› **and**
  Typ_q :: ‹*'q itself*›
begin

inductive_set *delta* :: ‹($'p_2$ × $'p_1$) *set*› (‹Δ›)
 where
 delta_substantial: ‹*X* ∈ *subst_eq_classes* ⟹ (φ (*f X*), *f X*) ∈ Δ›
 | *delta_moment*: ‹⟦ (x_1,y_1) ∈ Δ ; x_2 ∈ *src*.𝒫 ; φ x_2 ◁$_t$ x_1 ⟧ ⟹ (φ x_2,*f* (*eq_class* x_2)) ∈ Δ›

lemma *f_in_X[simp]*: ‹*f X* ∈ *X* ⟷ *X* ≠ ∅› **using** *f_in_X1* **by** *auto*

lemma *f_eq_class[simp]*: ‹*x* ∈ *src*.𝒫 ⟹ φ (*f* (*eq_class x*)) = φ *x*›
 by (*metis eq_class_def eq_class_non_empty f_in_X1 mem_Collect_eq same_image_E*)

lemma *f_eq_class_in_end[intro!,simp]*: ‹*x* ∈ *src*.𝒫 ⟹ *f* (*eq_class x*) ∈ *src*.𝒫›
 by (*metis eq_class_non_empty eq_class_unique f_in_X1*)

lemma *x_sim_f_eq_class[simp,intro!]*: ‹*x* ∼ *f* (*eq_class x*)› **if** ‹*x* ∈ *src*.𝒫› **for** *x*
 by (*auto simp: that*)

lemma *delta_dom*:
 assumes ‹(*x*,*y*) ∈ Δ›
 shows ‹*x* ∈ φ ‘ *src*.𝒫 ∧ *y* ∈ *src*.𝒫›
 using *assms*
 by (*induct rule: delta.induct ; safe? ; simp*)

lemma *delta_domE*:
 assumes ‹(*x*,*y*) ∈ Δ›
 obtains *z* **where** ‹*z* ∈ *src*.𝒫› ‹*x* = φ *z*› ‹φ *z* ∈ *tgt*.𝒫› ‹*y* ∈ *src*.𝒫›
 using *assms[THEN delta_dom]*
 by (*elim conjE imageE ; simp add: morph_preserves_particulars*)

lemma *delta_img*:

```
  assumes ‹(x,y) ∈ Δ›
  shows ‹φ y = x›
proof -
  show ?thesis
    using assms by (induct ; hypsubst_thin? ; simp)
qed

declare [[smt_timeout=600]]

lemma delta_E1:
  assumes ‹(x,y) ∈ Δ›
  obtains x_s where ‹x = φ x_s› ‹y = f (eq_class x_s)›
  using assms apply (cases ; simp)
  using f_eq_class by blast

lemma delta_E2:
  assumes ‹(x,y) ∈ Δ›
  obtains X where ‹X ∈ eq_classes› ‹x = φ y› ‹y = f X›
  using assms apply (cases ; simp)
  subgoal by auto
  by blast

lemma delta_single:
  assumes ‹(x,y_1) ∈ Δ› ‹(x,y_2) ∈ Δ›
  shows ‹y_1 = y_2›
  using assms
proof -
  obtain doms[simp,intro!]: ‹y_1 ∈ src.𝒫› ‹y_2 ∈ src.𝒫› ‹x ∈ φ ` src.𝒫›
      and phi_y[simp]: ‹φ y_1 = x› ‹φ y_2 = x›
    using assms
    by (simp add: delta_dom delta_img)

  show ?thesis
    using assms doms phi_y
  proof (induct arbitrary: y_2)
    show G1: ‹f X = y_2›
      if A: ‹X ∈ subst_eq_classes› ‹(φ (f X), y_2) ∈ Δ›
        and doms: ‹f X ∈ src.𝒫› ‹y_2 ∈ src.𝒫›
                  ‹φ (f X) ∈ φ ` src.𝒫›
                  ‹φ (f X) = φ (f X)›
                  ‹φ y_2 = φ (f X)›
                for X y_2
```

```
proof -
  have B: ‹f X ∈ src.𝒮› using A by blast
  then have C: ‹φ (f X) ∈ tgt.𝒮› using morph_preserves_substantials by blast
  from doms(2) show ‹f X = y₂›
  proof (cases y₂ rule: src.endurant_cases)
    assume substantial: ‹y₂ ∈ src.𝒮›
    then obtain D: ‹φ y₂ ∈ tgt.𝒮› ‹φ (f X) ∈ tgt.𝒮›
      using ‹f X ∈ src.𝒮› by (meson inherence_sig.𝒮_E morph_preserves_substantials)

    show ?thesis using A(2)
      apply (cases rule: delta.cases)
       subgoal by (metis (mono_tags, lifting) A(1) B doms(2) eq_class_I eq_classes_I
f_in_X1 mem_Collect_eq particular_struct_morphism.eq_classes_disj particular_struct_morph
particular_struct_morphism_axioms src.endurantI3 subst_eq_class_non_empty subst_eq_classe
       using C by auto
  next
    assume moment: ‹y₂ ∈ src.ℳ›
    show ?thesis using A(2)
      apply (cases)
      subgoal  using moment by blast
      by (metis C inherence_sig.ℳ_I inherence_sig.𝒮_E)
  qed
qed

fix x₃ y₃ x₄ y₄
assume A: ‹(x₃, y₃) ∈ Δ›
          ‹⋀y₂. ⟦ (x₃, y₂) ∈ Δ
                ; y₃ ∈ src.𝒫
                ; y₂ ∈ src.𝒫
                ; x₃ ∈ φ ‘ src.𝒫
                ; φ y₃ = x₃
                ; φ y₂ = x₃ ⟧ ⟹ y₃ = y₂›
          ‹x₄ ∈ src.𝒫›
          ‹φ x₄ ◁t x₃›
          ‹(φ x₄, y₄) ∈ Δ›
          ‹f (eq_class x₄) ∈ src.𝒫›
          ‹y₄ ∈ src.endurants›
          ‹φ x₄ ∈ φ ‘ src.endurants›
          ‹φ (f (eq_class x₄)) = φ x₄›
          ‹φ y₄ = φ x₄›
then have ‹eq_class x₄ = eq_class y₄› using A(3) A(7) by auto
obtain Y where Y: ‹Y ∈ eq_classes› ‹y₄ = f Y›
```

```
      using delta_E2[OF ‹(φ x4, y4) ∈ Δ›] by metis
    have ‹Y = eq_class y4›  using Y(1) Y(2) eq_class_unique by blast
    then have ‹f (eq_class y4) = y4› using Y(2) by simp
    then show ‹f (eq_class x4) = y4›
      using ‹eq_class x4 = eq_class y4› by simp
  qed
qed

lemma delta_range:
  assumes ‹x ∈ φ ‘ src.𝒫›
  shows ‹∃y. (x,y) ∈ Δ›
  using assms
  apply (induct x rule: wfP_induct[OF tgt.inherence_is_noetherian] ; simp)
proof -
  fix x
  assume A: ‹∀y. x ◁t y ⟶ y ∈ φ ‘ src.𝒫 ⟶ (∃ya. (y, ya) ∈ Δ)›
        ‹x ∈ φ ‘ src.𝒫›
  then obtain xs where B[simp]: ‹x = φ xs› ‹xs ∈ src.𝒫› using A(2) by blast
  have D: ‹eq_class xs ∈ subst_eq_classes› if ‹xs ∈ src.𝒮›
    using that by auto
  have E: ‹φ (f (eq_class xs)) = φ xs› using B(2) by simp

  have substantial: ‹∃y. (φ xs, y) ∈ Δ› if ‹xs ∈ src.𝒮›
  proof-
    have ‹(φ xs, f (eq_class xs)) ∈ Δ› if ‹xs ∈ src.𝒮›
      using delta.intros(1)[OF D,simplified E,OF that] .
    then show ?thesis
      using that by blast
  qed

  have moment: ‹∃y. (φ xs, y) ∈ Δ› if as: ‹xs ∈ src.ℳ›
  proof-
    obtain ys where F: ‹xs ◁s ys› using as by blast
    then obtain G: ‹ys ∈ src.𝒫› ‹φ xs ◁t φ ys›
      using morph_reflects_inherence by auto
    then have H: ‹φ ys ∈ φ ‘ src.𝒫› by blast
    obtain y where I: ‹(φ ys,y) ∈ Δ›
      using A(1)[rule_format,simplified,OF G(2) H]  by blast
    have ‹(φ xs, f (eq_class xs)) ∈ Δ›
      using delta.intros(2)[OF I B(2) G(2)] by blast
    then show ?thesis by blast
  qed
```

```
  show ‹∃y. (x, y) ∈ Δ›
    apply (simp)
    using B(2) apply (cases x_s rule: src.endurant_cases)
    using substantial moment by auto
qed

lemma delta_inj:
  assumes ‹(x_1,y) ∈ Δ› ‹(x_2,y) ∈ Δ›
  shows ‹x_1 = x_2›
  using assms delta_img by auto

definition someInvMorph :: ‹'p_2 ⇒ 'p_1› where
  ‹someInvMorph x ≡ if x ∈ φ ` src.𝒫 then THE y. (x,y) ∈ Δ else undefined›

lemma someInvMorph_ex:
  assumes ‹x ∈ φ ` src.𝒫›
  shows ‹∃!y. (x,y) ∈ Δ›
  using assms
  by (intro ex_ex1I ; simp add: delta_range delta_single)

lemma someInvMorph_eq_iff:
  assumes ‹x ∈ φ ` src.𝒫›
  shows ‹someInvMorph x = y ⟷ (x,y) ∈ Δ›
  apply (simp add: someInvMorph_def assms)
  apply (rule the1I2[OF someInvMorph_ex[OF assms]] ; intro iffI ; simp?)
  using delta_single by simp

lemma someInvMorph_inj_phi_img: ‹inj_on someInvMorph (φ ` src.𝒫)›
  apply (intro inj_onI)
  subgoal premises P for x y
    supply S = P(1,2)[THEN someInvMorph_eq_iff,simplified P(3)]
    using S(1)[simplified S(2)]
    by (meson P(1) delta_range delta_inj)
  done

lemma someInvMorph_delta_I[intro!]:
  assumes ‹x ∈ φ ` src.𝒫› ‹y = someInvMorph x›
  shows ‹(x,y) ∈ Δ›
  using assms someInvMorph_eq_iff by blast

lemma someInvMorph_delta_E:
```

```
  assumes ‹(x,y) ∈ Δ›
  obtains ‹x ∈ φ ` src.𝒫› ‹y = someInvMorph x›
  using assms
  by (metis delta_dom someInvMorph_eq_iff)

lemma someInvMorph_phi_phi[simp]:
  assumes ‹x ∈ src.𝒫›
  shows ‹φ (someInvMorph (φ x)) = φ x›
  apply (simp add: someInvMorph_def imageI[OF assms])
  apply (rule the1I2[of ‹λy. (φ x, y) ∈ Δ›])
  subgoal using assms by (simp add: someInvMorph_ex)
  using assms delta_img by blast

lemma someInvMorph_as_inv[simp]: ‹x ∈ φ ` src.𝒫 ⟹ φ (someInvMorph x) = x›
  by auto

lemma someInvMorph_delta_simp: ‹(x,y) ∈ Δ ⟷ x ∈ φ ` src.𝒫 ∧ y = someInvMorph
x›
  using someInvMorph_delta_E by blast

lemma someInvMorph_image: ‹someInvMorph ` φ ` src.𝒫 ⊆ src.𝒫›
  using delta_dom by blast

lemma someInvMorphImgUnique:
  assumes ‹x ∼ y› ‹x ∈ someInvMorph ` φ ` src.𝒫› ‹y ∈ someInvMorph ` φ ` src.𝒫›
  shows ‹x = y›
  using assms
  by auto



context
begin

interpretation img: particular_struct_surjection Γ1 ‹MorphImg φ Γ1› φ
  by simp

declare img.morph_is_surjective[simp del]

interpretation some_inv_to_some_inv_img: particular_struct_bijection_1 ‹MorphImg φ
Γ1› someInvMorph ‹TYPE('p2)› ‹TYPE('p1)›
  apply (intro img.tgt.inj_morph_img_isomorphism[simplified morph_image_tgt_struct])
```

```
  subgoal using someInvMorph_inj_phi_img img.morph_is_surjective by auto
  using src.injection_to_ZF_exist by blast

private lemma A1[simp]: ‹some_inv_to_some_inv_img.tgt.𝒬𝒮 = src.𝒬𝒮›
  by auto

private lemma S1[simp]: ‹some_inv_to_some_inv_img.tgt.endurants = someInvMorph ‘
φ ‘ src.𝒫›
  using img.morph_is_surjective
  by auto

private lemma C1: ‹someInvMorph ‘ φ ‘ src.𝒫 ⊆ src.𝒫›
    using someInvMorph_image by blast

private lemma A2: ‹x ∈ src.𝒫› if ‹x ∈ someInvMorph ‘ φ ‘ src.𝒫› for x
    using C1 that by blast

private lemma A3[simp]: ‹some_inv_to_some_inv_img.img_inheres_in x y ⟷
              (∃x₁ y₁. img.src_inheres_in x₁ y₁ ∧ x = someInvMorph (φ x₁) ∧ y =
someInvMorph(φ y₁))› for x y
    apply (intro iffI ; (elim exE conjE)? ; hypsubst_thin?)
    subgoal  by (metis delta_dom morph_image_inheres_in_E morph_reflects_inherence
                  someInvMorph_as_inv someInvMorph_delta_I
                  some_inv_to_some_inv_img.inv_inheres_in_reflects
                  some_inv_to_some_inv_img.inv_morph_morph
                  some_inv_to_some_inv_img.tgt.inherence_scope)
    subgoal for x₁ y₁
      by blast
    done

private lemma A4[simp]: ‹some_inv_to_some_inv_img.img_towards x y ⟷
      (∃x₁ y₁. img.src_towards x₁ y₁ ∧ x = someInvMorph (φ x₁) ∧ y = someInvMorph(φ
y₁))› for x y
    apply (intro iffI ; (elim exE conjE)? ; hypsubst_thin?)
    subgoal
      by (metis S1 delta_dom img.I_img_eq_tgt_I morph_image_def morph_image_towards_D

          morph_reflects_towardness someInvMorph_as_inv someInvMorph_delta_I
          some_inv_to_some_inv_img.inv_morph_morph
          some_inv_to_some_inv_img.inv_towardness_reflects
          some_inv_to_some_inv_img.morph_image_towards_D(1,2))
    subgoal for x₁ y₁
```

```
      apply (simp only: particular_struct_morphism_image_simps)
      by blast
    done

private lemma A5[simp]: ‹some_inv_to_some_inv_img.img_assoc_quale x q ⟷
      (∃y. img.src_assoc_quale y q ∧ x = someInvMorph (φ y))› for x  q
    apply (intro iffI ; (elim exE conjE)? ; hypsubst_thin?)
    subgoal
      by (metis delta_E2 delta_dom img.I_img_eq_tgt_I morph_image_def morph_image_dests(9)

                morph_reflects_quale_assoc someInvMorph_delta_I
                some_inv_to_some_inv_img.I_img_eq_tgt_I
                some_inv_to_some_inv_img.morph_image_E
                some_inv_to_some_inv_img.morph_image_iff
                some_inv_to_some_inv_img.morph_reflects_quale_assoc
                some_inv_to_some_inv_img.tgt.assoc_quale_scopeD(1)
                src.𝒫_def)
    subgoal for x1
      apply (simp only: particular_struct_morphism_image_simps)
      by blast
    done

private lemma ex_simp1: ‹(∃x ∈ X. y = x) ⟷ y ∈ X› for y :: 'p1 and X by blast

private lemma A6: ‹∃y∈someInvMorph ‘ φ ‘ src.𝒫. z = y›
    if as: ‹x ∈ someInvMorph ‘ φ ‘ src.𝒫› ‹img.src_towards x z› for x z
  proof (simp only: ex_simp1)
    obtain y where Y: ‹y ∈ src.𝒫› ‹x = someInvMorph (φ y)›
      using as imageE by blast
    have AA: ‹img.src_towards (someInvMorph (φ y)) z› using as(2) Y by simp
    then have ‹img.tgt_towards (φ (someInvMorph (φ y))) (φ z)›
      using Y(1)
      by (meson particular_struct_morphism_image_simps(5))
    then have ‹img.tgt_towards (φ y) (φ z)›
      using someInvMorph_phi_phi Y by simp
    then have ‹some_inv_to_some_inv_img.tgt_towards (someInvMorph (φ y)) (someInvMorph
(φ z))›
      apply (simp only: particular_struct_morphism_image_simps ; elim exE conjE)
      by blast
    then have ‹some_inv_to_some_inv_img.tgt_towards x (someInvMorph (φ z))›
      using Y(2) by simp
    then have BB: ‹img.src_towards x (someInvMorph (φ z))›
```

```
      by (metis A2 Y(2) ‹some_inv_to_some_inv_img.src_towards (φ (someInvMorph (φ
y))) (φ z)›
            img.I_img_eq_tgt_I img.morph_reflects_towardness morph_image_def
            morph_image_towards_D(2) someInvMorph_as_inv
            some_inv_to_some_inv_img.morph_image_towards_D(2) that(1))
    have CC: ‹someInvMorph (φ z) = z›
      using src.towardness_single as(2) BB by simp
    then show ‹z ∈ someInvMorph ‘ φ ‘ src.endurants›
      using CC
      by (metis ‹some_inv_to_some_inv_img.src_towards (φ y) (φ z)›
            image_eqI morph_image_towards_E)
  qed


interpretation some_inv_img_to_src: pre_particular_struct_morphism ‹MorphImg (someInvMorph
∘ φ) Γ1› Γ1 id ‹TYPE('p1)› ‹TYPE('p1)›
proof -

  show ‹pre_particular_struct_morphism (MorphImg (someInvMorph ∘ φ) Γ1) Γ1 id›
    apply (simp only: morph_img_comp ; unfold_locales ; (simp only: id_def A1 S1
A3 A4 A5)?)
    subgoal AX2 using A2 .
    subgoal AX3 for x y
      by (metis A2 S1 img.I_img_eq_tgt_I img.morph_reflects_inherence morph_image_def

            someInvMorph_as_inv someInvMorph_phi_phi some_inv_to_some_inv_img.I_img_eq_tg

            some_inv_to_some_inv_img.morph_image_E src.inherence_scope)
    subgoal AX4 for x z
      by (metis AX2 S1 img.morph_preserves_particulars img.morph_reflects_inherence

            someInvMorph_phi_phi some_inv_to_some_inv_img.I_img_eq_tgt_I
            some_inv_to_some_inv_img.morph_image_I src.endurantI2 src.moment_non_migratio

    subgoal AX5 for x y

      by (metis (no_types, lifting) A4 AX2 S1 img.morph_reflects_towardness morph_image_p

            particular_struct_morphism_sig.morph_image_iff someInvMorph_as_inv
            some_inv_to_some_inv_img.I_img_eq_tgt_I
            some_inv_to_some_inv_img.morph_reflects_towardness)
    subgoal AX6 for x z using A6 .
```

```
    subgoal AX7 for x q
      by (metis A2 S1 morph_image_particulars morph_reflects_quale_assoc someInvMorph_as_inv

            someInvMorph_phi_phi some_inv_to_some_inv_img.I_img_eq_tgt_I
            some_inv_to_some_inv_img.morph_image_iff src.assoc_quale_scopeD(1))
    done
qed

private lemma A7: ‹∃ w_s ∈ src.𝒲. w ⊆ w_s› if as: ‹w ∈ some_inv_to_some_inv_img.𝒲_φ›
for w
  proof (rule ccontr ; simp)
    assume AA: ‹∀ x∈src.𝒲. ¬ w ⊆ x›
    then have BB: False if ‹w_1 ∈ src.𝒲› ‹w ⊆ w_1› for w_1 using that by metis
    obtain w_2 where CC: ‹some_inv_to_some_inv_img.world_corresp w_2 w› using as by
blast
    then obtain DD: ‹w_2 ∈ some_inv_to_some_inv_img.src.𝒲›
                ‹⋀x. x ∈ some_inv_to_some_inv_img.src.𝒫 ⟹ x ∈ w_2 ⟷ someInvMorph
x ∈ w›
      using some_inv_to_some_inv_img.world_corresp_E[OF CC] by metis
    have EE: ‹w_2 ∈ ((`) φ) ` src.𝒲› using DD(1) by (simp add: morph_image_worlds_src)
    then obtain w_3 where FF: ‹w_2 = φ ` w_3› ‹w_3 ∈ src.𝒲› by blast
    have GG: ‹⋀x. x ∈ φ ` src.𝒫 ⟹ x ∈ φ ` w_3 ⟷ someInvMorph x ∈ w›
      using DD(2) img.morph_is_surjective by (simp only: FF(1) ; simp)
    have HH: ‹⋀x. x ∈ φ ` w_3 ⟹ someInvMorph x ∈ w›
      using GG src.worlds_are_made_of_particulars FF(2) by blast
    then have II: ‹⋀x. x ∈ w_3 ⟹ someInvMorph (φ x) ∈ w› by blast
    then have JJ: ‹⋀x. x ∈ w_3 ⟹ φ (someInvMorph (φ x)) ∈ φ ` w› by blast
    then have KK: ‹⋀x. x ∈ w_3 ⟹ (φ x) ∈ φ ` w›
      using FF(2) src.𝒫_I by auto
    show False
      by (metis A2 BB DD(1) DD(2) img.morph_worlds_correspond_tgt_src
          img.world_corresp_def morph_image_particulars
          someInvMorph_as_inv some_inv_to_some_inv_img.I_img_eq_tgt_I
          some_inv_to_some_inv_img.morph_image_iff
          some_inv_to_some_inv_img.morph_is_surjective some_inv_to_some_inv_img.tgt.𝒫_I

          subsetI that)
  qed

private abbreviation srcWorlds (‹𝒲_A›) where ‹𝒲_A ≡ src.𝒲›

private abbreviation srcParticulars (‹𝒫_A›) where ‹𝒫_A ≡ src.𝒫›
```

private abbreviation *srcInheresIn* **(infix** ‹$\triangleleft_A$› *75)* **where**
‹($\triangleleft_A$) ≡ *src_inheres_in*›

private abbreviation *srcAssocQuale* **(infix** ‹$\rightsquigarrow_A$› *75)* **where**
‹($\rightsquigarrow_A$) ≡ *src_assoc_quale*›

private abbreviation *srcQualia* (‹$\mathcal{Q}_A$›) **where**
‹$\mathcal{Q}_A$ ≡ *img.*$\mathcal{Q}_s$›

private abbreviation *srcQualitySpaces* (‹$\mathcal{QS}_A$›) **where**
‹$\mathcal{QS}_A$ ≡ *some_inv_img_to_src.tgt.*$\mathcal{QS}$›

private abbreviation *revImageParticulars* (‹$\mathcal{P}_R$›) **where** ‹$\mathcal{P}_R$ ≡ *some_inv_to_some_inv_img.tgt*

private abbreviation *revImageInheresIn* **(infix** ‹$\triangleleft_R$› *75)* **where**
‹($\triangleleft_R$) ≡ *some_inv_img_to_src.src_inheres_in*›

private lemma *some_inv_to_some_inv_img_img_inheres_in_eq:* ‹*some_inv_to_some_inv_img.img_i*
= ($\triangleleft_R$)›
by *(intro ext ; simp)*

private abbreviation *revImageAssocQuale* **(infix** ‹$\rightsquigarrow_R$› *75)* **where**
‹($\rightsquigarrow_R$) ≡ *some_inv_to_some_inv_img.img_assoc_quale*›

private abbreviation *revImageQualia* (‹$\mathcal{Q}_R$›) **where**
‹$\mathcal{Q}_R$ ≡ *some_inv_img_to_src.*$\mathcal{Q}_s$›

private abbreviation *revImageWorldCorresp* **(infix** ‹$\Leftrightarrow_R$› *75)* **where**
‹($\Leftrightarrow_R$) ≡ *some_inv_img_to_src.world_corresp*›

private abbreviation *revImageWorlds* (‹$\mathcal{W}_R$›) **where**
‹$\mathcal{W}_R$ ≡ *some_inv_img_to_src.src.*$\mathcal{W}$›

private lemma *some_inv_to_some_inv_img_*$\mathcal{W}_\varphi$*:* ‹*some_inv_to_some_inv_img.*$\mathcal{W}_\varphi$ = $\mathcal{W}_R$›
by *auto*

private abbreviation *imageInheresIn* **(infix** ‹$\triangleleft_I$› *75)* **where**
‹($\triangleleft_I$) ≡ *img.tgt_inheres_in*›

private abbreviation *imageWorlds* (‹$\mathcal{W}_I$›) **where**
‹$\mathcal{W}_I$ ≡ *img.tgt.*$\mathcal{W}$›

```
private abbreviation imageAssocQuale (infix ‹⇝I› 75) where
  ‹(⇝I) ≡ img.tgt_assoc_quale›

private abbreviation imageQualia (‹QI›) where
  ‹QI ≡ img.Qt›

private abbreviation imageParticulars (‹PI›) where ‹PI ≡ img.tgt.P›

private abbreviation imageWorldCorresp (infix ‹⇔I› 75) where
  ‹(⇔I) ≡ img.world_corresp›

private abbreviation someInvMorphAbbrev (‹φ←›) where
  ‹φ← ≡ someInvMorph›


interpretation some_inv_img_to_src: particular_struct_morphism ‹MorphImg (someInvMorph
∘ φ) Γ1› Γ1 id ‹TYPE('p1)› ‹TYPE('p1)›
  apply (unfold_locales)
  subgoal G1 for ws
    apply (auto ; simp only: particular_struct_morphism_sig.world_corresp_def)
    apply (simp only: particular_struct_morphism_image_simps
                possible_worlds_sig.P_def id_def ; simp)
    apply (elim exE conjE ; simp)
    subgoal for w1 w2
      apply (intro exI[of _ w2])
      apply (intro conjI allI impI ballI iffI ; (elim exE conjE)? ; simp? ; hypsubst_thin?)
      subgoal for _ y _ w4
        by (metis A2 S1 img.I_img_eq_tgt_I img.world_corresp_def morph_image_def
someInvMorph_as_inv some_inv_to_some_inv_img.I_img_eq_tgt_I some_inv_to_some_inv_img.morph_ima
some_inv_to_some_inv_img.src_world_corresp_image some_inv_to_some_inv_img.tgt.P_I
some_inv_to_some_inv_img.world_corresp_def src_to_img1_world_corresp)
      subgoal for _ y _ w4
        by (smt image_iff img.I_img_eq_tgt_I img.world_preserve_img morph_image_def
someInvMorph_as_inv some_inv_to_some_inv_img.I_img_eq_tgt_I some_inv_to_some_inv_img.morph_ima
some_inv_to_some_inv_img.tgt.P_I some_inv_to_some_inv_img.world_preserve_img)
      done
    done
  subgoal G2 for wt
    apply (auto ; simp only: particular_struct_morphism_sig.world_corresp_def)
    apply (simp only: particular_struct_morphism_image_simps
                possible_worlds_sig.P_def id_def )
```

```
    subgoal premises P
      apply (rule exE[OF img.morph_worlds_correspond_src_tgt[OF P]])
      subgoal  for w1
        apply (elim img.world_corresp_E)
        subgoal premises Q
          apply (rule exE[OF some_inv_to_some_inv_img.morph_worlds_correspond_src_tgt[OF
Q(2)]])
          subgoal for w2
            apply (elim some_inv_to_some_inv_img.world_corresp_E)
            subgoal premises T
              apply (rule T(2)[THEN A7, THEN bexE])
              subgoal premises V for w3
                apply (intro exI[of _ w2] conjI ballI iffI
                      ; (intro CollectI)? ; (elim UnionE CollectE exE conjE)?
                      ; simp)
                prefer 2
                subgoal G2_2 using P Q T V(1) V(2)[THEN subsetD]
                  by (metis A2 S1 morph_image_particulars someInvMorph_as_inv some_inv_to
some_inv_to_some_inv_img.morph_image_iff some_inv_to_some_inv_img.tgt.𝒫_I)
                prefer 2
                subgoal G2_3 using P Q T V(1) V(2)[THEN subsetD]
                  by (metis img.world_preserve_img morph_image_particulars someInvMorph_a
some_inv_to_some_inv_img.I_img_eq_tgt_I some_inv_to_some_inv_img.morph_image_iff
some_inv_to_some_inv_img.tgt.𝒫_I some_inv_to_some_inv_img.world_preserve_img src.𝒫_I)
                subgoal G2_1
                  apply (intro exI[of _ w1] conjI  exI[of _ wt] )
                  subgoal G2_1_1
                    using P Q T V apply auto
                    subgoal G2_1_1_1 for x
                      by (metis some_inv_to_some_inv_img.morph_image_E some_inv_to_some_i
some_inv_to_some_inv_img.morph_is_surjective some_inv_to_some_inv_img.src_world_corresp_i
some_inv_to_some_inv_img.tgt.𝒫_I some_inv_to_some_inv_img.world_corresp_E)
                    subgoal G2_1_1_2 for x by blast
                    done
                  subgoal G2_1_2
                    using P Q T V apply auto
                    subgoal for x
                      by (smt A2 imageI morph_image_particulars possible_worlds_sig.𝒫_I
someInvMorph_as_inv)
                    done
                  subgoal G2_1_3
                    using P Q T V by auto
```

```
                done
              done
            done
          done
        done
      done
    done
  done
  done

private lemma lemma1: ‹particular_struct_morphism Γ1  Γ1 (id ∘ someInvMorph ∘ φ)›
  apply (intro particular_struct_morphism_comp[of _ ‹MorphImg (someInvMorph ∘ φ)
Γ1›]
              particular_struct_morphism_comp[of _ ‹MorphImg φ Γ1›])
  using img.particular_struct_morphism_axioms
        some_inv_to_some_inv_img.particular_struct_morphism_axioms
        some_inv_img_to_src.particular_struct_morphism_axioms
  by auto

interpretation src_to_src: particular_struct_morphism Γ1  Γ1 ‹someInvMorph ∘ φ› ‹TYPE('p1)›
‹TYPE('p1)›
  using lemma1 by simp

private lemma someInvMorph_to_endomorphism: ‹particular_struct_endomorphism Γ1 (someInvMorph
∘ φ)›
  by (intro_locales)

private lemma someInvMorph_to_eq_class_choice: ‹(someInvMorph ∘ φ) x = f (eq_class
x)› if ‹x ∈ 𝒫A›
  using that apply simp
  by (smt delta_E2 delta_dom f_in_X1 img.eq_class_I img.eq_classes_I img.eq_classes_disj

        img.eq_classes_non_empty morph_image_I morph_image_def
        particular_struct_morphism.same_image_I particular_struct_morphism_axioms

        someInvMorph_delta_I)

lemma eq_class_choice_inv_morph_ex: ‹∃σ. particular_struct_endomorphism Γ1 σ ∧
(∀x ∈ src.𝒫. σ x = f (eq_class x))›
  by (intro exI[of _ ‹someInvMorph ∘ φ›] conjI ballI someInvMorph_to_endomorphism
someInvMorph_to_eq_class_choice ; simp)
```

```
end

lemmas eq_class_choice = eq_class_choice_inv_morph_ex


end

context particular_struct_morphism
begin

lemma choice_from_choice_ex:
  assumes ‹x ∈ src.𝒫› ‹y ∈ src.𝒫› ‹φ x = φ y›
        ‹∃(f :: 'p1 set ⇒ 'p1). particular_struct_morphism_with_choice Γ1 Γ2 φ
f›
  shows ‹∃(σ :: 'p1 ⇒ 'p1). particular_struct_endomorphism Γ1 σ ∧ σ x = y ∧ σ
y = y›
proof -
  obtain f :: ‹'p1 set ⇒ 'p1› where ‹particular_struct_morphism_with_choice Γ1 Γ2
φ f›
    using assms by blast
  then interpret f_choice: particular_struct_morphism_with_choice ‹Γ1› ‹Γ2› ‹φ› ‹f›
by simp
  define g where ‹g X ≡ if y ∈ X then y else f X› for X
  interpret g_choice: particular_struct_morphism_with_choice ‹Γ1› ‹Γ2› φ g
    apply (unfold_locales)
    apply (intro allI impI ; simp only: g_def)
    subgoal for X
      by (cases ‹y ∈ X› ; simp)
    done

  obtain σ where A: ‹particular_struct_endomorphism Γ1 σ›
          ‹⋀x. x ∈ src.𝒫 ⟹ σ x = g (eq_class x)›
    using g_choice.eq_class_choice by blast

  have B: ‹σ x = y› by (auto simp: g_def A(2) assms)
  have C: ‹σ y = y› by (auto simp: g_def A(2) assms)
  show ?thesis
    by (intro exI[of _ σ] conjI A(1) B C)
qed

lemma choice_exists:
  ‹∃(f :: 'p1 set ⇒ 'p1). particular_struct_morphism_with_choice Γ1 Γ2 φ f›
```

```
  apply (intro exI[of _ "λX. SOME x. x ∈ X"])
  apply (unfold_locales ; auto)
  subgoal for X x
    using someI[of ‹λx. x ∈ X›,of x] by blast
  done

lemma choice:
  assumes ‹x ∈ src.𝒫› ‹y ∈ src.𝒫› ‹φ x = φ y›
  shows ‹∃(σ :: 'p1 ⇒ 'p1). particular_struct_endomorphism Γ1  σ ∧ σ x = y ∧ σ
y = y›
  using choice_from_choice_ex[OF assms choice_exists] by blast

end

end
```

C.4 Structural Properties of UFO Particulars

C.4.1 Isomorphically Unique Particulars

```
theory IsomorphicalUniqueness
  imports ParticularStructureMorphisms
begin

context ufo_particular_theory_sig
begindefinition isomorphically_unique_particulars (‹𝒫≃!›)
  where ‹𝒫≃! ≡ { x . x ∈ 𝒫 ∧ (
          ∀φ ∈ BijMorphs1Γ,TYPE(ZF). ∀σ ∈ MorphsΓ,MorphImg φ Γ.
          ∀y ∈ 𝒫.
          σ y = φ x ⟷ y = x)}›

lemma isomorphically_unique_particulars_I[intro!]:
  assumes ‹x ∈ 𝒫›
    ‹⋀φ σ y. ⟦ φ ∈ BijMorphs1Γ,TYPE(ZF) ; σ ∈ MorphsΓ,MorphImg φ Γ
            ; y ∈ 𝒫 ⟧
      ⟹ σ y = φ x ⟷ y = x›
  shows ‹x ∈ 𝒫≃!›
  using assms by (auto simp: isomorphically_unique_particulars_def)

lemma isomorphically_unique_particulars_E[elim!]:
  assumes ‹x ∈ 𝒫≃!›
  obtains ‹x ∈ 𝒫›
```

```
‹⋀φ σ y. ⟦ φ ∈ BijMorphs1_{Γ,TYPE(ZF)} ; σ ∈ Morphs_{Γ,MorphImg φ Γ} ; y ∈ 𝒫 ⟧
  ⟹ σ y = φ x ⟷ y = x›
using assms by (auto simp: isomorphically_unique_particulars_def)

lemma isomorphically_unique_particulars_are_particulars: ‹𝒫_{≃!} ⊆ 𝒫›
  by auto

end

end
```

C.4.2 Permutability

```
theory Permutability
  imports ParticularStructureMorphisms
begincontext ufo_particular_theory_sig
begindefinition non_permutable :: ‹'p ⇒ bool› where
  ‹non_permutable x ⟷ (∀φ ∈ EndoMorphs_Γ.∀y ∈ 𝒫. φ y = x ⟷ y = x)›


lemma non_permutable_I[intro!]:
  assumes ‹⋀φ y. ⟦ φ ∈ EndoMorphs_Γ ; y ∈ 𝒫 ⟧ ⟹ φ y = x ⟷ y = x›
  shows ‹non_permutable x›
  using assms
  by (auto simp: non_permutable_def)

lemma non_permutable_E[elim]:
  assumes ‹non_permutable x› ‹φ ∈ EndoMorphs_Γ› ‹y ∈ 𝒫›
  shows ‹φ y = x ⟷ y = x›
  using assms
  by (auto simp: non_permutable_def)

lemmas non_permutable_iff[simp] = non_permutable_def definition non_permutables (‹𝒫_{1!}›)
where
  ‹𝒫_{1!} ≡ { x . x ∈ 𝒫 ∧ non_permutable x }›

lemma non_permutables_I[intro!]:
  assumes ‹x ∈ 𝒫› ‹non_permutable x›
  shows ‹x ∈ 𝒫_{1!}›
  using assms by (auto simp: non_permutables_def)

lemma non_permutables_E[elim!]:
```

```
  assumes ‹x ∈ 𝒫₁!›
  obtains ‹x ∈ 𝒫› ‹non_permutable x›
  using assms by (auto simp: non_permutables_def)

lemma non_permutables_iff[simp]:
  ‹x ∈ 𝒫₁! ⟷ x ∈ 𝒫 ∧ non_permutable x›
  by (auto simp: non_permutables_def)

end


end
theory CollapsableParticulars
  imports ParticularStructureMorphisms
begin

context ufo_particular_theory_sig
begin
```

C.4.3 Collapsability

```
definition collapsable :: ‹'p ⇒ bool› where
  ‹collapsable x ≡ x ∈ 𝒫 ∧ (∃φ ∈ EndoMorphsΓ.∃y ∈ 𝒫. x ≠ y ∧ φ x = φ y)›

lemma collapsableI[intro]:
  assumes ‹x ∈ 𝒫› ‹y ∈ 𝒫› ‹φ ∈ EndoMorphsΓ› ‹x ≠ y› ‹φ x = φ y›
  shows ‹collapsable x›
  using assms by (auto simp: collapsable_def)

lemma collapsableE[elim]:
  assumes ‹collapsable x›
  obtains φ y where ‹x ∈ 𝒫› ‹y ∈ 𝒫›
      ‹φ ∈ EndoMorphsΓ› ‹x ≠ y› ‹φ x = φ y›
  using assms by (auto simp: collapsable_def)

lemmas collapsable_iff[simp] = collapsable_defdefinition nonCollapsableParticulars
(‹𝒫nc›) where
  ‹𝒫nc ≡ { x . x ∈ 𝒫 ∧ ¬ collapsable x }›

lemma nonCollapsableParticularsI[intro!]:
  assumes ‹x ∈ 𝒫› ‹¬ collapsable x›
  shows ‹x ∈ 𝒫nc›
```

```
  using assms by (auto simp: nonCollapsableParticulars_def)

lemma nonCollapsableParticularsE[elim!]:
  assumes ‹x ∈ 𝒫nc›
  obtains ‹x ∈ 𝒫› ‹¬ collapsable x›
  using assms by (auto simp: nonCollapsableParticulars_def)

lemma nonCollapsableParticulars_iff[simp]:
  ‹x ∈ 𝒫nc ⟷ x ∈ 𝒫 ∧ ¬ collapsable x›
  by (auto simp: nonCollapsableParticulars_def)

lemma nonCollapsableParticularsAreParticulars: ‹𝒫nc ⊆ 𝒫›
  by blast

end

end
```

C.4.4 Permutability and Isomorphical Uniqueness

```
theory StructuralPropertiesTheorems
  imports IsomorphicalUniqueness Permutability CollapsableParticulars
begin

context ufo_particular_theory
beginlemma im_unique_particulars_are_non_permutable_particulars:
    ‹𝒫≃! ⊆ 𝒫1!›
proof (intro subsetI ballI ; clarsimp simp: isomorphically_unique_particulars_def)
  fix x φ y
  assume as1: ‹particular_struct_endomorphism Γ φ› ‹x ∈ ℰ›
      ‹∀φ∈BijMorphs1Γ,TYPE(ZF). ∀σ∈MorphsΓ,MorphImg φ Γ.∀y∈ℰ. (σ y = φ x) = (y
= x)›
      ‹y ∈ ℰ›
  interpret I: particular_struct_endomorphism ‹Γ› ‹φ› using as1(1) by blast
  have A: ‹σ y = φ x ⟷ y = x› if ‹φ ∈ BijMorphs1Γ,TYPE(ZF)› ‹σ ∈ MorphsΓ,MorphImg φ Γ›
‹y ∈ ℰ› for φ σ y
    using that as1(3) by metis
  obtain π :: ‹'p ⇒ ZF› where ‹inj π›
    using injection_to_ZF_exist by blast
  have pi_isomorph: ‹π ∈ BijMorphs1Γ,TYPE(ZF)›
    using ‹inj π› by (meson UNIV_I inj_morph_img_BijMorphs inj_on_id inj_on_subset
subsetI)
```

```
  have pi_sigma_morph: ‹π ∘ φ ∈ Morphs_{Γ,MorphImg π Γ}›
    by (meson I.particular_struct_morphism_axioms bijections1_are_morphisms morphisms_are_clos
morphs_I pi_isomorph)
  have ‹π x = π (φ x)›
    using A[of ‹π› ‹π ∘ φ›, OF pi_isomorph pi_sigma_morph ‹x ∈ ℰ›,simplified]
    by simp
  then have ‹x = φ x›
    by (rule ‹inj π›[THEN inj_onD]; simp)
  then show ‹φ y = x ⟷ y = x›
    apply (intro iffI ; simp)
    by (rule A[where y=y and φ = π and σ = ‹π ∘ φ›,OF _ _ ‹y ∈ ℰ›,simplified o_apply
          , THEN iffD1] ; (intro pi_sigma_morph pi_isomorph)? ; simp)
qed

lemma non_permutable_particulars_are_im_unique_particulars:
  ‹𝒫_{1!} ⊆ 𝒫_{≃!}›
proof (intro subsetI ballI ; clarsimp simp: isomorphically_unique_particulars_def)
  fix x y :: ‹'p› and φ σ :: ‹'p ⇒ 'p_1› and f :: ‹'p_1 ⇒ ZF›
  assume as:
      ‹particular_struct_morphism Γ (MorphImg φ Γ) σ›
      ‹∀φ∈EndoMorphs_Γ. ∀y∈ℰ. φ y = x ⟷ y = x›
      ‹inj_on φ ℰ›
      ‹inj f›
      ‹x ∈ ℰ›
      ‹y ∈ ℰ›
  have A: ‹φ y = x ⟷ y = x› if ‹φ ∈ EndoMorphs_Γ› ‹y ∈ ℰ› for φ y using as(2) that
by blast
  interpret I1: particular_struct_morphism ‹Γ› ‹MorphImg φ Γ› ‹σ› using as(1) by simp
  interpret I: particular_struct_bijection_1 ‹Γ› ‹φ›
    using as(3,4) inj_morph_img_isomorphism[of ‹φ›] by blast
  interpret Inv: particular_struct_bijection_1 ‹MorphImg φ Γ› ‹I.inv_morph›
    using particular_struct_bijection_iff_particular_struct_bijection_1 by blast

  have B: ‹particular_struct_morphism Γ Γ (I.inv_morph ∘ σ)›
    apply (intro particular_struct_morphism_comp[of _ ‹MorphImg φ Γ›])
    subgoal using I1.particular_struct_morphism_axioms by blast
    by (simp add: particular_struct_bijection.axioms(1) particular_struct_injection.axioms(1))

  have C: ‹I.inv_morph ∘ σ ∈ EndoMorphs_Γ›
    apply (simp)
    apply (intro_locales)
   using B particular_struct_morphism_def
```

```
      pre_particular_struct_morphism_def by blast+

  have D: ‹I.inv_morph (σ x) = x› using A[OF C ‹x ∈ ℰ›,simplified] .
  have E: ‹I.inv_morph (φ x) = x› using ‹x ∈ ℰ› by simp
  have F: ‹I.inv_morph (σ x) = I.inv_morph (φ x)› using D E by simp
  show ‹σ y = φ x ⟷ y = x›
    apply (intro iffI ; simp?)
    subgoal
      apply (rule A[where y=y and φ = ‹I.inv_morph ∘ σ›,OF C ‹y ∈ ℰ›,THEN iffD1])
      by (simp add: E)
    subgoal
      apply (rule F inj_onD[OF Inv.morph_is_injective,simplified,OF F])
      subgoal using ‹x ∈ ℰ› by (simp add: I1.morph_preserves_particulars)
      using ‹x ∈ ℰ› by (simp add: I.morph_preserves_particulars)
    done
qed

theorem non_permutable_particulars_are_the_unique_particulars:
    ‹𝒫₁! = 𝒫≃!›
  using im_unique_particulars_are_non_permutable_particulars
     non_permutable_particulars_are_im_unique_particulars by simptheorem  non_permutable
‹𝒫₁! ⊆ 𝒫nc›
proof (intro subsetI ; elim non_permutables_E ;
        intro nonCollapsableParticularsI notI ;
        (elim collapsableE)? ; assumption?
        ; rename_tac x₁ φ x₂)
  fix x₁ x₂ φ
  assume A: ‹x₁ ∈ ℰ› ‹x₂ ∈ ℰ› ‹x₁ ≠ x₂› ‹φ x₁ = φ x₂›
    and B: ‹non_permutable x₁› ‹φ ∈ EndoMorphsΓ›
  note A(1,2)[simp,intro!]
  note A1[simp] = A(3)[simplified A(4)] A(4)
  interpret phi: particular_struct_endomorphism Γ φ using B(2) by simp
  note C1 = B(1)[THEN non_permutable_E,OF B(2) ‹x₂ ∈ ℰ›] and
       C2 = B(1)[THEN non_permutable_E,OF B(2) ‹x₁ ∈ ℰ›]
  have D[simp]: ‹φ x₁ = x₁› using C2 by simp
  show False
    using A1(1) A1(2) C1 D by presburger
qed


end
```

end

C.5 Examples of UFO Particular Structures and Morphisms

C.5.1 Single Non-Empty World Structures

```
theory SingleWorldStructure
  imports ParticularStructureMorphisms
begin

locale single_world_particular_struct =
    particular_struct where Typ_p = Typ_p and Typ_q = Typ_q
  for Typ_p :: ‹'p itself› and Typ_q :: ‹'q itself› +
  assumes
    only_empty_and_another_world: ‹∃w ≠ ∅. 𝒲 = {∅,w}›
begin

definition ‹w ≡ THE w. w ≠ ∅ ∧ w ∈ 𝒲›

lemma sw_non_empty_world_unique: ‹∃!w. w ≠ ∅ ∧ w ∈ 𝒲›
  using only_empty_and_another_world
  by (metis empty_iff insert_iff)

lemma single_world[simp,intro!]: ‹w ∈ 𝒲› ‹w ≠ ∅›
  by (simp only: w_def ; rule the1I2[OF sw_non_empty_world_unique] ; simp)+

lemma sw_worlds_case[cases set]:
  assumes ‹w ∈ 𝒲›
  obtains (empty) ‹w = ∅› | (single) ‹w = w›
  using assms only_empty_and_another_world single_world
  by blast

lemma sw_in_worlds[dest!]: ‹⟦ w ∈ 𝒲 ; x ∈ w ⟧ ⟹ w = w ∧ x ∈ w›
  using sw_worlds_case by auto

lemma sw_particulars_simp[simp]: ‹endurants = w›
  apply (simp only: 𝒫_def)
  by blast

lemma sw_worlds_simp: ‹𝒲 = {∅, w}›
  using only_empty_and_another_world single_world(2) by blast
```

```
lemma sw_particulars_I[iff]: ‹x ∈ 𝒫 ⟷ x ∈ w›
  by (simp only: sw_particulars_simp)

lemma sw_ed_iff[simp]: ‹ed x y ⟷ x ∈ w ∧ y ∈ w›
  apply (simp only: ed_def ; intro iffI conjI ballI impI ; simp?)
  by blast

lemma sw_no_indep[simp]: ‹¬ indep x y›
  by (simp add: indep_def)


lemma sw_empty_world[simp,intro!]: ‹∅ ∈ 𝒲›
  by (simp add: sw_worlds_simp)

end

locale single_world_pre_particular_struct_morphism =
   pre_particular_struct_morphism where Typ_p1 = ‹Typ_p1› and Typ_p2 = ‹Typ_p2› and Typ_q
= ‹Typ_q› +
   src: single_world_particular_struct where Γ = ‹Γ_1› and Typ_p = ‹Typ_p1› and Typ_q
= ‹Typ_q› +
   tgt: single_world_particular_struct where Γ = ‹Γ_2› and Typ_p = ‹Typ_p2› and Typ_q
= ‹Typ_q›
  for
    Typ_p1 :: ‹'p_1 itself› and
    Typ_p2 :: ‹'p_2 itself› and
    Typ_q :: ‹'q itself›
begin

lemma sw_world_corresp_iff[simp]:
  ‹w_s ⇔ w_t ⟷ (w_s = ∅ ∧ w_t = ∅) ∨
      (w_s = src.w ∧ w_t = tgt.w ∧ φ ` src.w ⊆ tgt.w)›
  apply (intro iffI conjI disjCI
        ; (elim conjE disjE)?
        ; simp
        ; (elim world_corresp_E)?
        ; (intro world_corresp_I)?
        ; simp?)
  subgoal by blast
  subgoal by (metis Set.subset_empty  empty_is_image image_subsetI morph_preserves_partic

              src.sw_in_worlds src.sw_particulars_simp src.sw_worlds_case
```

```
            tgt.sw_particulars_simp tgt.sw_worlds_case)
  subgoal by blast
  done

lemma sw_particular_struct_morphism_iff:
  ‹particular_struct_morphism Γ1 Γ2 φ ⟷
    φ ` src.w ⊆ tgt.w ›
    (is ‹?P ⟷ ?Q›)
proof
  assume ‹?P›
  then interpret phi: particular_struct_morphism Γ1 Γ2 φ by simp
  show ‹?Q›
    apply (intro conjI ballI impI subsetI ; (elim conjE imageE)? ; simp)
    using morph_preserves_particulars by blast
next
  assume A: ‹?Q›
  show ‹?P›
    apply (unfold_locales ; simp)
    subgoal G1 for ws
      apply (cases ws rule: src.sw_worlds_case ; simp)
       apply (intro exI[of _ ∅] ; simp)
      using A by linarith
    subgoal G2 for wt
      apply (cases wt rule: tgt.sw_worlds_case ; simp)
      apply (intro exI[of _ ∅] ; simp)
      using A by linarith
    done
qed

end

locale single_world_particular_struct_morphism =
    single_world_pre_particular_struct_morphism
      where Typp1 = ‹Typp1› and Typp2 = ‹Typp2› and Typq = ‹Typq›
  for
    Typp1 :: ‹'p1 itself› and
    Typp2 :: ‹'p2 itself› and
    Typq :: ‹'q itself›  +
  assumes
    image_subset: ‹φ ` src.w ⊆ tgt.w› and
    morph_closes_quale_assoc_1:
      ‹⋀xt yt q. ⟦ xt ∈ φ ` src.w ; yt ∈ tgt.w ; q ∈ 𝒬s ; yt ◁t xt ; yt ⇝t q ⟧
```

```
            ⟹ y_t ∈ φ ` src.w⟩

sublocale single_world_particular_struct_morphism ⊆ particular_struct_morphism
  apply (simp only: sw_particular_struct_morphism_iff)
  using image_subset morph_closes_quale_assoc_1 by metis

end
```

C.6 Twin Spheres Examples

```
theory TwinSpheres
  imports SingleWorldStructure
begindatatype quale =
    Q_Dist200M
  | Q_Radius10M
  | Q_Radius20Mdatatype endurant =
      Sphere1
    | Sphere2
    | Sphere1Radius10M
    | Sphere2Radius20M
    | Sphere1Dist200M
    | Sphere2Dist200Mdefinition tw_worlds ::
    ⟨endurant set ⇒ endurant set set⟩
  where
    ⟨tw_worlds X = {∅,X}⟩definition tw_inheres_in ::
    ⟨endurant set ⇒ endurant ⇒ endurant ⇒ bool⟩
  where
    ⟨tw_inheres_in X y x ⟷
       x ∈ X ∧ y ∈ X ∧
       (x = Sphere1 ∧ (y = Sphere1Radius10M ∨ y = Sphere1Dist200M) ∨
        x = Sphere2 ∧ (y = Sphere2Radius20M ∨ y = Sphere2Dist200M))⟩

lemma tw_inheres_in_rtrancl_simp:
  ⟨(tw_inheres_in X)** x y ⟷ x = y ∨ tw_inheres_in X x y⟩
  apply (intro iffI conjI ; (elim conjE)?)
  subgoal G1
    apply (cases ⟨x = y⟩)
    subgoal x_eq_y by simp
    subgoal x_neq_y premises P
      using rtranclpD[of ⟨tw_inheres_in X⟩ x y,OF P(1),simplified P(2),simplified]
      apply (simp add: P(2))
```

```
      by (induct rule: tranclp_induct ; (simp add: tw_inheres_in_def)?
          ; safe)
    done
  by blastdefinition tw_assoc_quale ::
    ‹endurant set ⇒ endurant ⇒ quale ⇒ bool›
  where
    ‹tw_assoc_quale X x q ⟷ x ∈ X ∧
        (x = Sphere1Radius10M ∧ q = Q_Radius10M ∨
         x = Sphere1Dist200M ∧ q = Q_Dist200M ∨
         x = Sphere2Radius20M ∧ q = Q_Radius20M ∨
         x = Sphere2Dist200M ∧ q = Q_Dist200M) ›definition tw_towards ::
  ‹endurant set ⇒ endurant ⇒ endurant ⇒ bool›
    where
  ‹tw_towards X x y ⟷
     x ∈ X ∧ y ∈ X ∧
     (x = Sphere1Dist200M ∧ y = Sphere2 ∨
      x = Sphere2Dist200M ∧ y = Sphere1)›definition tw_all_quality_spaces ::
  ‹quale set set›
    where
  ‹tw_all_quality_spaces ≡
    { { Q_Dist200M},
      {Q_Radius10M,Q_Radius20M} }›definition tw_quality_spaces ::
  ‹endurant set ⇒ quale set set›
    where
  ‹tw_quality_spaces X =
     { Q | Q x q . Q ∈ tw_all_quality_spaces ∧
                   tw_assoc_quale X x q ∧ q ∈ Q}›definition tw_part_structure ::

  ‹endurant set ⇒ (endurant,quale) particular_struct›
    where
  ‹tw_part_structure X ≡ ⦇
      ps_quality_spaces = tw_quality_spaces X,
      ps_worlds = tw_worlds X,
      ps_inheres_in = tw_inheres_in X,
      ps_assoc_quale = tw_assoc_quale X,
      ps_towards = tw_towards X ⦈›definition ‹all_endurants ≡
       {Sphere1,Sphere1Radius10M,Sphere1Dist200M,
        Sphere2,Sphere2Radius20M,Sphere2Dist200M}›definition
  ‹just_the_substantials ≡ {Sphere1,Sphere2}›definition ‹conf1 ≡ tw_part_structure
all_endurants›definition ‹conf2 ≡
    tw_part_structure (
      all_endurants - {Sphere1Radius10M, Sphere2Radius20M}
```

```
    )›definition ‹conf3 ≡
    tw_part_structure just_the_substantials
  ›

lemmas all_defs =
  conf1_def conf2_def conf3_def
  tw_part_structure_def
  tw_quality_spaces_def
  tw_worlds_def
  tw_inheres_in_def
  tw_assoc_quale_def
  tw_towards_def
  tw_all_quality_spaces_def
  possible_worlds_sig.ed_def
  possible_worlds_sig.𝒫_def
  all_endurants_def
  just_the_substantials_def
  inherence_sig.ultimateBearer_def
  qualified_particulars_sig.qualifiedParticulars_def

  possible_worlds_sig.indep_def
  inherence_sig.order_def

lemma tw_inheres_in_bounded: ‹tw_inheres_in X = tw_inheres_in (X ∩ all_endurants)›
  apply (intro ext)
  by (simp only: tw_inheres_in_def all_endurants_def; auto)

lemma tw_inheres_in_scope: ‹tw_inheres_in X x y ⟹ x ∈ all_endurants ∧ y ∈ all_endurar
  by (simp only: tw_inheres_in_def all_endurants_def; auto)

lemma tw_inheres_in_wf[intro!,simp]: ‹wfP (tw_inheres_in X)›
proof (intro finite_acyclic_wf[to_pred])
  have ‹{(y, x). tw_inheres_in X y x} ⊆ all_endurants × all_endurants›
    by (auto dest: tw_inheres_in_scope)
  then show ‹finite {(y, x). tw_inheres_in X y x}›
    apply (rule finite_subset)
    by (simp add: all_endurants_def)
  show ‹acyclicP (tw_inheres_in X)›
    apply (intro acyclicI[to_pred] allI notI)
    subgoal for y
      apply (cases y ; simp add: tw_inheres_in_def)
      subgoal using converse_tranclpE by fastforce
```

```
      subgoal using converse_tranclpE by fastforce
      subgoal using tranclp.cases by fastforce
      subgoal using tranclp.cases by fastforce
      subgoal using tranclp.cases by fastforce
      subgoal using tranclp.cases by fastforce
      done
    done
qed

lemma tw_inheres_in_noetherian[intro!,simp]: ‹wfP (tw_inheres_in X)⁻¹⁻¹›
proof (intro finite_acyclic_wf[to_pred] ; simp)
  have ‹{(y, x). tw_inheres_in X x y} ⊆ all_endurants × all_endurants›
    by (auto dest: tw_inheres_in_scope)
  then show ‹finite {(x, y). tw_inheres_in X y x}›
    apply (rule finite_subset)
    by (simp add: all_endurants_def)
  show ‹acyclicP (λx y. tw_inheres_in X y x)›
    apply (intro acyclicI[to_pred] allI notI)
    subgoal for y
      apply (cases y ; simp add: tw_inheres_in_def)
      subgoal using tranclp.cases by fastforce
      subgoal using tranclp.cases by fastforce
      subgoal using converse_tranclpE by fastforce
      subgoal using converse_tranclpE by fastforce
      subgoal using converse_tranclpE by fastforce
      subgoal using converse_tranclpE by fastforce
      done
    done
qed

fun endurant_enum :: ‹endurant ⇒ nat› where
  ‹endurant_enum Sphere1 = 0› |
  ‹endurant_enum Sphere2 = 1› |
  ‹endurant_enum Sphere1Radius10M = 2› |
  ‹endurant_enum Sphere1Dist200M = 3› |
  ‹endurant_enum Sphere2Radius20M = 4› |
  ‹endurant_enum Sphere2Dist200M = 5›

lemma endurant_enum_inj: ‹inj endurant_enum›
  apply (intro inj_onI  ; simp)
  subgoal for x y
    by (cases x ; cases y ; simp)
```

```
  done

lemma endurant_inj_to_zf_ex[intro!,simp]: ‹∃(f :: endurant ⇒ ZF). inj f›
proof -
  obtain f :: ‹nat ⇒ ZF› where ‹inj f›
    using inj_nat2Nat by auto
  then have ‹inj (f ∘ endurant_enum)›
    apply (intro comp_inj_on[OF endurant_enum_inj,of f])
    using inj_on_subset by blast
  then show ?thesis by blast
qed

interpretation conf1: particular_struct_defs ‹conf1› ‹TYPE(endurant)› ‹TYPE(quale)›
.

interpretation conf1: possible_worlds ‹conf1.𝒲› ‹TYPE(endurant)›
  apply (unfold_locales)
  subgoal G1 using endurant_inj_to_zf_ex .
  subgoal G2 by (auto simp: all_defs)
  subgoal G3 by (auto simp: all_defs)
  done

interpretation conf1: inherence ‹conf1.𝒲› ‹conf1.inheresIn› ‹TYPE(endurant)›
  apply (unfold_locales)
  subgoal G1 by (auto simp: all_defs)
  subgoal G2 by (auto simp: all_defs)
  subgoal G3 by (auto simp: all_defs)
  subgoal G4 by (auto simp: all_defs)
  subgoal G5 by (auto simp: all_defs)
  done

lemma conf1_moments: ‹conf1.ℳ = {Sphere1Radius10M,Sphere1Dist200M,
                      Sphere2Radius20M,Sphere2Dist200M}›
  by (auto simp: all_defs inherence_sig.ℳ_def)

lemma conf1_substantials: ‹conf1.𝒮 = {Sphere1, Sphere2}›
  by (auto simp: all_defs inherence_sig.𝒮_def conf1_moments inherence_sig.ℳ_def)

lemma conf1_qualia: ‹conf1.qualia = {Q_Radius10M,Q_Radius20M,Q_Dist200M}›
  by (auto simp: all_defs quality_space_sig.qualia_def)

interpretation conf1: qualified_particulars
```

```
      ‹conf1.𝒲› ‹conf1.inheresIn› ‹conf1.𝒬𝒮› ‹conf1.assoc_quale›
      ‹TYPE(endurant)› ‹TYPE(quale)›
  apply (unfold_locales)
  subgoal G1 by (auto simp: all_defs)
  subgoal G2 by (auto simp: all_defs)
  subgoal G3
    apply (simp only: conf1_moments conf1_qualia)
    by (auto simp: all_defs)
  subgoal G4 by (auto simp: all_defs)
  subgoal G5 for w y1 y2 x q1 q2 Q

    apply (cases y1 ; simp ; cases y2 ; simp ; cases x ; simp add: all_defs)
    by (elim disjE conjE ; simp ; hypsubst_thin)+
  subgoal G6 by (auto simp: all_defs)
  subgoal G7 by (auto simp: all_defs)
  done

lemma conf1_ultimate_bearers[simp]:
  assumes ‹x ∈ conf1.𝒫›
  shows
  ‹conf1.ultimateBearer x =
    (if x = Sphere1Radius10M then Sphere1 else
     if x = Sphere1Dist200M then Sphere1 else
     if x = Sphere2Radius20M then Sphere2 else
     if x = Sphere2Dist200M then Sphere2 else x)›

  apply (cases x ; simp ; intro conf1.ultimate_bearer_eq_simp[THEN iffD2] conjI
        ; (simp add: conf1_substantials conf1.endurants_eq_un_moments_subst
                 conf1_moments)?)
  by (simp add: conf1_def tw_part_structure_def all_endurants_def
           tw_inheres_in_def tw_inheres_in_rtrancl_simp)+

lemma conf1_directed_moments[simp]:
  ‹conf1.directed_moments = {Sphere1Dist200M,Sphere2Dist200M}›
  by (auto simp: all_defs towardness_sig.directed_moments_def)

interpretation conf1: towardness
      ‹conf1.𝒲› ‹conf1.inheresIn› ‹conf1.towards› ‹TYPE(endurant)›
  apply (unfold_locales)
  subgoal G1 for x y
    by (intro conjI  ; cases x; cases y ; simp add: all_defs inherence_sig.ℳ_def
inherence_sig.𝒮_def)
```

```
  subgoal G2
    by (simp add: conf1.ed_def ; auto simp: all_defs)
  subgoal G3 premises P for x y
    apply (rule G2[OF P,THEN conf1.edE])
    subgoal premises Q
      apply (simp only: Q(1,2)[THEN conf1_ultimate_bearers])
      apply (cases x ; simp ; cases y ; simp)
      using P Q by (auto simp: all_defs)
    done
  subgoal G4 by (auto simp: all_defs)
  done

interpretation conf1: ufo_particular_theory
      ‹conf1.𝒲› ‹conf1.inheresIn› ‹conf1.𝒬𝒮› ‹conf1.assoc_quale› ‹conf1.towards›
      ‹TYPE(endurant)› ‹TYPE(quale)›
  apply (unfold_locales)
  by (simp add: conf1.qualifiedParticulars_def
      ; auto simp: all_defs)theorem conf1_particular_struct[simp,intro!]:
    ‹particular_struct conf1›
  by intro_locales

interpretation conf1: particular_struct conf1 ‹TYPE(endurant)› ‹TYPE(quale)›
  by simp



interpretation conf2: particular_struct_defs ‹conf2› ‹TYPE(endurant)› ‹TYPE(quale)›
.

interpretation conf2: possible_worlds ‹conf2.𝒲› ‹TYPE(endurant)›
  apply (unfold_locales)
  subgoal G1 using endurant_inj_to_zf_ex .
  subgoal G2 by (auto simp: all_defs)
  subgoal G3 by (auto simp: all_defs)
  done

interpretation conf2: inherence ‹conf2.𝒲› ‹conf2.inheresIn› ‹TYPE(endurant)›
  apply (unfold_locales)
  subgoal G1 by (auto simp: all_defs)
  subgoal G2 by (auto simp: all_defs)
  subgoal G3 by (auto simp: all_defs)
  subgoal G4 by (auto simp: all_defs)
```

```
  subgoal G5 by (auto simp: all_defs)
  done

lemma conf2_moments: ‹conf2.𝓜 = {Sphere1Dist200M,Sphere2Dist200M}›
  by (auto simp: all_defs inherence_sig.𝓜_def)

lemma conf2_qualia: ‹conf2.qualia = {Q_Dist200M}›
  by (auto simp: all_defs quality_space_sig.qualia_def)

interpretation conf2: qualified_particulars
      ‹conf2.𝓦› ‹conf2.inheresIn› ‹conf2.𝓠𝓢› ‹conf2.assoc_quale›
      ‹TYPE(endurant)› ‹TYPE(quale)›
  apply (unfold_locales)
  subgoal G1 by (auto simp: all_defs)
  subgoal G2 by (auto simp: all_defs)
  subgoal G3
    apply (simp only: conf2_moments conf2_qualia)
    by (auto simp: all_defs)
  subgoal G4 by (auto simp: all_defs)
  subgoal G5 for w y1 y2 x q1 q2 Q

    apply (cases y1 ; simp ; cases y2 ; simp ; cases x ; simp add: all_defs)
    done
  subgoal G6 by (auto simp: all_defs)
  subgoal G7 by (auto simp: all_defs)
  done

lemma conf2_substantials: ‹conf2.𝓢 = {Sphere1, Sphere2}›
  by (auto simp: all_defs inherence_sig.𝓢_def conf2_moments inherence_sig.𝓜_def)

lemma conf2_ultimate_bearers[simp]:
  assumes ‹x ∈ conf2.𝓟›
  shows
  ‹conf2.ultimateBearer x =
    (if x = Sphere1Dist200M then Sphere1 else
     if x = Sphere2Dist200M then Sphere2 else x)›
  using assms[simplified conf2_substantials conf2.endurants_eq_un_moments_subst
              conf2_moments]
  by (cases x ; simp ;
        intro conf2.ultimate_bearer_eq_simp[THEN iffD2] ;
        (simp add: conf2_substantials conf2.endurants_eq_un_moments_subst
              conf2_moments )? ;
```

```
        intro tranclp_into_rtranclp tranclp.intros(1);
        simp add: conf2_def all_endurants_def tw_part_structure_def tw_inheres_in_def
        )

lemma conf2_directed_moments[simp]:
  ‹conf2.directed_moments = {Sphere1Dist200M,Sphere2Dist200M}›
  by (auto simp: all_defs towardness_sig.directed_moments_def)

interpretation conf2: towardness
      ‹conf2.𝒲› ‹conf2.inheresIn› ‹conf2.towards› ‹TYPE(endurant)›
  apply (unfold_locales)
  subgoal G1 for x y
    by (intro conjI  ; cases x; cases y ; simp add: all_defs inherence_sig.ℳ_def
inherence_sig.𝒮_def)
  subgoal G2
    by (simp add: conf2.ed_def ; auto simp: all_defs)
  subgoal G3 premises P for x y
    apply (rule G2[OF P,THEN conf2.edE])
    subgoal premises Q
      apply (simp only: Q(1,2)[THEN conf2_ultimate_bearers])
      apply (cases x ; simp ; cases y ; simp)
      using P Q by (auto simp: all_defs)
    done
  subgoal G4 by (auto simp: all_defs)
  done

interpretation conf2: ufo_particular_theory
      ‹conf2.𝒲› ‹conf2.inheresIn› ‹conf2.𝒬𝒮› ‹conf2.assoc_quale› ‹conf2.towards›
      ‹TYPE(endurant)› ‹TYPE(quale)›
  apply (unfold_locales)
  by (simp add: conf2.qualifiedParticulars_def
      ; auto simp: all_defs)

theorem conf2_particular_struct[simp,intro!]:
    ‹particular_struct conf2›
  by intro_locales

interpretation conf2: particular_struct conf2 ‹TYPE(endurant)› ‹TYPE(quale)›
  by simp
```

```
interpretation conf3: particular_struct_defs ‹conf3› ‹TYPE(endurant)› ‹TYPE(quale)›
.

interpretation conf3: possible_worlds ‹conf3.𝒲› ‹TYPE(endurant)›
  apply (unfold_locales)
  subgoal G1 using endurant_inj_to_zf_ex .
  subgoal G2 by (auto simp: all_defs)
  subgoal G3 by (auto simp: all_defs)
  done

interpretation conf3: inherence ‹conf3.𝒲› ‹conf3.inheresIn› ‹TYPE(endurant)›
  apply (unfold_locales)
  subgoal G1 by (auto simp: all_defs)
  subgoal G2 by (auto simp: all_defs)
  subgoal G3 by (auto simp: all_defs)
  subgoal G4 by (auto simp: all_defs)
  subgoal G5 by (auto simp: all_defs)
  done

lemma conf3_moments: ‹conf3.ℳ = ∅›
  by (auto simp: all_defs inherence_sig.ℳ_def)

lemma conf3_qualia: ‹conf3.qualia = ∅›
  by (auto simp: all_defs quality_space_sig.qualia_def)

interpretation conf3: qualified_particulars
      ‹conf3.𝒲› ‹conf3.inheresIn› ‹conf3.𝒬𝒮› ‹conf3.assoc_quale›
      ‹TYPE(endurant)› ‹TYPE(quale)›
  apply (unfold_locales)
  subgoal G1 by (auto simp: all_defs)
  subgoal G2 by (auto simp: all_defs)
  subgoal G3
    apply (simp only: conf3_moments conf3_qualia)
    by (auto simp: all_defs)
  subgoal G4 by (auto simp: all_defs)
  subgoal G5 for w y1 y2 x q1 q2 Q

    apply (cases y1 ; simp ; cases y2 ; simp ; cases x ; simp add: all_defs)
    done
  subgoal G6 by (auto simp: all_defs)
  subgoal G7 by (auto simp: all_defs)
  done
```

```
lemma conf3_ultimate_bearers[simp]:
  assumes ‹x ∈ conf3.𝒫›
  shows ‹conf3.ultimateBearer x = x›
  using assms[simplified conf3_def just_the_substantials_def
              tw_part_structure_def,simplified
              possible_worlds_sig.𝒫_def,
              simplified,
              simplified tw_worlds_def,
              simplified]
  by (rule disjE ; simp ; intro conf3.ultimate_bearer_eq_simp[THEN iffD2]
          ; simp add: conf3_def just_the_substantials_def
            tw_part_structure_def possible_worlds_sig.𝒫_def
            inherence_sig.𝒮_def tw_worlds_def
            inherence_sig.ℳ_def
            tw_inheres_in_def )

lemma conf3_directed_moments[simp]:
  ‹conf3.directed_moments = ∅›
  by (auto simp: all_defs towardness_sig.directed_moments_def)

interpretation conf3: towardness
      ‹conf3.𝒲› ‹conf3.inheresIn› ‹conf3.towards› ‹TYPE(endurant)›
  apply (unfold_locales)
  subgoal G1
    by (simp only: conf3_moments ; auto simp: all_defs)
  subgoal G2
    by (simp add: conf3.ed_def ; auto simp: all_defs)
  subgoal G3 premises P for x y
    apply (rule G2[OF P,THEN conf3.edE])
    subgoal premises Q
      apply (simp only: Q(1,2)[THEN conf3_ultimate_bearers])
      apply (cases x ; simp ; cases y ; simp)
      using P Q by (auto simp: all_defs)
    done
  subgoal G4 by (auto simp: all_defs)
  done

interpretation conf3: ufo_particular_theory
      ‹conf3.𝒲› ‹conf3.inheresIn› ‹conf3.𝒬𝒮› ‹conf3.assoc_quale› ‹conf3.towards›
      ‹TYPE(endurant)› ‹TYPE(quale)›
  apply (unfold_locales)
```

```
  by (simp add: conf3.qualifiedParticulars_def
      ; auto simp: all_defs)

interpretation conf3: particular_struct conf3 ‹TYPE(endurant)› ‹TYPE(quale)›
  by intro_locales

theorem conf3_particular_struct[simp,intro!]:
  ‹particular_struct conf3›
  by intro_locales

interpretation conf3: particular_struct conf3 ‹TYPE(endurant)› ‹TYPE(quale)›
  by simp

end
```

APPENDIX D – Individuality and Identifiability

D.1 Individuality

```
theory Individuality
  imports "../ParticularStructures/CollapsableParticulars"
begincontext ufo_particular_theory_sig
begin

abbreviation particularsWithIndividuality (‹𝒫_ind›) where
  ‹𝒫_ind ≡ 𝒫_nc›lemmas particularsWithIndividualityI = nonCollapsableParticularsI

lemmas particularsWithIndividualityE = nonCollapsableParticularsE

lemmas particularsWithIndividuality_iff = nonCollapsableParticulars_iff

lemmas substantialsWithIndividualitySubset =
  nonCollapsableParticularsAreParticulars

end

end
```

D.2 Identity Through Permutability

D.2.1 Isomorphically-Invariant Identity Predicates

```
theory IdentityPredicate
  imports "../ParticularStructures/ParticularStructureMorphisms" "HOL-ZF.MainZF"
begindefinition identity_pred ::
  ‹('p,'q) particular_struct ⇒
   ((ZF,'q) particular_struct ⇒ ZF ⇒ bool) ⇒ 'p ⇒ bool›
  where ‹
    identity_pred Γ P x ⟷ x ∈ particulars Γ ∧
      BijMorphs1_{Γ,TYPE(ZF)} ≠ ∅ ∧
      (∀Γ' ∈ IsoModels_{Γ,TYPE(ZF)}.
        ∀φ ∈ Morphs_{Γ,Γ'}.
```

```
        ∀y. P Γ' y ⟷
          (∀z ∈ particulars Γ. y = φ z ⟷ z = x))›

lemma identity_pred_I:
  assumes
    ‹BijMorphs1_{Γ,TYPE(ZF)} ≠ ∅›
    ‹⋀Γ' φ y. ⟦
          Γ' ∈ IsoModels_{Γ,TYPE(ZF)} ;
          φ ∈ Morphs_{Γ,Γ'}
        ⟧ ⟹  P Γ' y ⟷ (∀z ∈ particulars Γ. y = φ z ⟷ z = x)›
    ‹x ∈ particulars Γ›
  shows ‹identity_pred Γ P x›
  using assms
  by (auto simp: identity_pred_def)

lemma identity_pred_I':
  assumes
    ‹x ∈ particulars Γ›
    ‹BijMorphs1_{Γ,TYPE(ZF)} ≠ ∅›
      ‹⋀Γ' φ y. ⟦
          Γ' ∈ IsoModels_{Γ,TYPE(ZF)} ;
          φ ∈ Morphs_{Γ,Γ'}
        ⟧ ⟹  P Γ' y ⟷ (∀z ∈ particulars Γ. y = φ z ⟷ z = x)›
  shows ‹identity_pred Γ P x›
  using assms
  by (auto simp: identity_pred_def)

lemma identity_pred_E:
  assumes
    ‹identity_pred Γ P x›
  obtains φ where
    ‹particular_struct Γ›
    ‹x ∈ particulars Γ›
    ‹φ ∈ BijMorphs1_{Γ,TYPE(ZF)}›
    ‹⋀Γ' φ y. ⟦
          Γ' ∈ IsoModels_{Γ,TYPE(ZF)} ;
          φ ∈ Morphs_{Γ,Γ'}
        ⟧ ⟹  P Γ' y ⟷ (∀z ∈ particulars Γ. y = φ z ⟷ z = x)›
proof -
  obtain φ where A:
    ‹φ ∈ BijMorphs1_{Γ,TYPE(ZF)}›
    using assms
```

```
    by (auto simp: identity_pred_def)
  have B: ‹particular_struct Γ› using A(1)
    by (meson bijections1_are_morphisms morphs_D particular_struct_morphism.axioms(1)
pre_particular_struct_morphism_def)
  show ‹?thesis›
    using B assms that
    by (auto simp: identity_pred_def)
qed

lemma identity_pred_E':
  assumes ‹identity_pred Γ P x›
  obtains
    ‹BijMorphs1_{Γ,TYPE(ZF)} ≠ ∅›
    ‹x ∈ particulars Γ›
    ‹⋀Γ' φ y. ⟦
          Γ' ∈ IsoModels_{Γ,TYPE(ZF)} ;
          φ ∈ Morphs_{Γ,Γ'}
        ⟧ ⟹  P Γ' y ⟷ (∀z ∈ particulars Γ. y = φ z ⟷ z = x)›
  using assms
  by (simp add: identity_pred_def)


lemma id_is_unique_1:
  fixes Γ :: ‹('p,'q) particular_struct› and σ :: ‹'p ⇒ ZF›
  assumes
    ‹identity_pred Γ P_1 x›
    ‹identity_pred Γ P_2 x›
    ‹particular_struct_bijection Γ Γ' φ›
    and σ: ‹inj σ›
  shows ‹P_1 (MorphImg σ Γ') y = P_2 (MorphImg σ Γ') y›
proof -
  have sigma[dest!,simp]: ‹σ x = σ y ⟹ x = y› for x y
    using injD σ by metis
  have Gamma1[simp]: ‹Γ' = MorphImg φ Γ›
    using assms(3)
    by (meson isomorphism_iff_isomorphism_to_morphimg)
  interpret I1: particular_struct_bijection_1 ‹Γ› ‹φ›
    using assms(3)
    by (simp add: particular_struct_bijection_iff_particular_struct_bijection_1)
  have I1_src_gamma[simp]: ‹I1.src.Γ = Γ›  by simp
  interpret IS2:  particular_struct ‹MorphImg (σ ∘ φ) I1.src.Γ›
    apply (intro I1.src.inj_morph_img_valid_structure)
```

```
    subgoal G1
      apply (intro comp_inj_on)
      subgoal G1_1 by simp
      subgoal G2_2
        apply (auto simp: possible_worlds_sig.𝒫_def)
        by (intro inj_onI ; simp)
      done
    subgoal G2
      by (rule exI[of _ ‹id›] ; simp)
    done
  have simp1[simp]: ‹particular_struct (MorphImg σ (MorphImg φ Γ))›
    using IS2.particular_struct_axioms
    by (simp add: fcomp_def)
  have simp_worlds_img_img: ‹(σ ‘ φ ‘ w₁ = σ ‘ φ ‘ w₂) ⟷ (w₁ = w₂)› if ‹w₁ ∈ I1.src.𝒲›
‹w₂ ∈ I1.src.𝒲› for w₁ w₂
    by (metis I1.phi_inv_phi_world σ inj_image_eq_iff that(1) that(2))
  have I1_tgt_gamma: ‹I1.tgt.Γ = MorphImg φ Γ›
    using I1.tgt_Gamma_eq_Morph_img by auto
  interpret sigma: particular_struct_bijection_1 ‹MorphImg φ Γ› σ
    apply (intro I1.tgt.inj_morph_img_isomorphism[simplified I1_tgt_gamma])
    subgoal by (simp add: inj_onI)
    by (simp add: IS2.injection_to_ZF_exist)

  have sigma_phi: ‹σ ∘ φ ∈ BijMorphs1Γ,TYPE(ZF)›
    by (intro bijections_I particular_struct_bijection_1_comp
        I1.particular_struct_bijection_1_axioms
        sigma.particular_struct_bijection_1_axioms
          )

  have sigma_phi_img: ‹MorphImg (σ ∘ φ) Γ ∈ IsoModelsΓ,TYPE(ZF)›
    using sigma_phi by blast

  have sigma_phi_morph: ‹σ ∘ φ ∈ MorphsΓ,MorphImg (σ ∘ φ) Γ›
    by (meson bijections1_are_morphisms sigma_phi)

  show ‹?thesis›
  proof (simp)
    fix y
    obtain P₁:
              ‹⋀Γ' φ y. ⟦
          Γ' ∈ IsoModelsΓ,TYPE(ZF) ;
          φ ∈ MorphsΓ,Γ'
```

```
        ⟧ ⟹  P1 Γ' y ⟷ (∀z ∈ particulars Γ. y = φ z ⟷ z = x)›
      using identity_pred_E[OF assms(1)] by metis
    obtain P2:
              ‹⋀Γ' φ y. ⟦
          Γ' ∈ IsoModels_Γ,TYPE(ZF) ;
          φ ∈ Morphs_Γ,Γ'
        ⟧ ⟹  P2 Γ' y ⟷ (∀z ∈ particulars Γ. y = φ z ⟷ z = x)›
      using identity_pred_E[OF assms(2)] by metis
    note Q1 = P1[OF sigma_phi_img,of ‹σ ∘ φ›,OF sigma_phi_morph]
    note Q2 = P2[OF sigma_phi_img,of ‹σ ∘ φ›,OF sigma_phi_morph]
    show ‹P1 (MorphImg σ (MorphImg φ Γ)) y = P2 (MorphImg σ (MorphImg φ Γ)) y›
      apply (intro iffI)
      subgoal using Q1[of ‹y›] Q2(1) by simp
      subgoal using Q2[of ‹y›] Q1(1) by simp
      done
  qed
qed


lemma morph_img_comp_1[simp]: ‹MorphImg φ1 ∘ MorphImg φ2 = MorphImg (φ1 ∘ φ2)›
  by (intro ext ; simp)


context ufo_particular_theory
beginlemma identity_respects_isomorphisms:
  assumes
    ‹identity_pred Γ P x›
    ‹φ ∈ BijMorphs1_Γ,TYPE('p1)›
  shows ‹identity_pred (MorphImg φ Γ) P (φ x)›
proof -
  obtain φ1 where
      A: ‹particular_struct Γ›
        ‹φ1 ∈ BijMorphs1_Γ,TYPE(ZF)›
    ‹⋀Γ' φ y. ⟦
          Γ' ∈ IsoModels_Γ,TYPE(ZF) ;
          φ ∈ Morphs_Γ,Γ'
        ⟧ ⟹  P Γ' y ⟷ (∀z ∈ particulars Γ. y = φ z ⟷ z = x)›
      ‹x ∈ particulars Γ›
    using identity_pred_E[OF assms(1)] by metis
  have x_E[simp,intro!]: ‹x ∈ ℰ› using A(4) by simp
  interpret I1: particular_struct ‹Γ› using A(1) by simp
  interpret I2: particular_struct_bijection_1 ‹Γ› ‹φ› using assms(2) by blast
```

```
  have B: ‹I2.tgt.Γ = MorphImg φ Γ›
    apply (intro particular_struct_eqI)
    by (simp_all only: I2.tgt.Γ_def particular_struct.simps)
  obtain f :: ‹'p1 ⇒ ZF› where f: ‹inj f› using I2.tgt.injection_to_ZF_exist by
blast
  have ‹f ∈ BijMorphs1MorphImg φ Γ,TYPE(ZF)›
    apply (auto)
    apply (intro I2.tgt.inj_morph_img_isomorphism[simplified B])
    subgoal by (meson f injD inj_onI)
    using inj_on_id by blast
  then have C: ‹BijMorphs1MorphImg φ Γ,TYPE(ZF) ≠ ∅› by blast
  have D: ‹φ' ∘ φ ∈ BijMorphs1Γ,TYPE(ZF)› if
    as: ‹φ' ∈ BijMorphs1MorphImg φ Γ,TYPE(ZF)› for φ'
  proof -
    interpret II1: particular_struct_bijection_1 ‹MorphImg φ Γ› ‹φ'›
      using as by blast
    show ‹?thesis›
      apply (intro bijections_I)
      apply (rule particular_struct_bijection_1_comp)
      subgoal
        by (simp only: I2.particular_struct_bijection_1_axioms)
      using II1.particular_struct_bijection_1_axioms by blast
  qed
  show ‹?thesis›
  proof (intro identity_pred_I C)
    show ‹φ x ∈ I2.tgt.endurants›
      using A(4) by blast
    fix Γ' φ' y'
    assume as3: ‹Γ' ∈ IsoModelsMorphImg φ Γ,TYPE(ZF)›
                ‹φ' ∈ MorphsMorphImg φ Γ,Γ'›
    then have AA: ‹φ' ∘ φ ∈ MorphsΓ,Γ'›
      using I2.particular_struct_morphism_axioms particular_struct_morphism_comp
by blast
    interpret phi'_phi: particular_struct_morphism ‹Γ› ‹Γ'› ‹φ' ∘ φ›
      using AA[THEN morphs_D] by simp

    obtain φ1 where phi1: ‹φ1 ∈ BijMorphs1MorphImg φ Γ,TYPE(ZF)› ‹Γ' = MorphImg φ1
(MorphImg φ Γ)›
      using as3(1) by blast
    have BB: ‹Γ' = MorphImg (φ1 ∘ φ) Γ› using phi1(2) by simp
    have CC: ‹φ1 ∘ φ ∈ BijMorphs1Γ,TYPE(ZF)›
      apply (simp ; safe)
```

```
      subgoal
        apply (rule comp_inj_on)
        subgoal using I2.morph_is_injective by auto
        by (metis I2.morph_is_surjective particular_struct_bijection_1_def particular_str
bijections_D phi1(1) ufo_particular_theory_sig.Γ_simps(2))
      using inj_on_id by blast
    interpret phi1_phi: particular_struct_bijection_1 Γ ‹φ₁ ∘ φ› using CC by blast

    have Gamma'_isomodel: ‹Γ' ∈ IsoModels_{Γ,TYPE(ZF)}›
      by (intro isomorphic_models_I[of ‹φ₁ ∘ φ›] BB CC)

    have phi1_comp_phi_morphs[simp,intro!]: ‹φ₁ ∘ φ ∈ Morphs_{Γ,Γ'}›
      using BB CC bijections1_are_morphisms by fastforce

    have ex_morph: ‹∃φ∈BijMorphs1_{Γ,TYPE(ZF)}. Γ' = MorphImg φ Γ›
      using BB CC by blast

    show ‹P Γ' y' = (∀z ∈ I2.tgt.endurants. (y' = φ' z) = (z = φ x))›
    proof (intro iffI ballI  ; simp?)
      fix z
      assume as4: ‹P Γ' (φ' z)› ‹z ∈ I2.tgt.𝒫› ‹y' = φ' z›

      have inv_z_dom[simp]: ‹I2.inv_morph z ∈ ℰ›
        using I2.phi_inv_scope as4(2) by auto
      have inv_z[simp]: ‹φ (I2.inv_morph z) = z›
        by (simp add: as4(2))
      note R1= A(3)[THEN iffD1, rule_format,OF _ phi1_comp_phi_morphs
                  ,where z=‹I2.inv_morph z›,simplified
                  ,OF ex_morph as4(1),simplified]

      note R2= A(3)[THEN iffD1, rule_format,OF _ AA,simplified
                , OF ex_morph as4(1),of ‹I2.inv_morph z›,
                  simplified]
      have R3: ‹φ' z = φ₁ z›  using R1 R2 by simp
      note as4_1 =  as4[simplified R3]
      show ‹z = φ x›
        using as4_1 A(3)[THEN iffD1,OF _ _ as4_1(1),rule_format
              , simplified,OF ex_morph,of _ ]
        using R2 by auto
    next
      fix z
      assume as4: ‹P Γ' y'› ‹φ x ∈ I2.tgt.𝒫› ‹z = φ x›
```

```
      note R1 = A(3)[simplified,THEN iffD1,OF ex_morph,rule_format
                  ,OF _ as4(1)]
      show ‹y' = φ' (φ x)›
        using R1 A(3)
        using AA Gamma'_isomodel as4(1) by auto
    next
      assume ‹∀z ∈ I2.tgt.𝒫. y' = φ' z ⟷ z = φ x›
      then have as4: ‹y' = φ' z ⟷ z = φ x› if ‹z ∈ I2.tgt.𝒫› for z
        using that by metis
      note R1 = A(3)[THEN iffD2,of Γ' _ y',rule_format,simplified,
                OF ex_morph,of ‹φ' ∘ φ›,simplified,
                OF phi'_phi.particular_struct_morphism_axioms]
      note R2 = A(3)[THEN iffD2,of Γ' _ y',rule_format,simplified,
                OF ex_morph,of ‹φ1 ∘ φ›,simplified
                , simplified
                 phi1_phi.particular_struct_morphism_axioms[simplified,
                   simplified phi1(2)[symmetric]],
                 simplified]
      note R3 = as4[simplified  I2.morph_is_surjective[symmetric]]
      note R1 R2 R3[of ‹φ z›,simplified]

      note I2.morph_is_surjective[symmetric]
      have R5: ‹φ z = φ x ⟷ z = x› if ‹z ∈ ℰ› for z
        using that ‹x ∈ ℰ› I2.morph_is_injective inj_onD
        by (metis Γ_simps(2))
      show ‹P Γ' y'›
        apply (intro R1 )
        subgoal premises P for z
          apply (rule as4[of ‹φ z›,simplified R5[OF P]])
          using P
          by (simp add: I2.morph_preserves_particulars)
        done
    qed
  qed
qed

end

context ufo_particular_theory
begin

lemma identity_pred:
```

```
  fixes
     P x and φ :: ‹'p ⇒ ZF›
  assumes ‹identity_pred Γ P x› ‹inj_on φ 𝒫›
  shows ‹P (MorphImg φ Γ) y ⟷ y = φ x›
proof -
  obtain A:
    ‹⋀Γ' φ y.
      ⟦ Γ' ∈ IsoModels_{Γ,TYPE(ZF)} ;
        φ ∈ Morphs_{Γ,Γ'} ⟧ ⟹
       P Γ' y ⟷ (∀z∈particulars Γ. (y = φ z) = (z = x))›
    ‹x ∈ particulars Γ›
    using identity_pred_E[OF assms(1)] by metis
  have B: ‹φ ∈ BijMorphs1_{Γ,TYPE(ZF)}›
    apply (simp ; intro conjI)
    subgoal using assms(2)[THEN inj_on_subset,simplified] by simp
    using inj_on_id by blast
  then interpret I: particular_struct_bijection_1 ‹Γ› ‹φ› by simp
  have C: ‹MorphImg φ Γ ∈ IsoModels_{Γ,TYPE(ZF)}›
    by (intro isomorphic_models_I[of ‹φ›] B ; simp)
  have D: ‹φ ∈ Morphs_{Γ,MorphImg φ Γ}›
    using I.particular_struct_morphism_axioms by blast
  show ‹?thesis›
    apply (intro iffI ; simp add: A(1)[OF C D,of y])
    subgoal
      using assms(1) identity_pred_E' by fastforce
    subgoal
      by (metis C D I.morph_is_injective ‹⋀thesis. (⟦⋀Γ' φ y. ⟦Γ' ∈ IsoModels_{Γ,TYPE(ZF)};
φ ∈ Morphs_{Γ,Γ'}⟧ ⟹ P Γ' y = (∀z∈I.src.endurants. (y = φ z) = (z = x)); x ∈ I.src.endur
⟹ thesis) ⟹ thesis› inj_onD)
    done
qed

lemma identity_pred_I1:
  fixes P and x
  assumes
    ‹x ∈ ℰ›
    ‹⋀Γ' φ y. ⟦
          Γ' ∈ IsoModels_{Γ,TYPE(ZF)} ;
          φ ∈ Morphs_{Γ,Γ'}
         ⟧ ⟹  P Γ' y ⟷ (∀z ∈ ℰ. y = φ z ⟷ z = x)›
  shows ‹identity_pred Γ P x›
proof -
```

```
  have A: ‹BijMorphs1Γ,TYPE(ZF) ≠ ∅›
  proof -
    obtain φ :: ‹'p ⇒ ZF› where ‹inj φ› using injection_to_ZF_exist by blast
    then have ‹inj_on φ 𝒫›
      using inj_on_subset ‹inj φ› by blast
    have ‹φ ∈ BijMorphs1Γ,TYPE(ZF)›
      apply (intro ‹inj_on φ 𝒫› inj_morph_img_BijMorphs)
      using  inj_on_id by blast
    then show ‹?thesis› by blast
  qed

  show ‹?thesis›
    apply (intro identity_pred_I[OF A])
    using assms by auto
qed


lemma identity_pred_eqI:
  assumes ‹identity_pred Γ P x›
    ‹σ ∈ BijMorphs1Γ,TYPE(ZF)› ‹φ ∈ MorphsΓ,MorphImg σ Γ›
  shows ‹P (MorphImg σ Γ) y ⟷ (∀z ∈ ℰ. y = φ z ⟷ z = x)›
proof -
  have ps[simp]: ‹particulars Γ = ℰ› by simp
  obtain A: ‹x ∈ ℰ›
    ‹⋀Γ' φ y. ⟦
          Γ' ∈ IsoModelsΓ,TYPE(ZF) ;
          φ ∈ MorphsΓ,Γ'
        ⟧ ⟹  P Γ' y ⟷ (∀z ∈ ℰ. y = φ z ⟷ z = x)›
    using assms(1)[THEN identity_pred_E,simplified ps] by metis
  have B: ‹MorphImg σ Γ ∈ IsoModelsΓ,TYPE(ZF)› using assms(2) by blast
  show ‹?thesis›
    using A B assms(3)
    by blast
qed

end


end
```

D.2.2 Identifiable Particulars

```
theory Identifiability
  imports IdentityPredicate
begincontext ufo_particular_theory_sig
begin

definition identifiables (‹𝒫=›) where
  ‹𝒫= ≡ { x | x P . x ∈ 𝒫 ∧ identity_pred Γ P x }›

lemma identifiables_I[intro]:
  assumes ‹x ∈ 𝒫› ‹identity_pred Γ P x›
  shows ‹x ∈ 𝒫=›
  using assms by (auto simp: identifiables_def)

lemma identifiables_E[elim!]:
  assumes ‹x ∈ 𝒫=›
  obtains P where ‹x ∈ 𝒫› ‹identity_pred Γ P x›
  using assms by (auto simp: identifiables_def)

endend
```

D.2.3 Identifiability, Isomorphical Uniqueness and Permutability

```
theory IdentifiabilityAndIsomorphicalUniqueness
  imports Identifiability
  "../ParticularStructures/StructuralPropertiesTheorems"
begincontext ufo_particular_theory
begin

lemma ex1_intro_1:
  fixes A and P
  assumes ‹f ∈ A› ‹⋀g. g ∈ A ⟹ g a = f a›
  shows ‹∃!z. ∀g ∈ A. g a = z›
  apply (intro ex1I[of _ ‹f a›] ballI)
  subgoal using assms by blast
  subgoal premises P for z
    using P[rule_format,OF assms(1)]
    by simp
  donelemma  im_uniques_are_identifiables: ‹𝒫≃! ⊆ 𝒫=›
proof (intro subsetI)
  fix x
  assume as: ‹x ∈ 𝒫≃!›
```

```
obtain ‹x ∈ ℰ› using as by blast

have A: ‹σ x = φ x›
  if ‹inj_on φ ℰ›
     ‹particular_struct_morphism Γ (MorphImg φ Γ) σ›
  for φ  σ :: ‹'p ⇒ ZF›
  apply (rule as[THEN isomorphically_unique_particulars_E])
  subgoal premises P
    apply (rule P(2)[simplified,OF conjI
              ,OF that(1)[simplified] _ that(2) ‹x ∈ ℰ›,
              THEN iffD2])
    using inj_on_id by blast+
  done

have pick_ex: ‹∃!y. ∀φ' ∈ Morphs_{Γ,Γ'}. φ' x = y›
  if as1: ‹Γ' ∈ IsoModels_{Γ,TYPE(ZF)}› for Γ'
proof -
  obtain φ1 where phi1:
      ‹φ1 ∈ BijMorphs1_{Γ,TYPE(ZF)}›
      ‹Γ' = MorphImg φ1 Γ›
    using as1 by blast
  interpret phi1: particular_struct_bijection_1 ‹Γ› ‹φ1› using phi1 by simp
  interpret phi1_inv: particular_struct_bijection_1 ‹MorphImg φ1 Γ› ‹phi1.inv_morph›
    using particular_struct_bijection_iff_particular_struct_bijection_1 by blast

  have AA: ‹φ1 ∈ Morphs_{Γ,Γ'}›
    apply (auto)
    using phi1(2) phi1.particular_struct_morphism_axioms
    by blast

  have BB: ‹φ' x = φ1 x› if as2: ‹φ' ∈ Morphs_{Γ,Γ'}› for φ'
    apply (rule A)
    subgoal using phi1.morph_is_injective[simplified] .
    using phi1(2) as2 morphs_D by metis

  show ‹?thesis›
    apply (rule ex1_intro_1[of ‹φ1›])
    subgoal using AA .
    subgoal premises P for φ'
      using BB[OF P] .
    done
qed
```

```
  show ‹x ∈ 𝒫=›
  proof (clarsimp simp:  identifiables_def ‹x ∈ ℰ›)

    define P where ‹P Γ' z ⟷ z = (THE y. ∀φ' ∈ Morphs_{Γ,Γ'}. φ' x = y)› for Γ'
and z :: ‹ZF›

    have P1: ‹P Γ' (φ' x)›
      if ‹∃φ∈BijMorphs1_{Γ,TYPE(ZF)}. Γ' = MorphImg φ Γ› ‹φ' ∈ Morphs_{Γ,Γ'}› for Γ' φ'
      using pick_ex[THEN the1I2,of ‹Γ'› ‹λy. ∀φ' ∈ Morphs_{Γ,Γ'}. φ' x = y›,
        simplified P_def[symmetric],simplified] that by metis
    have P2: ‹P (MorphImg φ Γ) (φ' x)›
      if ‹φ ∈ BijMorphs1_{Γ,TYPE(ZF)}› ‹φ' ∈ Morphs_{Γ, MorphImg φ Γ}› for φ φ'
      using P1 that by metis

    have P3: ‹y = φ x›
      if as3: ‹σ ∈ BijMorphs1_{Γ,TYPE(ZF)}› ‹φ ∈ Morphs_{Γ,MorphImg σ Γ}›
              ‹P (MorphImg σ Γ) y›  for σ φ y
      apply (rule the1_equality[of ‹λy. ∀φ' ∈ Morphs_{Γ,MorphImg σ Γ}. φ' x = y›,of
‹φ x›,
              simplified as3(3)[simplified P_def,symmetric],OF pick_ex]
              ; (intro ballI)?)
      subgoal G1 using that(1) by blast
      subgoal for φ3
        apply auto
        by (metis G1 morphs_iff pick_ex that(2))
      done

    show ‹∃P. identity_pred Γ P x›
      apply (intro exI[of _ ‹P›] identity_pred_I1 iffI )
      subgoal by (simp add: ‹x ∈ ℰ›)
      subgoal by (metis (full_types) P3 as isomorphic_models_E ufo_particular_theory_sig.

      using P2 ‹x ∈ ℰ› by auto
  qed
qed

lemma identifiables_are_im_uniques: ‹𝒫= ⊆ 𝒫≃!›
proof (intro subsetI)
  fix x
  assume ‹x ∈ 𝒫=›
  then obtain P where P: ‹x ∈ ℰ› ‹identity_pred Γ P x› by blast
```

```
  show ‹x ∈ 𝒫≃!›
  proof (rule)
    show ‹x ∈ ℰ› using P(1) .
    fix φ σ y
    assume as: ‹φ ∈ BijMorphs1Γ,TYPE(ZF)› ‹σ ∈ MorphsΓ,MorphImg φ Γ› ‹y ∈ ℰ›
    have P1: ‹identity_pred (MorphImg φ Γ) P (φ x)›
      using identity_respects_isomorphisms[OF P(2) as(1)] .
    interpret phi: particular_struct_bijection_1 ‹Γ› ‹φ›
      using as(1) by blast
    have A: ‹phi.tgt.Γ = MorphImg φ Γ›
      using phi.tgt_Gamma_eq_Morph_img by auto
    have B: ‹φ ∈ MorphsΓ,MorphImg φ Γ›
      using as(1) by (meson bijections1_are_morphisms)
    note C = identity_pred_eqI[
              where φ = ‹φ› and x = ‹x› and P = ‹P› and σ = ‹φ›,
                OF P(2) as(1) B]
            identity_pred_eqI[
              where φ = ‹σ› and x = ‹x› and P = ‹P› and σ = ‹φ›,
                OF P(2) as(1,2)]
    have D: ‹P (MorphImg φ Γ) (σ x) ⟷ P (MorphImg φ Γ) (φ x)›
      by (metis C(1) C(2) P(1))
    have E: ‹φ x = σ x ⟷ P (MorphImg φ Γ) (φ x)›
      using C(2)[of ‹φ x›]
      using P(1) P(2) identity_pred phi.morph_is_injective by auto
    have E1: ‹MorphImg φ Γ ∈ IsoModelsΓ,TYPE(ZF)›
      using as(1) by blast
    obtain F: ‹⋀Γ' φ y. Γ' ∈ IsoModelsΓ,TYPE(ZF) ⟹ φ ∈ MorphsΓ,Γ' ⟹ P Γ' y
= (∀z∈phi.src.endurants. (y = φ z) = (z = x))›
      using P(2)[THEN identity_pred_E] by metis
    note G = F[OF E1 as(2)]
    note H = C(1)[simplified C(2)]
    note J = H[THEN iffD1,simplified Ball_def,simplified,rule_format]
             H[THEN iffD2,simplified Ball_def,simplified,rule_format]
    show ‹σ y = φ x ⟷ y = x›
      apply (intro iffI)
      subgoal using G P(2) as(3) identity_pred phi.particular_struct_bijection_1_axioms
by auto
      subgoal using J by (metis E P(2) identity_pred phi.morph_is_injective ufo_particular_th
      done
  qed
qed
```

theorem *identifiables_are_the_im_uniques:* ‹$\mathcal{P}_{=}$ = $\mathcal{P}_{\simeq!}$›
  using *im_uniques_are_identifiables*
    identifiables_are_im_uniques
  by *blast***theorem** *identifiables_are_the_non_permutables:* ‹$\mathcal{P}_{=}$ = $\mathcal{P}_{1!}$›
  using *identifiables_are_the_im_uniques*
  by *(simp add: non_permutable_particulars_are_the_unique_particulars)*

end

end

D.3 Identity Characterization Equivalences

theory *Identity*
  imports *Correspondence "../ParticularStructures/StructuralPropertiesTheorems"*
  Individuality AnchoringAndIdentifiability

begin

context *ufo_particular_theory*
beginlemma *identifiable_particulars_are_individuals:*
    ‹$\mathcal{P}_{=} \subseteq \mathcal{P}_{ind}$›
  by *(simp add: identifiables_are_the_non_permutables non_permutables_are_non_collapsable*

theorem *identifiable_particulars_are_the_non_permutables:*
    ‹$\mathcal{P}_{=}$ = $\mathcal{P}_{1!}$›
  using *identifiables_are_the_non_permutables* **.**

theorem *identifiable_particulars_are_the_isomorphically_unique:*
    ‹$\mathcal{P}_{=}$ = $\mathcal{P}_{\simeq!}$›
  using *identifiables_are_the_im_uniques* **.**

theorem *identifiable_particulars_are_the_anchored_particulars:*
    ‹$\mathcal{P}_{=}$ = $\mathcal{P}_{\downarrow}$›
  by *(simp add: anchored_are_the_non_permutable identifiables_are_the_im_uniques*

non_permutable_particulars_are_the_unique_particulars)

end

end

D.3.1 Correspondence Function

theory *Correspondence*
  imports *IdentifiabilityAndIsomorphicalUniqueness "../ParticularStructures/StructuralProperti*
beginlocale *isomorphic_pair_of_particular_structures_sig* =
    src: particular_struct_defs **where** Γ = ‹Γ_1› **and** Typ_p = ‹Typ_{p1}› **and** Typ_q = ‹Typ_q›
+
    tgt: particular_struct_defs **where** Γ = ‹Γ_2› **and** Typ_p = ‹Typ_{p2}› **and** Typ_q = ‹Typ_q›
  for Γ_1 :: ‹('p_1,'q) particular_struct› **and**
      Γ_2 :: ‹('p_2,'q) particular_struct› **and**
      Typ_{p1} :: ‹'p_1 itself› **and**
      Typ_{p2} :: ‹'p_2 itself› **and**
      Typ_q :: ‹'q itself›
begindefinition π :: ‹'p_1 $\Rightarrow$ 'p_2› **where**
  ‹$\pi \equiv$ SOME σ. particular_struct_bijection_1 Γ_1 σ $\wedge$ Γ_2 = MorphImg σ Γ_1›

end

context *particular_struct*
begindefinition Γ_corresp :: ‹'p $\Rightarrow$ ('p_1,'q) particular_struct $\Rightarrow$ 'p_1› **where**
 ‹Γ_corresp x Γ' $\equiv$
      let φ = (SOME σ. particular_struct_bijection_1 Γ σ $\wedge$ Γ' = MorphImg σ Γ)
      in φ x ›**end**

locale *isomorphic_pair_of_particular_structures* =
    isomorphic_pair_of_particular_structures_sig
    where Γ_1 = ‹Γ_1› **and** Γ_2 = ‹Γ_2› **and**
      Typ_{p1} = ‹Typ_{p1}› **and** Typ_{p2} = ‹Typ_{p2}› **and** Typ_q = ‹Typ_q› +
    src: particular_struct **where** Γ = ‹Γ_1› **and** Typ_p = ‹Typ_{p1}› **and** Typ_q = ‹Typ_q› +
    tgt: particular_struct **where** Γ = ‹Γ_2› **and** Typ_p = ‹Typ_{p2}› **and** Typ_q = ‹Typ_q›
  for Γ_1 :: ‹('p_1,'q) particular_struct› **and**
      Γ_2 :: ‹('p_2,'q) particular_struct› **and**
      Typ_{p1} :: ‹'p_1 itself› **and**
      Typ_{p2} :: ‹'p_2 itself› **and**
      Typ_q :: ‹'q itself› +
  assumes
    structs_are_isomorphic:
    ‹$\exists\sigma$. particular_struct_bijection_1 Γ_1 σ $\wedge$ Γ_2 = MorphImg σ Γ_1›
begin

```
lemma pi_isomorphism: ‹particular_struct_bijection_1 Γ1 π›
  and Gamma2_eq: ‹Γ2 = MorphImg π Γ1›
  using someI_ex[OF structs_are_isomorphic, simplified π_def[symmetric]]
  by auto

end

notation isomorphic_pair_of_particular_structures (infix ‹≃i› 75)

lemma  isomorphic_pair_of_particular_structures_refl[intro]:
  assumes ‹particular_struct Γ›
  shows ‹Γ ≃i Γ›
  apply (simp add: isomorphic_pair_of_particular_structures_def assms)
  apply unfold_locales
  apply (intro exI[of _ ‹id›] ; simp)
  by (metis assms particular_struct_def particular_struct_eqI
      ufo_particular_theory.id_is_isomorphism ufo_particular_theory_sig.Γ_simps)

lemma isomorphic_pair_of_particular_structures_sym[sym]:
  assumes ‹Γ1 ≃i Γ2›
  shows ‹Γ2 ≃i Γ1›
proof -
  interpret isomorphic_pair_of_particular_structures ‹Γ1› ‹Γ2›
    using assms .

  interpret pi: particular_struct_bijection_1 ‹Γ1› ‹π›
    using pi_isomorphism .

  show ‹?thesis›
    apply (intro_locales)
    apply (unfold_locales)
    apply (intro exI[of _ ‹pi.inv_morph›] conjI )
    subgoal using Gamma2_eq particular_struct_bijection_iff_particular_struct_bijection_1
pi.inv_is_bijective_morphism by auto
    by (metis Gamma2_eq particular_struct_bijection_iff_particular_struct_bijection_1

              pi.inv_is_bijective_morphism)
qed

lemma isomorphic_pair_of_particular_structures_trans[trans]:
  assumes ‹Γ1 ≃i Γ2› ‹Γ2 ≃i Γ3›
```

```
  shows ‹Γ1 ≃i Γ3›
proof -
  interpret P12: isomorphic_pair_of_particular_structures ‹Γ1› ‹Γ2›
    using assms(1) .
  interpret pi12: particular_struct_bijection_1 ‹Γ1› ‹P12.π›
    using P12.pi_isomorphism .
  interpret P23: isomorphic_pair_of_particular_structures ‹Γ2› ‹Γ3›
    using assms(2) .
  interpret pi23: particular_struct_bijection_1 ‹Γ2› ‹P23.π›
    using P23.pi_isomorphism .
  have A: ‹Γ3 = MorphImg P23.π (MorphImg P12.π Γ1)›
    using P12.Gamma2_eq P23.Gamma2_eq by auto
  interpret P13: particular_struct_bijection_1 ‹Γ1› ‹P23.π ∘ P12.π›
    apply (rule particular_struct_bijection_1_comp)
    subgoal using pi12.particular_struct_bijection_1_axioms by auto
    subgoal using P12.Gamma2_eq pi23.particular_struct_bijection_1_axioms by auto
    done

  show ‹?thesis›
    apply (unfold_locales)
    apply (intro exI[of _ ‹P23.π ∘ P12.π›] conjI P13.particular_struct_bijection_1_axioms

              ; (simp)?)
    using A by metis
qed


context ufo_particular_theory_sig
begin

definition correspondentIn :: ‹'p ⇒ ('p1,'q) particular_struct ⇒ 'p1›
  where ‹correspondentIn x Γ' ≡
  THE y. ∀σ ∈ Morphs_{Γ,Γ'}. ∀z ∈ particulars Γ. σ z = y ⟷ z = x›

end

context particular_struct_bijection_1
begin

lemma
  assumes ‹x ∈ src.identifiables›
          ‹φ1 ∈ Morphs_{src.Γ,MorphImg φ src.Γ}›
```

```
        ‹e ∈ src.𝒫›
  shows ‹φ1 e = src.correspondentIn x (MorphImg φ src.Γ) ⟷ e = x›
proof -
  have np_x: ‹src.non_permutable x›
    using assms(1) src.identifiables_are_the_non_permutables
    by auto
  note np_x_E = src.non_permutable_E[OF np_x]
  interpret phi1: particular_struct_morphism ‹src.Γ› ‹MorphImg φ src.Γ› φ1
    using assms by blast
  have phi1_tgt: ‹phi1.tgt.endurants = tgt.endurants›
      using tgt.Γ_simps(2) tgt_Gamma_eq_Morph_img by auto
  obtain P where P: ‹x ∈ src.𝒫› ‹identity_pred src.Γ P x›
    using assms(1) by blast
  obtain ω1 :: ‹'p1 ⇒ ZF› where ω1: ‹inj ω1› using src.injection_to_ZF_exist by
blast
  obtain ω2 :: ‹'p2 ⇒ ZF› where ω2: ‹inj ω2› using tgt.injection_to_ZF_exist by
blast
  have A: ‹ω1 ∈ BijMorphs1_{src.Γ,TYPE(ZF)}›
    apply (simp ; safe)
    subgoal using ω1 inj_on_subset by auto
    using inj_on_id by blast
  have tgt_G: ‹tgt.Γ = MorphImg φ src.Γ›
    using tgt_Gamma_eq_Morph_img by blast

  interpret omega1: particular_struct_bijection_1 ‹src.Γ› ω1 ‹TYPE('p1)› ‹TYPE(ZF)›
    apply (intro src.inj_morph_img_isomorphism)
    subgoal using ω1 inj_on_subset by auto
    using inj_on_id by blast

  interpret omega2: particular_struct_bijection_1 ‹MorphImg φ src.Γ› ω2 ‹TYPE('p2)›
‹TYPE(ZF)›
    apply (intro tgt.inj_morph_img_isomorphism[simplified tgt_G])
    subgoal using ω2 inj_on_subset by blast
    using inj_on_id by blast

  obtain Pi: ‹⋀Γ' φ y. Γ' ∈ IsoModels_{src.Γ,TYPE(ZF)} ⟹ φ ∈ Morphs_{src.Γ,Γ'} ⟹ P
Γ' y = (∀z∈phi1.src.endurants. (y = φ z) = (z = x))›
    using P(2)[THEN identity_pred_E] by metis
  have B: ‹ω1 ∘ φ⁻¹ ∘ φ1 ∈ Morphs_{src.Γ,MorphImg ω1 src.Γ}›
    apply (intro morphs_I
        particular_struct_morphism_comp[of _ ‹MorphImg φ src.Γ›]
        particular_struct_morphism_comp[of _ ‹src.Γ›])
```

```
    subgoal using assms(2) by blast
    subgoal
      by (metis particular_struct_eqI particular_struct_bijection.inv_is_bijective_morphism

            particular_struct_bijection_1_axioms
            particular_struct_bijection_def
            particular_struct_bijection_iff_particular_struct_bijection_1
            particular_struct_injection_def ufo_particular_theory_sig.Γ_simps)
    using A particular_struct_bijection_1_def particular_struct_injection_def by
blast

  have COND2: ‹MorphImg ω1 src.Γ ∈ IsoModels_src.Γ,TYPE(ZF)›
    using A by blast
  have COND3: ‹ω1 ∈ Morphs_src.Γ,MorphImg ω1 src.Γ›
    using omega1.particular_struct_morphism_axioms by blast
  have COND4: ‹∀z∈src.𝒫. (ω1 x = ω1 z) = (z = x)›
    using ω1[THEN injD] by auto
  note R1 = Pi[of ‹MorphImg ω1 src.Γ› _ ‹ω1 x›,OF COND2]
  have P_1: ‹P (MorphImg ω1 src.Γ) (ω1 x)›
    using R1[of ω1,OF COND3,simplified] COND4 by auto
  have P_2: ‹ω1 x = φ' z ⟷ z = x› if
    ‹φ' ∈ Morphs_src.Γ,MorphImg ω1 src.Γ›
    ‹z ∈ src.𝒫› for φ' z
    using that Pi[of ‹MorphImg ω1 src.Γ› _ ‹ω1 x›,simplified P_1 simp_thms,
        rule_format,OF COND2] by auto
  have P_3: ‹x = φ⁻¹ (φ1 z) ⟷ z = x› if AA: ‹z ∈ src.𝒫› for z
    using P_2[OF B,simplified,OF AA] injD[OF ω1] by auto



  interpret phi_inv: particular_struct_bijection_1 ‹MorphImg φ src.Γ› ‹φ⁻¹› ‹TYPE('p2)›
‹TYPE('p1)›
    apply (simp  only: tgt.isomorphism_1_iff_inj[simplified tgt_G] ; intro conjI)
    subgoal by simp
    using ω1 by blast

  have phi_inv_img_phi_img[simp]: ‹MorphImg φ⁻¹ (MorphImg φ src.Γ) = src.Γ›
    by (metis inv_is_bijective_morphism particular_struct_eqI particular_struct_bijection_iff_
src.Γ_simps)

  interpret phi_inv_phi: particular_struct_morphism src.Γ src.Γ ‹φ⁻¹ ∘ φ1› ‹TYPE('p1)›
‹TYPE('p1)›
```

```
    by (intro  particular_struct_morphism_comp[of _ ‹MorphImg φ src.Γ›]
        phi1.particular_struct_morphism_axioms
        phi_inv.particular_struct_morphism_axioms[simplified phi_inv_img_phi_img])

  interpret phi_inv_phi_auto: particular_struct_endomorphism src.Γ ‹φ⁻¹ ∘ φ₁›
    by (intro_locales)


  note ident_pred_I = src.identity_respects_isomorphisms[OF P(2),simplified tgt_G]
  have phi_inv_phi_auto_endurants[simp]: ‹phi_inv_phi_auto.endurants = src.𝒫›
    by auto
  let ?guess = ‹φ₁ x›
  have case1: ‹σ z = ?guess ⟷ z = x›
    if as: ‹σ ∈ Morphs_{src.Γ,MorphImg φ src.Γ}› ‹z ∈ omega1.src.endurants›
    for σ z
  proof -
    interpret sigma: particular_struct_morphism src.Γ ‹MorphImg φ src.Γ› σ
      using as(1) by blast
    note AA = as(2)[simplified phi_inv_phi_auto_endurants]

    interpret phi_inv_sigma: particular_struct_morphism src.Γ src.Γ ‹φ⁻¹ ∘ σ› ‹TYPE('p₁)›
‹TYPE('p₁)›
      by (intro particular_struct_morphism_comp[of _ ‹MorphImg φ src.Γ›]
                sigma.particular_struct_morphism_axioms
                phi_inv.particular_struct_morphism_axioms[simplified phi_inv_img_phi_img]
    interpret phi_inv_sigma_auto: particular_struct_endomorphism src.Γ ‹φ⁻¹ ∘ σ›
      by (intro_locales)
    have phi_inv_sigma_in_auto: ‹φ⁻¹ ∘ σ ∈ EndoMorphs_{src.Γ}›
      using phi_inv_sigma_auto.particular_struct_endomorphism_axioms by blast

    have x_const_phi1[simp]: ‹φ⁻¹ (φ₁ x) = x› using P_3[OF ‹x ∈ src.𝒫›] by simp
    have x_const_sigma[simp]: ‹φ⁻¹ (σ x) = x› using np_x_E[OF phi_inv_sigma_in_auto,simpl
‹x ∈ src.𝒫›] by simp
    have omega2_src_end[simp]: ‹omega2.src.endurants = tgt.𝒫›
      using phi1_tgt by blast
    obtain DD: ‹φ₁ z ∈ tgt.endurants› ‹σ z ∈ tgt.endurants›
      using ‹z ∈ src.𝒫›
      using phi1.morph_preserves_particulars phi1_tgt sigma.morph_preserves_particulars
by auto
    have BB[simp]: ‹φ₁ x = σ x›
      using x_const_phi1 x_const_sigma
          phi_inv.morph_is_injective[simplified omega2_src_end,THEN inj_onD
```

```
            , OF _ DD]
      by (metis I_img_eq_tgt_I P(1) morph_inv_morph_img phi1.morph_preserves_particulars
sigma.morph_preserves_particulars src.Γ_simps(2) tgt.Γ_simps(2) tgt_Gamma_eq_Morph_img)

    have EE: ‹φ⁻¹ (φ₁ z) = φ⁻¹ (σ z) ⟷ σ z = φ₁ z›
      using inv_morph_injective[THEN inj_onD,OF _ DD] by auto
    note ident_pred_I[of φ,
            THEN identity_pred_E]
    show ?thesis
      apply simp
      using AA np_x_E phi_inv_sigma_in_auto by force
  qed
  have case2: ‹∃!y. ∀σ∈Morphs_{src.Γ,MorphImg φ src.Γ}·
          ∀z∈omega1.src.endurants. (σ z = y) = (z = x)›
  proof (intro ex1I[of _ ‹?guess›] , metis case1)
    fix y
    assume
        ‹∀σ ∈ Morphs_{src.Γ,MorphImg φ src.Γ}·
         ∀z∈omega1.src.endurants. σ z = y ⟷ z = x›
    then have as: ‹σ z = y ⟷ z = x›
      if ‹σ ∈ Morphs_{src.Γ,MorphImg φ src.Γ}›
         ‹z ∈ src.𝒫› for σ z
      using that
      by simp

    show ‹y = φ₁ x›
      by (rule as[of φ₁ x,THEN iffD2,symmetric] ;
            (intro ‹x ∈ src.𝒫› assms(2))? ; simp)
  qed

  have case3: ‹σ z = φ x ⟷ z = x›
    if as: ‹σ ∈ Morphs_{src.Γ,MorphImg φ src.Γ}› ‹z ∈ omega1.src.endurants›
    for σ z
  proof -
    interpret sigma: particular_struct_morphism src.Γ ‹MorphImg φ src.Γ› σ
      using as(1) by blast
    note AA = as(2)[simplified phi_inv_phi_auto_endurants]

    interpret phi_inv_sigma: particular_struct_morphism src.Γ src.Γ ‹φ⁻¹ ∘ σ› ‹TYPE('p₁)›
‹TYPE('p₁)›
      by (intro particular_struct_morphism_comp[of _ ‹MorphImg φ src.Γ›]
               sigma.particular_struct_morphism_axioms
```

```
                phi_inv.particular_struct_morphism_axioms[simplified phi_inv_img_phi_img]
    interpret phi_inv_sigma_auto: particular_struct_endomorphism src.Γ ‹φ⁻¹ ∘ σ›
      by (intro_locales)
    have phi_inv_sigma_in_auto: ‹φ⁻¹ ∘ σ ∈ EndoMorphs_src.Γ›
      using phi_inv_sigma_auto.particular_struct_endomorphism_axioms by blast

    have COND3: ‹particular_struct_morphism src.Γ (MorphImg ω₁ src.Γ) (ω₁ ∘ φ⁻¹
∘ φ)›
      apply (intro
            particular_struct_morphism_comp[of _ ‹MorphImg φ src.Γ›]
            particular_struct_morphism_comp[of _ ‹src.Γ›])
      subgoal using particular_struct_bijection_1.phi_in_iso_morphs particular_struct_bij
particular_struct_bijection_1_def particular_struct_injection_def by blast
      subgoal using phi_inv.particular_struct_morphism_axioms by auto
      by (simp add: omega1.particular_struct_morphism_axioms)

    have P_4: ‹x = φ⁻¹ (φ z) ⟷ z = x› if AA: ‹z ∈ src.𝒫› for z
      using inv_into_f_f[OF morph_is_injective,simplified inv_morph_def[symmetric]
          , OF AA] by metis

    have x_const_phi1[simp]: ‹φ⁻¹ (φ x) = x› using P_4[OF ‹x ∈ src.𝒫›] by simp
    have x_const_sigma[simp]: ‹φ⁻¹ (σ x) = x› using np_x_E[OF phi_inv_sigma_in_auto,simpl
‹x ∈ src.𝒫›] by simp
    have omega2_src_end[simp]: ‹omega2.src.endurants = tgt.𝒫›
      using phi1_tgt by blast
    obtain DD: ‹φ₁ z ∈ tgt.endurants› ‹σ z ∈ tgt.endurants›
      using ‹z ∈ src.𝒫›
      using phi1.morph_preserves_particulars phi1_tgt sigma.morph_preserves_particulars
by auto
    have BB[simp]: ‹φ x = σ x›
      using x_const_sigma
      by (metis I_img_eq_tgt_I P(1) morph_inv_morph_img sigma.morph_preserves_particulars
src.Γ_simps(2) tgt.Γ_simps(2) tgt_Gamma_eq_Morph_img)

    show ?thesis
      apply simp
      by (metis P(1) case2 src.Γ_simps(2) that)
  qed


  have corresp_x_eq_phi_x: ‹src.correspondentIn x (MorphImg φ src.Γ) = φ x›
    apply (simp only: src.correspondentIn_def)
```

```
    apply (intro the1_equality case2)
    apply (intro ballI)
    by (intro case3 ; simp)


  show ‹?thesis›
    apply (intro iffI ; simp?)
    subgoal
      apply (simp add: corresp_x_eq_phi_x)
      using assms(2) assms(3) case3 by auto
    by (simp only: src.correspondentIn_def
        ; intro the1_equality[symmetric] case2
        ; clarify ; intro case1 ; simp)
qed

end




end
```

D.4 Anchoring and Identification Dependence

D.5 Identity Through Identity Anchors

D.5.1 Anchoring

```
theory Anchoring
  imports "../ParticularStructures/SubStructures"
begin

context ufo_particular_theory_sig
begindefinition anchors ::
    ‹'p₂ ⇒ ('p₂,'q) particular_struct ⇒
     ('p₂ ⇒ 'p) ⇒ 'p ⇒ bool›
     (‹_ −_,_→₁ _› [74,1,1,74] 75) where
  ‹y −Γₓ,φ→₁ x ≡
     x ∈ 𝒫 ∧ Γₓ ≪φ Γ ∧ y ∈ particulars Γₓ ∧
     (∀σ ∈ Morphsₓ,Γ.
      ∀z ∈ particulars Γₓ. σ z = x ⟷ z = y)›lemma anchorsI[intro!]:
  assumes
```

```
    ‹x ∈ 𝒫› ‹Γx ≪φ Γ› ‹y ∈ particulars Γx›
    ‹⋀φ z. ⟦ z ∈ particulars Γx ; φ ∈ Morphs_{Γx,Γ} ⟧ ⟹ φ z = x ⟷ z = y›
  shows ‹y −Γx,φ→1 x›
  apply (simp add: anchors_def assms(1,2,3)
             del: morphs_iff injectives_iff)
  apply (intro ballI)
  using assms by metis

lemma anchorsE[elim!]:
  assumes ‹y −Γx,φ→1 x›
  obtains
    ‹x ∈ 𝒫› ‹Γx ≪φ Γ› ‹y ∈ particulars Γx›
    ‹⋀φ z. ⟦ z ∈ particulars Γx ; φ ∈ Morphs_{Γx,Γ} ⟧ ⟹ φ z = x ⟷ z = y›
  using assms by (simp add: anchors_def)

definition anchored_particulars :: ‹'p set› (‹𝒫↓›) where
  ‹𝒫↓ ≡ { x | x (y :: ZF) Γx φ . y −Γx,φ→1 x }›

lemma anchored_particulars_I[intro]:
  fixes y :: ZF and Γx and x
  assumes ‹y −Γx,φ→1 x›
  shows ‹x ∈ 𝒫↓›
  using assms
  by (simp add: anchored_particulars_def ; metis)

lemma anchored_particulars_E[elim!]:
  assumes ‹x ∈ 𝒫↓›
  obtains y :: ZF and Γx φ where ‹y −Γx,φ→1 x›
  using assms
  by (simp add: anchored_particulars_def ; metis)


lemma anchored_particulars_I1[intro!]:
  fixes y :: ‹'p1›
  assumes ‹y −Γx,φx→1 x›
  shows ‹x ∈ 𝒫↓›
proof -
  obtain A: ‹x ∈ 𝒫› ‹Γx ≪φx Γ› ‹y ∈ particulars Γx›
     and B: ‹⋀φ z. ⟦ z ∈ particulars Γx ; φ ∈ Morphs_{Γx,Γ} ⟧ ⟹ φ z = x ⟷ z =
y›
    using assms by blast
  interpret I: particular_struct_injection ‹Γx› ‹Γ› ‹φx› using A(2) by simp
```

```
  obtain σ :: ‹'p1 ⇒ ZF› where ‹inj σ› using I.src.injection_to_ZF_exist by blast
  interpret I2: particular_struct_bijection_1 ‹Γx› σ
    using I.src.inj_morph_img_isomorphism[of σ]
    by (metis I.src.Γ_simps UNIV_I ‹inj σ› inj_on_id inj_on_subset particular_struct_eqI
subsetI)
  have C: ‹I2.tgt.Γ = MorphImg σ Γx›
    using I2.tgt.Γ_simps by blast

  interpret I3: particular_struct_bijection_1 ‹MorphImg σ Γx› ‹inv σ›
    apply (intro I2.tgt.inj_morph_img_isomorphism[simplified C])
    subgoal by (metis I2.inv_morph_morph UNIV_I image_eqI inj_on_inv_into subsetI)
    using I.src.injection_to_ZF_exist by blast

  have D[simp]: ‹inv σ ‘ σ ‘ X = X› for X
    using ‹inj σ› by (auto simp: image_def)

  have E[simp]: ‹inv σ (σ x) = x› for x
    using ‹inj σ› by (auto simp: image_def)

  have F[simp]: ‹MorphImg (inv σ) (MorphImg σ Γx) = Γx›
    apply (intro particular_struct_eqI ext ; auto simp add: particular_struct_morphism_image_s
    subgoal using D by blast
    subgoal by force
    by (metis UNIV_I ‹inj σ› inv_into_f_f)

  interpret I4: particular_struct_injection ‹MorphImg σ Γx› Γ ‹φx ∘ inv σ›
    apply (intro particular_struct_injection_comp[of _ ‹Γx›])
    using I3.particular_struct_injection_axioms[simplified]
          I.particular_struct_injection_axioms
    by simp+

  have G: ‹φx ∘ inv σ ∈ InjMorphsMorphImg σ Γx,Γ›
    using I4.particular_struct_injection_axioms by blast
  then have H: ‹MorphImg σ Γx ≪φx ∘ inv σ Γ› by blast

  have J[simp]: ‹(φ z = x) = (z = σ y)›
    if as: ‹z ∈ I3.src.endurants›
           ‹particular_struct_morphism (MorphImg σ Γx) Γ φ› for z φ
  proof -
    interpret I5: particular_struct_morphism ‹MorphImg σ Γx› Γ φ using as by simp
    have AA: ‹φ ∘ σ ∈ MorphsΓx,Γ›
      apply (intro morphs_I particular_struct_morphism_comp[of _ ‹MorphImg σ Γx›]
```

```
as)
      by (simp add: I2.particular_struct_morphism_axioms)
    have BB: ‹inv σ z ∈ I.src.endurants›
      by (metis F I3.I_img_eq_tgt_I I3.morph_image_def image_eqI as(1))
    have CC:‹σ (inv σ z) = z› using as(1)
      by (meson BB E I2.morph_preserves_particulars I3.morph_is_injective inj_onD)
    have DD: ‹(φ z = x) = (inv σ z = y)›
      using B[OF BB AA] CC
      by (simp ; metis)
    show ?thesis
      apply (simp add: DD)
      using CC by auto
  qed
  have K: ‹σ y −MorphImg σ Γx,φx ∘ inv σ→1 x›
    apply (intro anchorsI I4.particular_struct_injection_axioms H A)
    using A(3) by auto
  show ?thesis
    by (intro anchored_particulars_I[OF K])
qed

lemma anchor_to_zf_I:
  fixes y :: 'a
  assumes ‹y −Γx,φ→1 x›
  shows ‹∃(y1 :: ZF) Γ1 σ. y1 −Γ1,σ→1 x ∧ Γ1 ∈ IsoModelsΓx,TYPE(ZF)›
proof -
  obtain A: ‹x ∈ 𝒫› ‹y ∈ particulars Γx› ‹Γx ≪φ Γ›
            ‹⋀σ z. ⟦ z ∈ particulars Γx ; σ ∈ MorphsΓx,Γ ⟧ ⟹ σ z = x ⟷ z =
y›
    using anchorsE[OF assms] by metis
  interpret phi: particular_struct_injection Γx Γ φ
    using A(3) .
  obtain f :: ‹'a ⇒ ZF› where f: ‹inj f› using phi.src.injection_to_ZF_exist by
blast
  have ‹phi.src.Γ = Γx› by auto
  have ‹particular_struct_bijection_1 Γx f› using f
    apply (subst ‹phi.src.Γ = Γx›[symmetric])
    apply (intro phi.src.inj_morph_img_isomorphism)
    subgoal using inj_on_subset by blast
    using inj_on_id by blast
  then interpret gamma_x: particular_struct_bijection_1 Γx f ‹TYPE('a)› ‹TYPE(ZF)›
by blast
  have ‹particular_struct_injection (MorphImg f Γx) Γx gamma_x.inv_morph›
```

```
    using particular_struct_bijection_def by blast
  then interpret gamma_x_inv: particular_struct_injection ‹MorphImg f Γx› Γx gamma_x.inv_morph
.
  have ‹particular_struct_injection (MorphImg f Γx) Γ (φ ∘ gamma_x.inv_morph)›
    apply (intro particular_struct_injection_comp[of _ Γx])
    by (intro_locales)
  then interpret phi_gamma_x_inv: particular_struct_injection
          ‹MorphImg f Γx› Γ ‹φ ∘ gamma_x.inv_morph› ‹TYPE(ZF)› ‹TYPE('p)› .
  have R1: ‹MorphImg f Γx ≪φ ∘ gamma_x.inv_morph Γ›
    using injectives_I[OF phi_gamma_x_inv.particular_struct_injection_axioms]
    by blast
  have R2: ‹φ ∈ Morphs_{Γx,Γ}›
    using phi.particular_struct_morphism_axioms by blast
  have R3[simp]: ‹φ y = x›
    using A(4)[OF _ R2,simplified,of y,simplified] A(2) by metis
  have R4: ‹f y ∈ gamma_x_inv.src.𝒫› using A(2) by blast

  have R5: ‹f y −MorphImg f Γx,φ ∘ gamma_x.inv_morph→1 x›
  proof (intro anchorsI A(1) R1 R4)
    fix σ z
    assume as: ‹z ∈ gamma_x_inv.src.𝒫› ‹σ ∈ Morphs_{MorphImg f Γx,Γ}›
    interpret sigma: particular_struct_morphism ‹MorphImg f Γx› Γ σ ‹TYPE(ZF)› ‹TYPE('p)›
      using as(2) by blast
    interpret particular_struct_morphism Γx Γ ‹φ ∘ gamma_x.inv_morph ∘ f› ‹TYPE('a)›
‹TYPE('p)›
      apply (intro particular_struct_morphism_comp[of _ ‹MorphImg f Γx›])
      by intro_locales

    interpret sigma_f: particular_struct_morphism Γx Γ ‹σ ∘ f› ‹TYPE('a)› ‹TYPE('p)›

      apply (intro particular_struct_morphism_comp[of _ ‹MorphImg f Γx›])
      by intro_locales
    have RR1: ‹σ ∘ f ∈ Morphs_{Γx,Γ}›
      using sigma_f.particular_struct_morphism_axioms by blast
    have I1: ‹gamma_x.inv_morph (f x) = x› if ‹x ∈ phi.src.𝒫› for x
      using that  by simp
    have I2: ‹f (gamma_x.inv_morph x) = x› if ‹x ∈ gamma_x.tgt.𝒫› for x
      using that  by simp
    show ‹σ z = x ⟷ z = f y›
      supply R =  I1 I2 A(4)[OF _ RR1,simplified] R3 as(1) A(2)
      apply (intro iffI)
      subgoal using R
```

```
        by (metis gamma_x_inv.morph_preserves_particulars)
      using R by blast
  qed
  have R6: ‹MorphImg f Γx ∈ IsoModelsΓx,TYPE(ZF)›
    using gamma_x.particular_struct_bijection_1_axioms by blast
  then show ?thesis using R5 by blast
qed

lemma anchored_particulars_are_particulars: ‹P↓ ⊆ P› by blast

end

end


theory AnchoringAndIdentifiability
  imports Anchoring "../ParticularStructures/Permutability" "../ParticularStructures/Inve
begin

context ufo_particular_theory
beginlemma non_permutable_are_anchored: ‹P1! ⊆ P↓›
proof (intro subsetI ; elim non_permutables_E)
  fix x
  assume ‹x ∈ E› ‹non_permutable x›
  obtain σ :: ‹'p ⇒ ZF› where ‹inj σ› using injection_to_ZF_exist by blast
  interpret sigma: particular_struct_bijection_1 Γ σ
    by (metis ‹inj σ› inj_on_subset inj_morph_img_isomorphism UNIV_I inj_on_id subsetI)

  interpret sigma_inv: particular_struct_bijection ‹MorphImg σ Γ› Γ sigma.inv_morph

    using sigma.inv_is_bijective_morphism by blast

  have ‹sigma.inv_morph ∈ InjMorphsMorphImg σ Γ,Γ›
    by (intro injectives_I sigma_inv.particular_struct_injection_axioms)
  then have A: ‹MorphImg σ Γ ≪sigma.inv_morph Γ›
    by blast

  have B: ‹σ x ∈ sigma_inv.src.endurants›
    by (simp add: ‹x ∈ E› sigma.morph_preserves_particulars)

  have C: ‹φ z = x ⟷ z = σ x›
    if as: ‹z ∈ sigma_inv.src.endurants›
```

```
      ‹φ ∈ Morphs_MorphImg σ Γ,Γ› for φ z
  proof -
    interpret phi: particular_struct_morphism ‹MorphImg σ Γ› Γ φ
      using as(2) by blast
    have AA: ‹δ y = x ⟷ y = x› if ‹δ ∈ EndoMorphs_Γ› ‹y ∈ ℰ› for y δ
      using non_permutable_E[OF ‹non_permutable x›] that by metis
    interpret phi_sigma: particular_struct_morphism Γ Γ ‹φ ∘ σ›
      by (intro particular_struct_morphism_comp[of _ ‹MorphImg σ Γ›]
              sigma.particular_struct_morphism_axioms
              phi.particular_struct_morphism_axioms)
    have ‹particular_struct_endomorphism Γ (φ ∘ σ)› by intro_locales
    then have BB: ‹φ ∘ σ ∈ EndoMorphs_Γ› by blast
    have CC: ‹sigma.inv_morph z ∈ ℰ›
      using sigma_inv.morph_preserves_particulars that(1) by auto
    have DD: ‹sigma.inv_morph z = x ⟷ z = σ x›
      using ‹x ∈ ℰ› as(1) by fastforce
    have EE: ‹z = σ x ⟷ φ (σ (sigma.inv_morph z)) = x›
      using AA[of ‹φ ∘ σ›,of ‹sigma.inv_morph z›,OF BB,simplified,OF CC] DD by simp
    show ?thesis
      by (simp add: EE as(1))
  qed
  show ‹x ∈ 𝒫↓›
    apply (intro anchored_particulars_I[of ‹σ x› ‹MorphImg σ Γ› ‹sigma.inv_morph›]
anchorsI ‹x ∈ ℰ› A B)
    using C by metis
qed

lemma anchored_are_non_permutable: ‹𝒫↓ ⊆ 𝒫1!›
proof (intro subsetI ; elim anchored_particulars_E anchorsE
        ; intro non_permutables_I non_permutable_I ; assumption?
        ; intro iffI ; hypsubst_thin)
  fix x y and Γx :: ‹('p1,'q) particular_struct› and φ φ' y1
  assume A: ‹Γx ≪φ Γ› ‹φ' ∈ EndoMorphs_Γ›
  then interpret phi: particular_struct_injection ‹Γx› ‹Γ› φ
    by blast
  have phi_in_morphs: ‹φ ∈ Morphs_Γx,Γ›
    using phi.particular_struct_morphism_axioms by blast
  interpret phi1_auto: particular_struct_endomorphism ‹Γ› φ'
    using A(2) by blast
  interpret phi1: particular_struct_morphism Γ Γ φ'
    using phi1_auto.particular_struct_morphism_axioms by blast
  have ‹particular_struct_morphism Γx Γ (φ' ∘ φ)›
```

```
    apply (intro particular_struct_morphism_comp[of _ Γ])
    subgoal using phi.particular_struct_morphism_axioms by blast
    by (simp add: phi1_auto.particular_struct_morphism_axioms)
  then have D: ‹(φ' ∘ φ) ∈ Morphs_{Γx,Γ}› by blast

  show ‹y1 = φ' y1›
    if A:
      ‹φ' y1 ∈ 𝒫›
      ‹y ∈ particulars Γx›
      ‹⋀φ z. ⟦ z ∈ particulars Γx ;
                φ ∈ Morphs_{Γx,Γ} ⟧ ⟹ φ z = φ' y1 ⟷ z = y›
      ‹y1 ∈ 𝒫›
  proof -
    obtain B: ‹y ∈ phi.src.endurants› using A(2) by simp
    have C: ‹φ y = φ' y1›
      using A(3)[OF B phi_in_morphs] by simp
    have E: ‹φ' (φ y) = φ' y1› using A(3)[OF B D] by simp
    have F: ‹φ y ∈ 𝒫›  by (simp add: C A(1))
    obtain σ where ‹particular_struct_endomorphism Γ σ›
                   and sigma[simp]: ‹σ (φ y) = y1›
      using phi1.choice[simplified,of ‹φ y› y1,OF F A(4) E] by metis
    then interpret sigma_auto: particular_struct_endomorphism Γ σ by simp
    interpret sigma: particular_struct_morphism Γ Γ σ
      using sigma_auto.particular_struct_morphism_axioms by simp
    interpret sigma_phi: particular_struct_morphism Γx Γ ‹σ ∘ φ›
      apply (intro particular_struct_morphism_comp[of _ Γ])
      subgoal using phi.particular_struct_morphism_axioms by blast
      by (simp add: sigma.particular_struct_morphism_axioms)
    have G: ‹(σ ∘ φ) ∈ Morphs_{Γx,Γ}›
      using sigma_phi.particular_struct_morphism_axioms
      by blast
    show ?thesis
      using A(3)[OF B G,simplified] by simp
  qed
  show ‹φ' x = x›
    if A:
      ‹x ∈ 𝒫›
      ‹y ∈ particulars Γx›
      ‹⋀φ z. ⟦ z ∈ particulars Γx ;
                φ ∈ Morphs_{Γx,Γ} ⟧ ⟹ φ z = x ⟷ z = y›
  proof -
    obtain B: ‹y ∈ phi.src.endurants› using A(2) by simp
```

```
    have C: ‹φ y = x› using A(3)[OF B phi_in_morphs] by simp
    then show ‹φ' x = x› using A(3)[OF B D] by simp
  qed
qed

theorem  anchored_are_the_non_permutable: ‹𝒫↓ = 𝒫1!›
  using anchored_are_non_permutable non_permutable_are_anchored by blast

end

end
```

D.6 Anchoring Classes and Identification Dependence

D.6.1 Anchoring Classes

```
theory AnchoringClasses
  imports "../Identity/Anchoring"
begin

context ufo_particular_theory
begindefinition minimallyAnchors ::
  ‹'p2 ⇒  ('p2,'q) particular_struct ⇒
  ('p2 ⇒ 'p) ⇒ 'p ⇒ bool›
  (‹_ −_,_→⊥ _› [74,1,1,74] 75)
  where
  ‹y −Γx,φ→⊥ x ≡
    y −Γx,φ→1 x ∧
    (∀Γ' z σ1 σ2.
        Γ' ≪σ1 Γx ∧ z −Γ',σ2→1 x ⟶
        Γ' ∈ IsoModelsΓx,TYPE(ZF))›

lemma minimallyAnchorsI[intro!]:
  assumes ‹y −Γx,φ→1 x› ‹⋀Γ' z σ1 σ2. ⟦ Γ' ≪σ1 Γx ; z −Γ',σ2→1 x ⟧ ⟹ Γ' ∈
IsoModelsΓx,TYPE(ZF)›
  shows ‹y −Γx,φ→⊥ x›
  using assms by (simp only: minimallyAnchors_def ; metis)

lemma minimallyAnchorsE[elim!]:
  assumes ‹y −Γx,φ→⊥ x›
  obtains ‹y −Γx,φ→1 x› ‹⋀Γ' z σ1 σ2. ⟦ Γ' ≪σ1 Γx ; z −Γ',σ2→1 x ⟧ ⟹ Γ' ∈
IsoModelsΓx,TYPE(ZF)›
```

using *assms* **by** *(simp only: minimallyAnchors_def ; metis)*

lemma *minimallyAnchorsD:*
assumes ‹$y\ -\Gamma_x,\varphi\rightarrow_\perp\ x$›
shows ‹$y\ -\Gamma_x,\varphi\rightarrow_1\ x$› ‹$\bigwedge\Gamma'\ z\ \sigma_1\ \sigma_2.\ [\![\ \Gamma'\ \ll_{\sigma_1}\ \Gamma_x\ ;\ z\ -\Gamma',\sigma_2\rightarrow_1\ x\ ]\!] \Longrightarrow \Gamma' \in IsoModels_\Gamma$
using *assms* **by** *(simp only: minimallyAnchors_def ; metis)*+**definition** *minimallyAnchored* *::* ‹*'p set*› (‹$\mathcal{P}_{\Downarrow}$›)
where
‹$\mathcal{P}_{\Downarrow} \equiv \{\ x\ |\ x\ (y\ ::\ ZF)\ \Gamma_x\ \varphi\ .\ y\ -\Gamma_x,\varphi\rightarrow_\perp\ x\ \}$›

lemma *minimallyAnchored_I[intro]:* ‹$(y\ ::\ ZF)\ -\Gamma_x,\varphi\rightarrow_\perp\ x \Longrightarrow x \in \mathcal{P}_{\Downarrow}$›
by *(simp only: minimallyAnchored_def mem_Collect_eq ; metis)*

lemma *minimallyAnchored_E[elim!]:*
assumes ‹$x \in \mathcal{P}_{\Downarrow}$›
obtains $y\ ::\ ZF$ **and** $\Gamma_x\ \varphi$ **where** ‹$y\ -\Gamma_x,\varphi\rightarrow_\perp\ x$›
using *assms* **by** *(simp only: minimallyAnchored_def mem_Collect_eq ; metis)*

lemma *minimallyAnchored_are_anchored:* ‹$\mathcal{P}_{\Downarrow} \subseteq \mathcal{P}_{\downarrow}$›
using *minimallyAnchored_E*
by *(metis minimallyAnchorsD(1) subsetI ufo_particular_theory_sig.anchored_particulars_I*
finitelyAnchored :: ‹*'p set*› (‹$\mathcal{P}^{fin}{}_{\downarrow}$›) **where**
‹$\mathcal{P}^{fin}{}_{\downarrow} \equiv \{\ x\ |\ x\ (y\ ::\ ZF)\ \Gamma_x\ \varphi.\ y\ -\Gamma_x,\varphi\rightarrow_1\ x\ \wedge$
$finite\ (particulars\ \Gamma_x)\ \}$›

lemma *finitelyAnchored_I[intro]:*
assumes ‹$(y\ ::\ ZF)\ -\Gamma_x,\varphi\rightarrow_1\ x$› ‹$finite\ (particulars\ \Gamma_x)$›
shows ‹$x \in \mathcal{P}^{fin}{}_{\downarrow}$›
using *assms* **by** *(simp only: finitelyAnchored_def mem_Collect_eq ; metis)*

lemma *finitelyAnchored_E[elim!]:*
assumes ‹$x \in \mathcal{P}^{fin}{}_{\downarrow}$›
obtains $y\ \Gamma_x\ \varphi$ **where** ‹$(y\ ::\ ZF)\ -\Gamma_x,\varphi\rightarrow_1\ x$› ‹$finite\ (particulars\ \Gamma_x)$›
using *assms* **by** *(simp only: finitelyAnchored_def mem_Collect_eq ; metis)***definition**
intrinsicallyAnchored :: ‹*'p set*› (‹$\mathcal{P}^{int}{}_{\downarrow}$›)
where
‹$\mathcal{P}^{int}{}_{\downarrow} \equiv \{\ x\ |\ x\ (y\ ::\ ZF)\ \Gamma_x\ \varphi.\ y\ -\Gamma_x,\varphi\rightarrow_\perp\ x\ \wedge$
$(\forall z \in particulars\ \Gamma_x.\ (ps_inheres_in\ \Gamma_x)^{**}\ z\ y)\ \}$›

lemma *intrinsicallyAnchored_I[intro]:*
assumes ‹$(y\ ::\ ZF)\ -\Gamma_x,\varphi\rightarrow_\perp\ x$›
‹$\bigwedge z.\ z \in particulars\ \Gamma_x \Longrightarrow (ps_inheres_in\ \Gamma_x)^{**}\ z\ y$›

```
  shows ‹x ∈ 𝒫^int_↓›
  using assms by (simp only: intrinsicallyAnchored_def mem_Collect_eq ; metis)

lemma intrinsicallyAnchored_E[elim!]:
  assumes ‹x ∈ 𝒫^int_↓›
  obtains y Γ_x φ where ‹(y :: ZF) −Γ_x,φ→_⊥ x›
        ‹⋀z. z ∈ particulars Γ_x ⟹ (ps_inheres_in Γ_x)** z y›
  using assms by (simp only: intrinsicallyAnchored_def mem_Collect_eq ; metis)

lemma intrinsicallyAnchored_are_Anchored: ‹𝒫^int_↓ ⊆ 𝒫_⇓›
  using intrinsicallyAnchored_E
  by (meson minimallyAnchored_I subsetI)

end


context ufo_particular_theory
begin

lemma finite_anchors_imp_min_anchor_ex:
  fixes x and y :: ZF and Γ_x φ_x
  assumes ‹y −Γ_x,φ_x→_1 x› ‹finite (particulars Γ_x)›
  shows ‹∃(y_2 :: ZF) Γ_2 σ. y_2 −Γ_2,φ_x ∘ σ→_⊥ x ∧ Γ_2 ≪_σ Γ_x›
proof -
  obtain gamma_x: ‹x ∈ 𝒫› ‹Γ_x ≪_φ_x Γ› ‹y ∈ particulars Γ_x›
        ‹⋀φ z. ⟦ z ∈ particulars Γ_x ; φ ∈ Morphs_Γ_x,Γ ⟧ ⟹ φ z = x ⟷ z = y›
    using anchorsE[OF assms(1)] by metis
  then interpret phi_x: particular_struct_injection Γ_x Γ φ_x ‹TYPE(ZF)› ‹TYPE('p)›
‹TYPE('q)› by simp
  have ‹∃(z :: ZF) Γ' φ. z −Γ',φ_x ∘ φ→_⊥ x ∧ Γ' ≪_φ Γ_x›
  proof (rule ccontr; simp)
    assume AA: ‹∀(z :: ZF) Γ' φ. z −Γ',φ_x ∘ φ→_⊥ x ⟶ ¬ Γ' ≪_φ Γ_x ›
    have no_min_anchor: ‹False› if ‹z −Γ',φ_x ∘ φ→_⊥ x › ‹Γ' ≪_φ Γ_x› for z :: ZF and
Γ' φ
      using AA[rule_format] that by simp
    define N where ‹N ≡ { card (particulars Γ') | Γ' (z :: ZF) φ_1 . z −Γ',φ_x ∘
φ_1→_1 x ∧ Γ'≪_φ_1 Γ_x ∧ finite (particulars Γ')}›
    have ‹N ≠ ∅›
      apply (auto simp: N_def)
      apply (rule exI[of _ Γ_x] ; rule exI[of _ y]
              ; rule exI[of _ id] ; intro conjI
              ; (simp only: o_id)?)
```

```
      subgoal by (simp add: phi_x.src.particular_struct_axioms sub_structure_by_refl)
      subgoal using assms(1) by blast
      by (simp add: assms(2))

    then obtain n where N: ‹n ∈ N› ‹⋀i. i ∈ N ⟹ n ≤ i›
      by (meson Inf_nat_def1 bdd_below_def bot.extremum cInf_less_iff leI less_imp_neq)
    then obtain Γ' and z :: ZF and φ'
      where gamma_z: ‹z −Γ',φx ∘ φ'→1 x› ‹Γ'≪φ' Γx› ‹finite (particulars Γ')›
            ‹n = card (particulars Γ')›
      using N apply (simp add: N_def)
      by blast
    interpret phi_x': particular_struct_injection Γ' Γ ‹φx ∘ φ'› using gamma_z(1)
by blast
    interpret phi': particular_struct_injection Γ' Γx φ' using gamma_z(2) by blast

    have gamma'': ‹¬ z' −Γ'',φ1→1 x› if ‹Γ'' ≪φ2 Γ'› ‹∀φ. ¬ Γ' ≪φ Γ''› for Γ''
and z' :: ZF and φ1 φ2
    proof (rule ccontr ; simp)
      assume as3: ‹z' −Γ'',φ1→1 x›
      have C1: ‹Γ'' ≪φ' ∘ φ2 Γx› using that(1) gamma_z(2) sub_structure_by_trans
        by metis
      then have C2: ‹finite (particulars Γ'')›
        using assms(2) finite_card_sub_structure_by_finite by metis
      have C3: ‹card (particulars Γ'') < card (particulars Γ')›
        using that(1,2) by (simp add: finite_card_substruct_lt gamma_z(3))
      have C3_1: ‹∃z φ1. z −Γ'',φx ∘ φ1→1 x ∧ Γ'' ≪φ1 Γx ∧ finite (particulars
Γ'')›
      proof -
        have "∀f p. Γ'' ≪(f::ZF ⇒ 'p) ∘ (φ' ∘ φ2) p ∨ ¬ Γx ≪f p"
          using C1 sub_structure_by_trans by blast
        then have "z' −Γ'',φx ∘ (φ' ∘ φ2)→1 x"
          using as3 phi_x.particular_struct_injection_axioms by blast
        then show ?thesis
          by (meson C1 C2)
      qed

      have C4: ‹card (particulars Γ'') ∈ N›
        apply (simp add: N_def)
        apply (intro exI[of _ Γ''] conjI C1 C2 ; simp?)
        using C3_1 by metis

      then have ‹card (particulars Γ') ≤ card (particulars Γ'')›
```

```
        using N(2) gamma_z(4) by simp
      then show False using C3 by auto
    qed
    have gamma_1: ‹Γ'' ∈ IsoModels_{Γ',TYPE(ZF)}› if as3: ‹Γ'' ≪_{φ1} Γ'› ‹z' −Γ'',φ2→1
x› for Γ'' and z' :: ZF and φ1 φ2
    proof -
      have A1: ‹finite (particulars Γ'')›
        using finite_card_sub_structure_by_finite gamma_z that by metis
      obtain φ3 where A2: ‹Γ' ≪_{φ3} Γ''›
        using gamma''[OF as3(1),of z' φ2] as3(2) by metis
      have A3: ‹Γ'' ∼_{φ1} Γ'›
        using sub_structure_by_finite_weak_antisym that(1) A2 A1 by metis
      then show ?thesis
      proof -
        have "MorphImg φ1 Γ'' = Γ'"
          by (metis A3 particular_struct_bijection_iff_particular_struct_bijection_1)
        then have "Γ' ∈ {MorphImg f Γ'' |f. f ∈ Collect (particular_struct_bijection_1
Γ'')}"
          using A3 particular_struct_bijection_iff_particular_struct_bijection_1
by auto
        then show ?thesis
          by (metis (no_types) isomorphic_models_def isomorphic_models_sym bijections1_def)
      qed
    qed

    have ZZ: ‹z −Γ',φx ∘ φ'→⊥ x›
      apply (intro minimallyAnchorsI gamma_z(1))
      subgoal for Γ'' z' σ1 σ2
        using gamma_1 by metis
      done

    show False using AA[rule_format,of z Γ'] ZZ
        gamma_z(2) by simp
  qed
  then obtain  z :: ZF and  Γ' φ
    where ‹z −Γ',φx ∘ φ→⊥ x› ‹Γ' ≪_φ Γx›
    using that by metis
  then show ‹?thesis› by metis
qedlemma finitely_anchored_are_minimally_anchored: ‹P^fin_↓ ⊆ P_⇓›
proof (intro subsetI ; elim finitelyAnchored_E)
  fix x and y :: ZF and Γx φx
  assume assms: ‹y −Γx,φx→1 x› ‹finite (particulars Γx)›
```

```
  then obtain  z :: ZF and  Γ' φ where ‹z −Γ',φ→⊥ x›
    using assms finite_anchors_imp_min_anchor_ex   by metis
  then show ‹x ∈ 𝒫⇓›
    using  minimallyAnchored_I[of z Γ'] by simp
qed

lemma min_anchor_to_zf_I:
  fixes y :: 'a
  assumes ‹y −Γx,σ→⊥ x›
  shows ‹∃(y1 :: ZF) Γ1 φz. y1 −Γ1,φz→⊥ x ∧ Γ1 ∈ IsoModelsΓx,TYPE(ZF)›
proof -
  obtain A: ‹y −Γx,σ→1 x› ‹⋀Γ' z σ1 σ2. ⟦ Γ' ≪σ1 Γx ; z −Γ',σ2→1 x ⟧ ⟹ Γ' ∈
IsoModelsΓx,TYPE(ZF)›
    using minimallyAnchorsE[OF assms] by metis
  obtain yz :: ZF and Γz φz where B: ‹yz −Γz,φz→1 x› ‹Γz ∈ IsoModelsΓx,TYPE(ZF)›

    using anchor_to_zf_I[OF A(1)] by blast
  obtain φ where C: ‹particular_struct_bijection_1 Γx φ› ‹Γz = MorphImg φ Γx›
    using B(2) by blast
  then interpret phi: particular_struct_bijection_1 Γx φ by simp
  have ‹Γx ∼φ Γz›
    by (simp add: C(2) phi.particular_struct_bijection_axioms)
  have D: ‹Γz ≪phi.inv_morph Γx›
    using C(2) phi.inv_is_bijective_morphism by blast
  obtain E: ‹x ∈ ℰ›
   ‹Γx ≪σ Γ›
   ‹y ∈ phi.src.𝒫›
   ‹⋀φ z. ⟦ z ∈ phi.src.endurants; φ ∈ MorphsΓx,Γ ⟧ ⟹ φ z = x ⟷ z = y›
    using A(1)[THEN anchorsE] by metis
  obtain F: ‹Γz ≪φz Γ›
   ‹yz ∈ particulars Γz›
   ‹⋀φ z. ⟦ z ∈ particulars Γz ; φ ∈ MorphsΓz,Γ ⟧ ⟹ φ z = x ⟷ z = yz›
    using B(1)[THEN anchorsE] by metis
  have G: ‹Γz ≪σ ∘ phi.inv_morph Γ›
    by (intro particular_struct_injection_comp[of _ Γx] D E(2))
  have ‹yz −Γz,σ ∘ phi.inv_morph→⊥ x›
    apply (intro minimallyAnchorsI anchorsI B E(1) F(2) G F(3) ; assumption?)
    subgoal premises P for Γ' z σ1 σ2
      supply R = A(2)[OF particular_struct_injection_comp[OF P(1) D] P(2)]
      by (metis (no_types, hide_lams) R C isomorphic_models_iff isomorphic_models_sym

          bijections_iff morph_img_comp particular_struct_bijection_1_comp)
```

```
    done
  then show ?thesis using B(2) by metis
qed


end



end
```

D.6.2 Anchoring Sets

```
theory AnchoringSets
  imports AnchoringClasses
begin

context ufo_particular_theory
begindefinition anchorSets :: ‹'p ⇒ 'p set set› (‹𝒜_› [1] 1000)
  where
  ‹𝒜x ≡ { φ ` particulars Γx | φ Γx (y :: ZF) . y −Γx,φ→⊥ x }›definition finiteAnchorSets
::
  ‹'p ⇒ 'p set set›
  (‹𝒜fin_› [1] 1000)
  where
    ‹𝒜finx ≡ {X . X ∈ 𝒜x ∧ finite X}›

definition intrinsicAnchorSets ::
  ‹'p ⇒ 'p set set›
  (‹𝒜int_› [1] 1000)
  where
    ‹𝒜intx ≡ {X . X ∈ 𝒜x ∧ (∀y ∈ X. (◁)** y x)}›definition identityCore ::
  ‹'p ⇒ 'p set›
  (‹IdCore_› [1] 999)
  where
    ‹IdCorex ≡ ⋂𝒜x›definition intrinsicIdentityCore ::
  ‹'p ⇒ 'p set›
  (‹IntIdCore_› [1] 999)
  where
    ‹IntIdCorex ≡ ⋂𝒜intx›

definition finiteIdentityCore ::
```

‹'p ⇒ 'p set›
(‹FinIdCore_› [1] 999)
where
‹$FinIdCore_X \equiv \bigcap \mathcal{A}^{fin}{}_X$›

definition identityComplements :: ‹'p ⇒ 'p set set› (‹IdComps_› [1] 999) **where**
‹$IdComps_X \equiv \{ X - IdCore_X \mid X . X \in \mathcal{A}_X\}$›

definition finiteIdentityComplements :: ‹'p ⇒ 'p set set› (‹FinIdComps_› [1] 999)
where
‹$FinIdComps_X \equiv \{ X - FinIdCore_X \mid X . X \in \mathcal{A}^{fin}{}_X\}$›

definition intrinsicIdentityComplements :: ‹'p ⇒ 'p set set› (‹IntIdComps_› [1] 999)
where
‹$IntIdComps_X \equiv \{ X - IntIdCore_X \mid X . X \in \mathcal{A}^{int}{}_X\}$›

lemma anchorSets_I[intro]: ‹$(y :: ZF) -\Gamma_x,\varphi\rightarrow_\perp x \Longrightarrow \varphi$ ‘ particulars $\Gamma_x \in \mathcal{A}_X$›
by (simp only: anchorSets_def mem_Collect_eq ; metis)

lemma anchorSets_E[elim!]:
assumes ‹$X \in \mathcal{A}_X$›
obtains $\varphi\ \Gamma_x\ y$ **where** ‹$(y :: ZF) -\Gamma_x,\varphi\rightarrow_\perp x$› ‹$X = \varphi$ ‘ particulars Γ_x›
using assms
by (simp only: anchorSets_def mem_Collect_eq ; metis)

lemma anchorSets_iff:
‹$X \in \mathcal{A}_X \longleftrightarrow (\exists \varphi\ \Gamma_x\ y.\ (y :: ZF) -\Gamma_x,\varphi\rightarrow_\perp x \wedge X = \varphi$ ‘ particulars $\Gamma_x)$›
by (simp only: anchorSets_def mem_Collect_eq ; metis)

lemma anchorSets_subset_particulars: ‹$X \in \mathcal{A}_X \Longrightarrow X \subseteq \mathcal{P}$›
proof (auto)
fix φ :: "ZF ⇒ 'p" **and** Γ_x :: "(ZF, 'q) particular_struct" **and** y :: ZF **and** xa :: ZF
assume a1: "$\Gamma_x \ll_\varphi \Gamma$"
assume a2: "xa ∈ particulars Γ_x"
have f3: "∀z za. possible_worlds_sig.ed (ps_worlds Γ) (φ z) (φ za) ∨ ¬ possible_world
(ps_worlds Γ_x) z za"
using a1 **by** (meson particular_struct_injection.axioms(1) particular_struct_morphism.m
possible_worlds_sig.edE)
have "∀z za. ¬ possible_worlds_sig.indep (ps_worlds Γ_x) z za ∨ possible_worlds_sig.in
(ps_worlds Γ) (φ z) (φ za)"
using a1 **by** (meson particular_struct_injection.axioms(1) particular_struct_morphism.m

```
possible_worlds_sig.indep_def)
  then show "φ xa ∈ ℰ"
    using f3 a2 by (metis (full_types) Γ_simps(2) edE possible_worlds_sig.indep_def)
qed

lemma finiteAnchorSets_I[intro!]:
  assumes ‹X ∈ 𝒜x› ‹finite X›
  shows ‹X ∈ 𝒜^fin_x›
  using assms
  by (auto simp: finiteAnchorSets_def)

lemma finiteAnchorSets_E[elim!]:
  assumes ‹X ∈ 𝒜^fin_x›
  obtains ‹X ∈ 𝒜x› ‹finite X›
  using assms
  by (auto simp: finiteAnchorSets_def)

lemma finiteAnchorSets_D:
  assumes ‹X ∈ 𝒜^fin_x›
  shows ‹X ∈ 𝒜x› ‹finite X›
  using assms
  by (auto simp: finiteAnchorSets_def)

lemma finiteAnchorSets_are_AnchorSets: ‹𝒜^fin_x ⊆ 𝒜x›
  by auto

lemma finiteAnchorSets_are_Particulars: ‹X ∈ 𝒜^fin_x ⟹ X ⊆ 𝒫›
  using anchorSets_subset_particulars
  by (meson finiteAnchorSets_D(1))

lemma intrinsicAnchorSets_I[intro!]:
  assumes ‹X ∈ 𝒜x› ‹⋀y. y ∈ X ⟹ (◁)** y x›
  shows ‹X ∈ 𝒜^int_x›
  using assms by (auto simp: intrinsicAnchorSets_def)

lemma intrinsicAnchorSets_E[elim!]:
  assumes ‹X ∈ 𝒜^int_x›
  obtains ‹X ∈ 𝒜x› ‹⋀y. y ∈ X ⟹ (◁)** y x›
  using assms by (auto simp: intrinsicAnchorSets_def)

lemma intrinsicAnchorSets_D[elim!]:
  assumes ‹X ∈ 𝒜^int_x›
```

shows ‹$X \in \mathcal{A}_X$› ‹$\bigwedge y.\ y \in X \Longrightarrow (\triangleleft)^{**}\ y\ x$›
using *assms* **by** *(auto simp: intrinsicAnchorSets_def)*

lemma *intrinsicAnchorSets_are_AnchorSets:* ‹$\mathcal{A}^{int}{}_X \subseteq \mathcal{A}_X$›
by *auto*

lemma *intrinsicAnchorSets_are_Particulars:* ‹$X \in \mathcal{A}^{int}{}_X \Longrightarrow X \subseteq \mathcal{P}$›
by *(meson anchorSets_subset_particulars intrinsicAnchorSets_D(1))*

lemma *identityCoreI[intro]:* ‹$(\bigwedge X.\ X \in \mathcal{A}_X \Longrightarrow y \in X) \Longrightarrow y \in IdCore_X$›
by *(auto simp: identityCore_def)*

lemma *identityCoreD[dest]:*
assumes ‹$y \in IdCore_X$› ‹$X \in \mathcal{A}_X$›
shows ‹$y \in X$›
using *assms* **by** *(auto simp: identityCore_def)*

lemma *intrinsicIdentityCoreI[intro]:* ‹$(\bigwedge X.\ X \in \mathcal{A}^{int}{}_X \Longrightarrow y \in X) \Longrightarrow y \in IntIdCore_X$›
by *(auto simp: intrinsicIdentityCore_def)*

lemma *intrinsicIdentityCoreD[dest]:*
assumes ‹$y \in IntIdCore_X$› ‹$X \in \mathcal{A}^{int}{}_X$›
shows ‹$y \in X$›
using *assms* **by** *(auto simp: intrinsicIdentityCore_def)*

lemma *intrinsic_id_core:* ‹$IdCore_X \subseteq IntIdCore_X$›
using *identityCore_def intrinsicAnchorSets_D(1) intrinsicIdentityCoreI* **by** *auto*

lemma *finiteIdentityCoreI[intro]:* ‹$(\bigwedge X.\ X \in \mathcal{A}^{fin}{}_X \Longrightarrow y \in X) \Longrightarrow y \in FinIdCore_X$›
by *(auto simp: finiteIdentityCore_def)*

lemma *finiteIdentityCoreD[dest]:*
assumes ‹$y \in FinIdCore_X$› ‹$X \in \mathcal{A}^{fin}{}_X$›
shows ‹$y \in X$›
using *assms* **by** *(auto simp: finiteIdentityCore_def)*

lemma *finite_id_core:* ‹$IdCore_X \subseteq FinIdCore_X$›
using *identityCore_def finiteAnchorSets_D(1) finiteIdentityCoreI* **by** *auto*

lemma *identityComplementsI[intro!]:*
assumes ‹$X \in \mathcal{A}_X$› ‹$Y = X - IdCore_X$›
shows ‹$Y \in IdComps_X$›

```
  using assms
  apply (simp add: identityComplements_def)
  by metis

lemma identityComplementsI1:
  assumes ‹X ∈ 𝒜_X›
  shows ‹X - IdCore_X ∈ IdComps_X›
  using assms
  apply (simp add: identityComplements_def)
  by metis

lemma identityComplementsE[elim!]:
  assumes ‹Y ∈ IdComps_X›
  obtains X where ‹X ∈ 𝒜_X› ‹Y = X - IdCore_X›
  using assms
  apply (simp add: identityComplements_def)
  by metis

lemma identityComplementsD:
  assumes ‹Y ∈ IdComps_X›
  shows ‹IdCore_X ∩ Y = ∅› ‹IdCore_X ∪ Y ∈ 𝒜_X›
  subgoal using assms by (elim identityComplementsE ; simp)
  subgoal
    using assms
    apply (elim identityComplementsE ; hypsubst_thin ; simp)
    by (metis Diff_partition Un_Diff_cancel identityCoreD subset_eq)
  done

lemma identityComplementsI2:
  assumes ‹IdCore_X ∩ Y = ∅› ‹IdCore_X ∪ Y ∈ 𝒜_X›
  shows ‹Y ∈ IdComps_X›
  apply (intro identityComplementsI[OF assms(2)])
  using assms Diff_subset_conv UnI2
  by auto

lemma finiteIdentityComplementsI[intro!]:
  assumes ‹X ∈ 𝒜^fin_X› ‹Y = X - FinIdCore_X›
  shows ‹Y ∈ FinIdComps_X›
  using assms
  apply (simp add: finiteIdentityComplements_def)
  by metis
```

```
lemma finiteIdentityComplementsI1:
  assumes ‹X ∈ 𝒜^fin_X›
  shows ‹X - FinIdCore_X ∈ FinIdComps_X›
  using assms
  apply (simp add: finiteIdentityComplements_def)
  by metis

lemma finiteIdentityComplementsE[elim!]:
  assumes ‹Y ∈ FinIdComps_X›
  obtains X where ‹X ∈ 𝒜^fin_X› ‹Y = X - FinIdCore_X›
  using assms
  apply (simp add: finiteIdentityComplements_def)
  by metis

lemma finiteIdentityComplementsD:
  assumes ‹Y ∈ FinIdComps_X›
  shows ‹FinIdCore_X ∩ Y = ∅› ‹FinIdCore_X ∪ Y ∈ 𝒜^fin_X›
  subgoal using assms by (elim finiteIdentityComplementsE ; simp)
  subgoal
    using assms
    apply (elim finiteIdentityComplementsE ; hypsubst_thin ; simp)
    by (metis Diff_partition Un_Diff_cancel finiteIdentityCoreD subset_eq)
  done

lemma finiteIdentityComplementsI2:
  assumes ‹FinIdCore_X ∩ Y = ∅› ‹FinIdCore_X ∪ Y ∈ 𝒜^fin_X›
  shows ‹Y ∈ FinIdComps_X›
  apply (intro finiteIdentityComplementsI[OF assms(2)])
  using assms Diff_subset_conv UnI2
  by auto

lemma intrinsicIdentityComplementsI[intro!]:
  assumes ‹X ∈ 𝒜^int_X› ‹Y = X - IntIdCore_X›
  shows ‹Y ∈ IntIdComps_X›
  using assms
  apply (simp add: intrinsicIdentityComplements_def)
  by metis

lemma intrinsicIdentityComplementsI1:
  assumes ‹X ∈ 𝒜^int_X›
  shows ‹X - IntIdCore_X ∈ IntIdComps_X›
  using assms
```

```
  apply (simp add: intrinsicIdentityComplements_def)
  by metis

lemma intrinsicIdentityComplementsE[elim!]:
  assumes ‹Y ∈ IntIdComps_X›
  obtains X where ‹X ∈ 𝒜^int_X› ‹Y = X - IntIdCore_X›
  using assms
  apply (simp add: intrinsicIdentityComplements_def)
  by metis

lemma intrinsicIdentityComplementsD:
  assumes ‹Y ∈ IntIdComps_X›
  shows ‹IntIdCore_X ∩ Y = ∅› ‹IntIdCore_X ∪ Y ∈ 𝒜^int_X›
  subgoal using assms by (elim intrinsicIdentityComplementsE ; simp)
  subgoal
    using assms
    apply (elim intrinsicIdentityComplementsE ; hypsubst_thin ; simp)
    by (metis Diff_partition Un_Diff_cancel intrinsicIdentityCoreD subset_eq)
  done

lemma intrinsicIdentityComplementsI2:
  assumes ‹IntIdCore_X ∩ Y = ∅› ‹IntIdCore_X ∪ Y ∈ 𝒜^int_X›
  shows ‹Y ∈ IntIdComps_X›
  apply (intro intrinsicIdentityComplementsI[OF assms(2)])
  using assms Diff_subset_conv UnI2
  by auto

lemma minimallyAnchored_have_an_AnchorSet: ‹x ∈ 𝒫_⇓ ⟷ 𝒜_x ≠ ∅›
proof
  assume A: ‹x ∈ 𝒫_⇓›
  then obtain y Γ_x φ where
    B: ‹(y :: ZF) −Γ_x,φ→_⊥ x›
    using minimallyAnchored_E
    by metis
  note B[THEN minimallyAnchorsE]
  then have C: ‹φ ` particulars Γ_x ∈ 𝒜_x›
    using anchorSets_I B by auto
  then show ‹𝒜_x ≠ ∅›
    by (metis emptyE)
next
  assume ‹𝒜_x ≠ ∅›
  then obtain X where A: ‹X ∈ 𝒜_x›
```

```
    using anchorSets_E
    by (meson all_not_in_conv)
  then obtain φ Γx y  where B:
      ‹(y :: ZF) −Γx,φ→⊥ x› ‹X = φ ‘ particulars Γx›
    using anchorSets_E by metis
  show ‹x ∈ 𝒫⇓›
    by (intro minimallyAnchored_I[OF B(1)])
qed

lemma finitelyAnchored_have_a_finite_AnchorSet:  ‹x ∈ 𝒫fin↓ ⟷ 𝒜finx ≠ ∅›
proof
  assume A: ‹x ∈ 𝒫fin↓›
  then obtain y Γx φ where
    B: ‹(y :: ZF) −Γx,φ→1 x› ‹finite (particulars Γx)›
    by blast
  obtain y1 Γx1 φ1 σ where
    C: ‹(y1 :: ZF) −Γx1,φ1→⊥ x› ‹Γx1 ≪σ Γx›
    using finite_anchors_imp_min_anchor_ex B by metis
  then interpret sigma: particular_struct_injection Γx1 Γx σ by simp
  have ‹finite (particulars Γx1)›
    apply (intro sigma.morph_is_injective[THEN inj_on_finite,OF _ B(2)])
    by blast
  then have ‹φ1 ‘ particulars Γx1 ∈ 𝒜finx›
    using C(1) anchorSets_I finiteAnchorSets_I by auto
  then show ‹𝒜finx ≠ ∅›
    by (metis emptyE)
next
  assume ‹𝒜finx ≠ ∅›
  then obtain X where A: ‹X ∈ 𝒜x› ‹finite X›
    using finiteAnchorSets_E
    by (meson all_not_in_conv)
  then obtain φ Γx y  where B:
      ‹(y :: ZF) −Γx,φ→⊥ x› ‹X = φ ‘ particulars Γx›
    using anchorSets_E by metis
  then interpret phi: particular_struct_injection Γx Γ φ by blast
  have C: ‹finite (particulars Γx)›
    using phi.morph_is_injective B(2) finite_image_iff A(2) by metis
  obtain D: ‹(y :: ZF) −Γx,φ→1 x› using B(1) by blast
  show ‹x ∈ 𝒫fin↓›
    by (intro finitelyAnchored_I[OF D C])
qed
```

lemma *minimallyAnchored_to_AnchorSet:*
  assumes ‹$x \in \mathcal{P}_{\Downarrow}$›
  obtains X **where** ‹$X \in \mathcal{A}_x$›
  using *assms* **apply** *(simp only: minimallyAnchored_have_an_AnchorSet)*
  by *(meson all_not_in_conv)*

lemma *AnchorSet_to_minimallyAnchored:*
  assumes ‹$X \in \mathcal{A}_x$›
  shows ‹$x \in \mathcal{P}_{\Downarrow}$›
  using *assms* **apply** *(simp only: minimallyAnchored_have_an_AnchorSet)*
  by *(metis emptyE)*

end

end

D.6.3 Identity-Related Dependency Relations

theory *IdentificationDependence*
  imports *AnchoringSets*
begin

context *ufo_particular_theory*
begindefinition *identification_dep* :: ‹$'p \Rightarrow 'p \Rightarrow bool$› (‹*IdDep*›)
  where ‹$IdDep\ x\ y \longleftrightarrow (\forall X \in \mathcal{A}_x.\ \exists Y \in \mathcal{A}_y.\ Y \subseteq X)$›

lemma *identification_dep_I[intro!]:*
  assumes ‹$\bigwedge X.\ X \in \mathcal{A}_x \Longrightarrow \exists Y \in \mathcal{A}_y.\ Y \subseteq X$›
  shows ‹$IdDep\ x\ y$›
  using *assms*
  by *(simp only: identification_dep_def ; metis)*

lemma *identification_dep_E[elim]:*
  assumes ‹$IdDep\ x\ y$› ‹$X \in \mathcal{A}_x$›
  obtains Y **where** ‹$Y \in \mathcal{A}_y$› ‹$Y \subseteq X$›
  using *assms*
  by *(simp only: identification_dep_def ; metis)***lemma** *identification_dep_refl[intro!]:*

```
⟨IdDep x x⟩
  apply (intro identification_dep_I)
  subgoal for X
    by (intro bexI[of _ X] ; simp)
  done
```

```
lemma identification_dep_trans[trans]:
  assumes ⟨IdDep x y⟩ ⟨IdDep y z⟩
  shows ⟨IdDep x z⟩
  using assms
  using assms apply (simp add: identification_dep_def)
  apply (intro ballI)
  subgoal premises P for X
    using P(1)[rule_format,OF P(3)] apply (elim bexE)
    subgoal premises Q for Y
      using P(2)[rule_format,OF Q(1)] apply (elim bexE)
      subgoal for Z
        apply (intro bexI[of _ Z])
        using P Q by simp+
      done
    done
  done

end

end
```

APPENDIX E – Sortality

E.1 Finite Particular Structures

```
theory FiniteParticularStructure
  imports "../ParticularStructures/InverseImageMorphismChoice" "../Identity/Individualit
beginlocale finite_particulars_with_individuality =
    ufo_particular_theory where Typ_q = Typ_q and Typ_p = Typ_p
  for
    Typ_p :: ‹'p itself› and
    Typ_q :: ‹'q itself› +
  assumes
    finite_particulars[intro!,simp]: ‹finite 𝒫› and
    particulars_have_individuality: ‹𝒫 = 𝒫_ind›
beginlemma mono_to_iso_is_an_iso:
  assumes
    ‹particular_struct_injection Γ Γ' φ›
    ‹∃σ. particular_struct_bijection Γ Γ' σ›
  shows ‹particular_struct_bijection Γ Γ' φ›
proof -
  interpret phi: particular_struct_injection Γ Γ' φ
    using assms by simp
  obtain σ where ‹particular_struct_bijection Γ Γ' σ›
    using assms by blast
  then interpret sigma: particular_struct_bijection Γ Γ' σ
    by simp
  have A: ‹finite (particulars Γ')›
    using finite_particulars sigma.morph_bijective
      sigma.morph_image_def
    by auto
  have ‹bij_betw φ 𝒫 phi.tgt.𝒫›
    using sigma.morph_bijective phi.morph_is_injective
    A finite_particulars
    by (metis Γ_simps(2) bij_betw_def card_image card_mono card_seteq phi.morph_image_de

          phi.morph_scope)
  then show ?thesis
    apply (unfold_locales)
    by (simp add: bij_betw_def)
qedlemma morphism_to_injective:
```

```
  assumes
    ‹particular_struct_morphism Γ Γ' φ›
    ‹∃σ. particular_struct_bijection Γ Γ' σ›
  shows ‹particular_struct_injection Γ Γ' φ›
proof -
  interpret phi: particular_struct_morphism Γ Γ' φ
    using assms by simp
  obtain σ where ‹particular_struct_bijection Γ Γ' σ›
    using assms by blast
  then interpret sigma: particular_struct_bijection Γ Γ' σ
    by simp
  interpret inv_sigma_phi:
    particular_struct_endomorphism Γ ‹sigma.inv_morph ∘ φ›
    apply (simp only: particular_struct_endomorphism_def
        ; intro conjI)
    subgoal by unfold_locales
    subgoal
      apply (intro particular_struct_morphism_comp[of _ Γ'] conjI)
      subgoal by unfold_locales
      using particular_struct_bijection_def particular_struct_injection_def by blast
    done
  have inv_sigma_phi: ‹sigma.inv_morph ∘ φ ∈ EndoMorphsΓ›
    using inv_sigma_phi.particular_struct_endomorphism_axioms
    by blast
  have P1: ‹finite (particulars Γ')›
    using finite_particulars sigma.morph_bijective[simplified]
    using bij_betw_finite by blast
  then have P2: ‹card (particulars Γ') = card 𝒫›
    using finite_particulars sigma.morph_bijective[simplified]
    using bij_betw_same_card by fastforce
  have ‹inj_on φ 𝒫›
  proof (rule ccontr)
    assume ‹¬ inj_on φ 𝒫›
    then obtain x y where
        AA: ‹x ∈ 𝒫› ‹y ∈ 𝒫› ‹x ≠ y› ‹φ x = φ y›
      by (meson inj_onI)
    obtain υ where BB: ‹particular_struct_endomorphism Γ υ›
        ‹υ x = y› ‹υ y = y›
      using phi.choice[simplified,OF AA(1,2,4)] by metis
    then have CC: ‹υ ∈ EndoMorphsΓ› by blast
    obtain A: ‹x ∈ 𝒫ind› ‹y ∈ 𝒫ind›
      using particulars_have_individuality AA by metis
```

```
    then obtain B: ‹¬ collapsable x› ‹¬ collapsable y›
      using particularsWithIndividualityE by blast
    show False
      using B(1)[simplified,rule_format,
        of υ,OF AA(1) CC AA(2)] AA(3) BB(2,3) by metis
  qed
  then show ?thesis
    by (unfold_locales  ; simp)
qedlemma  all_endomorphisms_are_perms: ‹EndoMorphsΓ ⊆ PermsΓ›
proof (intro subsetI permutations_I ; drule endomorphisms_D)
  fix φ
  assume A: ‹particular_struct_endomorphism Γ φ›
  then interpret phi: particular_struct_endomorphism  Γ φ .
  have B: ‹∃σ. particular_struct_bijection Γ Γ σ›
    apply (intro exI[of _ id])
    using particular_struct_permutation_def by blast
  then have C: ‹particular_struct_injection Γ Γ φ›
    using morphism_to_injective phi.particular_struct_morphism_axioms
    by metis
  then have D: ‹particular_struct_bijection Γ Γ φ›
    using mono_to_iso_is_an_iso B by metis
  then interpret phi_iso: particular_struct_bijection Γ Γ φ .
  show ‹particular_struct_permutation Γ φ›
    by (intro_locales)
qed

end


end
```

E.2 Isomorphism-invariant Instantiation

```
theory Instantiation
  imports "../ParticularStructures/MorphismImage" "../Universals/Universals"

begin

type_synonym ('q,'u) iof_predicate =
  ‹(ZF,'q) particular_struct ⇒ ZF ⇒ ZF set ⇒ 'u ⇒ bool›
```

```
locale iso_invariant_iof_predicate_sig =
  fixes zf_iof :: ‹('q,'u) iof_predicate› and
        Typ_q :: ‹'q itself› and
        Typ_u :: ‹'u itself›

locale iso_invariant_iof_predicate =
    iso_invariant_iof_predicate_sig  +
  assumes
    zf_iof_scope:
        ‹zf_iof Γ x w U ⟹ w ∈ ps_worlds Γ ∧ x ∈ w› and
    invariant_under_isomorphisms:
      ‹⟦ φ ∈ BijMorphs1_{Γ,TYPE(ZF)}
       ; x ∈ particulars Γ
       ; w ∈ ps_worlds Γ
        ⟧ ⟹
        zf_iof Γ x  w U ⟷ zf_iof (MorphImg φ Γ) (φ x) (φ ‘ w) U›
beginlemma invariant_under_isomorphisms_A:
  assumes
    ‹particular_struct_bijection_1 Γ φ›
    ‹x ∈ particulars Γ›
    ‹w ∈ ps_worlds Γ›
  shows ‹zf_iof Γ x  w U ⟷ zf_iof (MorphImg φ Γ) (φ x) (φ ‘ w) U›
  using invariant_under_isomorphisms[OF _ assms(2,3)]
    assms(1) by blast

end

context particular_struct_bijection_1
begin

declare src.isomorphism_1_iff_inj[simp del]

lemma invariant_under_isomorphisms_B:
  fixes zf_iof :: ‹('q,'u) iof_predicate›
  assumes
    ‹iso_invariant_iof_predicate zf_iof›
    ‹inj_on (f :: 'p_1 ⇒ ZF) (particulars Γ_1)›
    ‹inj_on (g :: 'p_2 ⇒ ZF) (φ ‘ particulars Γ_1)›
    ‹x ∈ src.𝒫›
    ‹w ∈ src.𝒲›
```

```
  shows ‹zf_iof (MorphImg f Γ1) (f x)  (f ` w) U ⟷
        zf_iof (MorphImg (g ∘ φ) Γ1) (g (φ x)) (g ` φ ` w) U›
proof -
  have inj_ZF[intro!,simp]: ‹∃(f :: ZF ⇒ ZF). inj f› using inj_on_id by blast
  have A[simp]: ‹ufo_particular_theory_sig.Γ 𝒲φ (◁φ) src.𝒬𝒮 (⇝φ) (⟶φ)
          = MorphImg φ Γ1›
    by (auto simp: ufo_particular_theory_sig.Γ_def
          particular_struct_morphism_image_simps)
  have A1[simp]: ‹src.Γ = Γ1› by auto
  interpret f: particular_struct_bijection_1 Γ1 f ‹TYPE('p1)› ‹TYPE(ZF)›
    using src.inj_morph_img_isomorphism[OF inj_on_subset[OF assms(2)]
          , simplified] by simp
  interpret g: particular_struct_bijection_1 ‹MorphImg φ Γ1› g ‹TYPE('p2)› ‹TYPE(ZF)›
    using tgt.inj_morph_img_isomorphism[OF inj_on_subset[OF assms(3)]
          , simplified] .
  have B[simp]: ‹MorphImg f.inv_morph (MorphImg f Γ1) = Γ1›
    by (metis f.particular_struct_bijection_axioms
          particular_struct_bijection.inv_is_bijective_morphism
          particular_struct_bijection_iff_particular_struct_bijection_1)
  have C: ‹f x ∈ f.tgt.𝒫› using assms(4) by blast
  have D: ‹f ` w ∈ f.𝒲φ› using assms(5) by blast
  interpret phi_zf: particular_struct_bijection_1 ‹MorphImg f Γ1› ‹g ∘ φ ∘ f.inv_morph›
‹TYPE(ZF)› ‹TYPE(ZF)›
    apply (intro particular_struct_bijection_1_comp ; (simp only: B)?)
    subgoal using particular_struct_bijection_iff_particular_struct_bijection_1
by blast
    subgoal using particular_struct_bijection_1_axioms by simp
    using g.particular_struct_bijection_1_axioms by blast

  interpret iso_invariant_iof_predicate zf_iof using assms(1) by simp
  have x_inv_f[simp]: ‹f.inv_morph (f x) = x› using assms(4) by auto
  have w_inv_f[simp]: ‹f.inv_morph ` f ` w = w› using assms(5) by auto
  show ?thesis
    by (simp only: invariant_under_isomorphisms_A[
      OF phi_zf.particular_struct_bijection_1_axioms, OF C D,of U
      ] o_apply image_comp[symmetric] x_inv_f w_inv_f
      ; simp)
qed


end
```

```
locale iso_instantiation_sig =
    ufo_particular_theory_sig where Typ_p = Typ_p and Typ_q = Typ_q +
    iso_invariant_iof_predicate_sig where Typ_q = Typ_q and Typ_u = Typ_u
  for
    Typ_p :: ‹'p itself› and
    Typ_q :: ‹'q itself› and
    Typ_u :: ‹'u itself›
begindefinition iof (‹_ ::_ _› [74,1,74] 75) where
  ‹x ::_w U ⟷
    (∃f. inj_on f 𝒫 ∧ x ∈ 𝒫
       ∧ w ∈ 𝒲
       ∧ zf_iof (MorphImg f Γ) (f x) (f ‘ w) U)›lemma iof_I[intro]:
  assumes ‹inj_on f 𝒫› ‹w ∈ 𝒲› ‹x ∈ 𝒫› ‹zf_iof (MorphImg f Γ) (f x) (f ‘ w) U›
  shows ‹x ::_w U›
  using assms by (auto simp: iof_def)

lemma iof_E1:
  assumes ‹x ::_w U›
  obtains f where ‹inj_on f 𝒫› ‹w ∈ 𝒲› ‹x ∈ 𝒫›  ‹zf_iof (MorphImg f Γ) (f x) (f
‘ w) U›
  using assms by (auto simp: iof_def)

lemma iof_scope_world:
  assumes ‹x ::_w U›
  shows ‹w ∈ 𝒲›
  using assms by (auto simp: iof_def)

end

sublocale iso_instantiation_sig ⊆ instantiation_sig
  where iof = iof .


context iso_instantiation_sig
begin
notation subsumes (infix ‹⊑› 75)

end

locale iso_instantiation =
    iso_instantiation_sig where Typ_p = Typ_p and Typ_q = Typ_q and Typ_u = Typ_u +
    ufo_particular_theory where Typ_p = Typ_p and Typ_q = Typ_q +
```

```
    iso_invariant_iof_predicate where Typ_q = Typ_q and Typ_u = Typ_u
  for
    Typ_p :: ‹'p itself› and
    Typ_q :: ‹'q itself› and
    Typ_u :: ‹'u itself›
begin

lemma iof_E[elim]:
  assumes ‹x ::_w U›
  obtains f where ‹inj_on f 𝒫› ‹x ∈ 𝒫› ‹w ∈ 𝒲› ‹U ∈ 𝒰› ‹zf_iof (MorphImg f Γ) (f
x) (f ‘ w) U›
      ‹x ∈ w›
proof -
  obtain f where A: ‹inj_on f ℰ› ‹w ∈ 𝒲› ‹x ∈ 𝒫› ‹zf_iof (MorphImg f Γ) (f x) (f
‘ w) U›
    using assms iof_E1 by blast
  have B: ‹U ∈ 𝒰› using assms by blast
  interpret f: particular_struct_bijection_1 Γ f
    apply simp
    using A(1) inj_on_id by blast
  obtain ‹f ‘ w ∈ f.𝒲_φ› ‹f x ∈ f ‘ w›
    using zf_iof_scope[OF A(4)] by metis
  then have ‹x ∈ w› using A(2,3)f.tgt_world_corresp_inv_image by force
  then show ?thesis using A B that  by metis
qed

lemma iof_scope:
  assumes ‹x ::_w U›
  shows ‹x ∈ 𝒫› ‹w ∈ 𝒲› ‹U ∈ 𝒰› ‹x ∈ w›
  using assms zf_iof_scope by blast+

endsublocale
  iso_instantiation ⊆ instantiation_thy
  where iof = iof
  by (unfold_locales ; blast)

locale iso_universals =
    iso_instantiation where Typ_p = Typ_p and Typ_q = Typ_q and Typ_u = Typ_u +
    universals where iof = iof and Typ_p = Typ_p and Typ_q = Typ_q and Typ_u = Typ_u

  for
    Typ_p :: ‹'p itself› and
```

```
    Typ_q :: ‹'q itself› and
    Typ_u :: ‹'u itself›
begin

lemma iof_E[elim]:
  assumes ‹x ::_w U›
  obtains f where ‹inj_on f 𝒫› ‹x ∈ 𝒫› ‹w ∈ 𝒲› ‹U ∈ 𝒰› ‹zf_iof (MorphImg f Γ) (f
x) (f ‘ w) U›
proof -
  obtain f where A: ‹inj_on f ℰ› ‹w ∈ 𝒲› ‹x ∈ 𝒫› ‹zf_iof (MorphImg f Γ) (f x) (f
‘ w) U›
    using assms iof_E1 by blast
  have B: ‹U ∈ 𝒰› using assms by blast
  show ?thesis using A B that by metis
qed

lemma iof_scope:
  assumes ‹x ::_w U›
  shows ‹x ∈ 𝒫› ‹w ∈ 𝒲› ‹U ∈ 𝒰›
  using assms by blast+

end


end
```

E.3 Instantiation in Finite Particular Structures

```
theory FiniteInstantiation
  imports FiniteParticularStructure Instantiation Trimming
begin

locale finite_instantiation =
  finite_particulars_with_individuality where Typ_p = Typ_p and Typ_q = Typ_q  +
  iso_universals where Typ_p = Typ_p and Typ_q = Typ_q and Typ_u = Typ_u
  for
    Typ_p :: ‹'p itself› and
    Typ_q :: ‹'q itself› and
    Typ_u :: ‹'u itself›
begin
```

```
context
  fixes φ
  assumes phi: ‹φ ∈ EndoMorphs_Γ›
begin

private lemma phi_perm: ‹φ ∈ Perms_Γ›
  using phi all_endomorphisms_are_perms by blast

interpretation phi: particular_struct_permutation Γ φ
  using phi_perm by blast

interpretation phi_iso: particular_struct_bijection_1 Γ φ
  by blast

lemma phi_detailingMoments_invariant[simp]: ‹φ ` Δ_U = Δ_U›
  using detailingMoments_invariant[
      OF phi_iso.particular_struct_bijection_1_axioms,of U,simplified]
  by metis

lemma phi_particulars_img[simp]: ‹φ ` 𝒫 = 𝒫›
  using phi.morph_is_surjective by auto


lemma phi_detailingMoment_complement_invariant[simp]: ‹φ ` (𝒫 - Δ_U) = 𝒫 - Δ_U›
  apply (rule inj_on_image_set_diff[OF phi.morph_is_injective,simplified
        ,of 𝒫 ‹Δ_U›,simplified])
  by (metis detailingMoments.cases endurantI1
      subsetI trans_inheres_in_scope
      all_ufo_particular_theory_axioms(4))

context
  fixes U :: 'u
begin

interpretation sigma: particular_struct_injection ‹trim U› Γ id
  by blast

interpretation phi_proj: particular_struct_bijection_1 ‹trim U› φ
proof -
  have A: ‹sigma.src.Γ = trim U› by auto
  show ‹particular_struct_bijection_1 (trim U) φ›
    apply (intro sigma.src.inj_morph_img_isomorphism[simplified A])
```

```
    subgoal
      apply (intro inj_on_subset[OF phi.morph_is_injective])
      by auto
    using sigma.tgt.injection_to_ZF_exist by blast
qed

lemma proj_phi_img[simp]: ‹MorphImg φ (trim U) = trim U›
  apply (intro particular_struct_eqI ext
      ; simp add: trim_simps trim_worlds_def trim_inheres_in_def
          trim_assoc_quale_def trim_towards_def
          particular_struct_morphism_image_simps
          ; (intro set_eqI iffI conjI CollectI)?
          ; (elim conjE exE CollectE)?
          ; hypsubst_thin? ; simp?)

  subgoal G1 premises P for w
    using phi.morph_worlds_correspond_src_tgt[simplified,OF P]
    apply (elim exE phi.world_corresp_E ; simp)
    subgoal premises Q for w1
      supply R1 =  Q(1,2,3) Q(1,2)[THEN worlds_are_made_of_particulars,THEN subsetD]
      supply R2 = R1(4,5)[THEN Q(3)]
      apply (intro exI[of _ w1] conjI Q set_eqI iffI DiffI
            ; (elim imageE ; simp ; elim conjE)? ; (elim DiffE)?)
      subgoal G1_1 for x x1
        using R1 R2
        by blast
      subgoal G1_2 for x x1
        using R1(4) phi_detailingMoment_complement_invariant by blast
      subgoal G1_3 for x
        apply (subst inj_on_image_set_diff[of φ 𝒫 w ‹Δ_U›] ; (intro DiffI)? ; simp?)
        subgoal using phi_iso.morph_is_injective by auto
        subgoal using P by blast
        subgoal
          by (metis inherence_scope inherence_sig.ℳ_E instantiation_sig.detailingMoments.case

                subsetI trans_inheres_in_scope)
        subgoal by (metis (mono_tags, lifting) R1(3) R1(5) image_iff phi_particulars_img)
        done
      done
    done
  subgoal G2 premises P for w
    using phi.morph_worlds_correspond_tgt_src[simplified,OF P]
```

```
    apply (elim exE phi.world_corresp_E ; simp)
    subgoal premises Q for w1
      supply R1 =  Q(1,2,3) Q(1,2)[THEN worlds_are_made_of_particulars,THEN subsetD]
      supply R2 = R1(4,5)[THEN Q(3)]
      apply (rule exI[of _ ‹w1 - ΔU›] ; intro conjI Q set_eqI iffI DiffI
            ; (elim imageE ; simp ; elim conjE)? ; (elim DiffE imageE ; simp?)?)
      subgoal G1_1 for x
        apply (subst  inj_on_image_set_diff[of φ 𝒫 w1 ‹ΔU›] ; (intro DiffI)? ; simp?)
        subgoal using phi_iso.morph_is_injective by auto
        subgoal using R1(4) by auto
        subgoal
          by (metis inherence_scope inherence_sig.ℳ_E instantiation_sig.detailingMoments

                subsetI trans_inheres_in_scope)
        by (metis (mono_tags, lifting) R1(3) R1(5) image_iff phi_particulars_img)

      subgoal G1_2 for x x1 using R2(1) by blast
      subgoal G1_3 for x y using R1(4) phi_detailingMoment_complement_invariant
by blast
      by (intro exI[of _ w1] conjI R1 ; simp )
    done
  subgoal G3 for x y
    using phi.morph_reflects_inherence[simplified,of x y] inherence_scope[of x y]
by metis
  subgoal G4 using phi_detailingMoment_complement_invariant by blast
  subgoal G5 by (metis G3 G4 instantiation_sig.detailingMoments.intros(2) tranclp.r_into_
  subgoal G6 for x y
    apply (rule exI[of _ ‹phi.inv_morph x›] ; rule exI[of _ ‹phi.inv_morph y›]
            ; frule inherence_scope ; intro conjI ; (elim conjE)? ; simp?)
    subgoal G6_1 using phi.inv_inheres_in_reflects_on_image by auto
    subgoal G6_2 by (metis Γ_simps(2) image_eqI phi.inv_morph_morph phi_detailingMoments
    subgoal G6_3 by (metis Γ_simps(2) image_eqI phi.inv_morph_morph phi_detailingMoments
    done
  subgoal G7 for x y
    apply (rule phi.morph_reflects_quale_assoc[simplified,THEN iffD1] ; simp?)
    by (simp add: assoc_quale_scopeD(1))
  subgoal G8 by (metis G4 assoc_quale_scopeD(3) inherence_sig.ℳ_E instantiation_sig.det
tranclp.r_into_trancl)
  subgoal G9 for x q
    apply (rule exI[of _ ‹phi.inv_morph x›]
            ; frule assoc_quale_scopeD ; intro conjI ; (elim conjE)? ; simp?)
    subgoal by (smt Γ_simps(2) imageE particular_struct_endomorphism_def particular_stru
```

```
particular_struct_morphism_def pre_particular_struct_morphism.morph_reflects_quale_assoc
phi.particular_struct_endomorphism_axioms phi.particular_struct_injection_axioms
phi_particulars_img ufo_particular_theory_sig.Γ_simps(4))
    by (metis Γ_simps(2) image_eqI phi.inv_morph_morph phi_detailingMoments_invariant)

  subgoal G10 for x y
    by (rule phi.morph_reflects_towardness[simplified,THEN iffD2] ; simp?
            ; frule towardness_scopeD ; simp? ; auto)
  subgoal G11 by (meson G4 𝓜_E detailingMoments.sub_moments towardness_apply_to_moments
tranclp.r_into_trancl)
  subgoal G12 using phi_detailingMoment_complement_invariant by blast
  subgoal G13 for x y
    apply (rule exI[of _ ‹phi.inv_morph x›] ; rule exI[of _ ‹phi.inv_morph y›]
            ; frule towardness_scopeD ; intro conjI ; (elim conjE)? ; simp?)
    subgoal by (metis Γ_simps(2) phi.inv_towardness_reflects towardness_scopeD(2)
towardness_scopeD(3) ufo_particular_theory_sig.Γ_simps(5))
    subgoal by (metis (mono_tags, hide_lams) Γ_simps(2) image_eqI phi.inv_morph_morph
phi_detailingMoments_invariant towardness.towardness_scopeE towardness_axioms)
    subgoal by (metis (full_types) Γ_simps(2) image_eqI phi.inv_morph_morph phi_detailingMomen
towardness.towardness_scopeE towardness_axioms)
    subgoal by (simp add: endurantI1)
    by (simp add: towardness_scopeD(3))
  done

lemma phi_in_trim_perm: ‹φ ∈ Perms_trim U›
  apply (intro permutations_I)
  using phi_proj.particular_struct_bijection_axioms[simplified, simplified proj_phi_img]
  by (simp add: particular_struct_endomorphism.intro particular_struct_bijection_def

            particular_struct_injection_def particular_struct_permutation.intro
            sigma.src.particular_struct_axioms)

end

end

end

end
```

E.4 Universal Trimming Operation

theory *Trimming*
  imports *Instantiation "../ParticularStructures/Permutability"*
begincontext *instantiation_sig*
begin

E.4.1 Detailing moments

inductive_set *detailingMoments ::*
    ‹'u ⇒ 'p set›
    (‹Δ_› [999] 1000)
    for *U*
    where
  non_char_moments:
    ‹⟦ U ∈ $\mathcal{U}_S$; x ∈ Insts U ; y ◁ x ;
     ∀ w u. y ::$_w$ u ⟶ ¬ char_by U u
    ⟧ ⟹ y ∈ Δ_U›
  | *sub_moments:* ‹⟦ x ∈ Δ_U ; y ◁◁ x ⟧ ⟹ y ∈ Δ_U›**definition** ‹trim_worlds U ≡ { w - Δ_U | w . w ∈ $\mathcal{W}$ }›

lemma *trim_worlds_I:*
  assumes ‹w ∈ $\mathcal{W}$› ‹w' = w - Δ_U›
  shows ‹w' ∈ trim_worlds U›
  using *assms* **by** *(auto simp: trim_worlds_def)*

lemma *trim_worlds_E:*
  assumes ‹w' ∈ trim_worlds U›
  obtains *w* **where** ‹w ∈ $\mathcal{W}$› ‹w' = w - Δ_U›
  using *assms* **by** *(auto simp: trim_worlds_def)***definition** ‹trim_inheres_in U x y ≡ x ◁ y ∧ x ∉ Δ_U ∧ y ∉ Δ_U›

lemma *trim_inheres_in_I:*
  assumes ‹x ◁ y› ‹x ∉ Δ_U› ‹y ∉ Δ_U›
  shows ‹trim_inheres_in U x y›
  using *assms* **by** *(auto simp: trim_inheres_in_def)*

lemma *trim_inheres_in_E:*
  assumes ‹trim_inheres_in U x y›
  obtains ‹x ◁ y› ‹x ∉ Δ_U› ‹y ∉ Δ_U›
  using *assms* **by** *(auto simp: trim_inheres_in_def)*

lemma *trim_inheres_in_D:*
assumes ‹*trim_inheres_in U x y*›
shows ‹$x \triangleleft y$› ‹$x \notin \Delta_U$› ‹$y \notin \Delta_U$›
using *assms* **by** *(auto simp: trim_inheres_in_def)*

definition ‹*trim_assoc_quale U x q* $\equiv x \rightsquigarrow q \wedge x \notin \Delta_U$›

lemma *trim_assoc_quale_I:*
assumes ‹$x \rightsquigarrow q$› ‹$x \notin \Delta_U$›
shows ‹*trim_assoc_quale U x q*›
using *assms* **by** *(auto simp: trim_assoc_quale_def)*

lemma *trim_assoc_quale_E:*
assumes ‹*trim_assoc_quale U x q*›
obtains ‹$x \rightsquigarrow q$› ‹$x \notin \Delta_U$›
using *assms* **by** *(auto simp: trim_assoc_quale_def)*

lemma *trim_assoc_quale_D:*
assumes ‹*trim_assoc_quale U x q*›
shows ‹$x \rightsquigarrow q$› ‹$x \notin \Delta_U$›
using *assms* **by** *(auto simp: trim_assoc_quale_def)*

definition ‹*trim_towards U x y* $\equiv$
$x \longrightarrow y \wedge x \notin \Delta_U \wedge y \notin \Delta_U$›

lemma *trim_towards_I:*
assumes ‹$x \longrightarrow y$› ‹$x \notin \Delta_U$› ‹$y \notin \Delta_U$›
shows ‹*trim_towards U x y*›
using *assms* **by** *(auto simp: trim_towards_def)*

lemma *trim_towards_E:*
assumes ‹*trim_towards U x y*›
obtains ‹$x \longrightarrow y$› ‹$x \notin \Delta_U$› ‹$y \notin \Delta_U$›
using *assms* **by** *(auto simp: trim_towards_def)*

lemma *trim_towards_D:*
assumes ‹*trim_towards U x y*›
shows ‹$x \longrightarrow y$› ‹$x \notin \Delta_U$› ‹$y \notin \Delta_U$›
using *assms* **by** *(auto simp: trim_towards_def)*

definition ‹*trim_quality_spaces U* $\equiv$
$\{ Q \mid Q\ x\ q\ .\ Q \in \mathcal{QS} \wedge q \in Q$

```
                    ∧ trim_assoc_quale U x q }›

lemma trim_quality_spaces_I:
  assumes ‹Q ∈ 𝒬𝒮› ‹q ∈ Q› ‹trim_assoc_quale U x q›
  shows ‹Q ∈ trim_quality_spaces U›
  using assms by (auto simp: trim_quality_spaces_def)

lemma trim_quality_spaces_E:
  assumes ‹Q ∈ trim_quality_spaces U›
  obtains q x where ‹Q ∈ 𝒬𝒮› ‹q ∈ Q› ‹trim_assoc_quale U x q›
  using assms by (auto simp: trim_quality_spaces_def)

lemma trim_quality_spaces_subset: ‹trim_quality_spaces U ⊆ 𝒬𝒮›
  by (simp add: trim_quality_spaces_def)definition ‹trim U = ⦇
    ps_quality_spaces = trim_quality_spaces U,
    ps_worlds = trim_worlds U,
    ps_inheres_in = trim_inheres_in U,
    ps_assoc_quale = trim_assoc_quale U,
    ps_towards = trim_towards U
  ⦈›abbreviation ‹trim_non_permutables U ≡
  ufo_particular_theory_sig.non_permutables
    (trim_worlds U)
    (trim_inheres_in U)
    (trim_quality_spaces U)
    (trim_assoc_quale U)
    (trim_towards U)›

lemma trim_simps:
  ‹ps_quality_spaces (trim U) = trim_quality_spaces U›
  ‹ps_worlds (trim U) = trim_worlds U›
  ‹ps_inheres_in (trim U) = trim_inheres_in U›
  ‹ps_assoc_quale (trim U) = trim_assoc_quale U›
  ‹ps_towards (trim U) = trim_towards U›
  by (auto simp: trim_def)

lemma trim_particulars[simp]:
   ‹possible_worlds_sig.𝒫 (ps_worlds (trim U)) = 𝒫 - Δ_U›
  apply (auto simp: trim_simps possible_worlds_sig.𝒫_def)
  subgoal by (metis Diff_iff trim_worlds_E)
  subgoal using trim_worlds_E by auto
  subgoal using trim_worlds_def by auto
  done
```

```
lemma trim_particulars_1[simp]:
  ‹possible_worlds_sig.𝒫 (trim_worlds U) = 𝒫 - Δ_U›
  using trim_particulars[simplified trim_simps] .


lemma trim_moments[simp]: ‹inherence_sig.ℳ (ps_inheres_in (trim U)) = ℳ - Δ_U›
  apply (auto simp: inherence_sig.ℳ_def trim_simps)
  subgoal G1 using trim_inheres_in_D(1) by blast
  subgoal G2 by (simp add: trim_inheres_in_def)
  subgoal for y x
    apply (intro exI[of _ x] trim_inheres_in_I ; simp?)
    apply (rule ccontr ; simp)
    subgoal premises P
      using P(3,1,2) apply (induct x arbitrary: y rule: detailingMoments.induct
; simp?)
      subgoal G3_1 premises P for x1 x2 x3
        supply R1 = tranclp.intros(1)[of ‹(◁)›,OF P(6)]
        supply R2 = tranclp.intros(2)[OF R1 P(3)]
        supply R3 = detailingMoments.intros(2)[of x1 U x3,OF _ R2, THEN P(5)[THEN
notE]]
        supply R4 = detailingMoments.intros(1)[OF P(1,2,3,4)]
        apply (rule R3)
        using detailingMoments.intros(2)[OF R4 R1] P(5) by simp
      subgoal G3_2 premises P for x1 x2 x3
        supply R1 = tranclp_trans[OF tranclp.intros(1)[of ‹(◁)›, OF P(5)] P(3)]
        using detailingMoments.intros(2)[OF P(1) R1] P(4) by simp
      done
    done
  done

lemma trim_moments_1[simp]: ‹inherence_sig.ℳ (trim_inheres_in U) = ℳ - Δ_U›
  using trim_moments[simplified trim_simps] .


end

context iso_universals
begin

notation detailingMoments (‹Δ_› [999] 1000)
```

```
context
  fixes φ :: ‹'p ⇒ 'p₁› and Γ'
  assumes A: ‹particular_struct_bijection_1 Γ φ›
begin

interpretation phi: particular_struct_bijection_1 Γ φ using A by simp

lemma phi_tgt_iof_1:
  assumes ‹x ∈ φ ` 𝒫› ‹w ∈ ((`) φ) ` 𝒲›
  shows
  ‹iso_instantiation_sig.iof phi.𝒲φ phi.img_inheres_in phi.tgt.𝒬𝒮
      phi.img_assoc_quale phi.img_towards zf_iof x w u ⟷
    (∃x₁ w₁. x = φ x₁ ∧ w = φ ` w₁ ∧ x₁ ::w₁ u)›
proof -
  obtain x₁ where x1[simp]: ‹x₁ ∈ 𝒫› ‹x = φ x₁› using assms(1) by blast
  obtain w₁ where w1[simp]: ‹w₁ ∈ 𝒲› ‹w = φ ` w₁› using assms(2) by blast
  obtain f₁ :: ‹'p ⇒ ZF› where f1: ‹inj f₁›
    using phi.src.injection_to_ZF_exist by blast
  obtain f₂ :: ‹'p₁ ⇒ ZF› where f2: ‹inj f₂›
    using phi.tgt.injection_to_ZF_exist by blast
  have A: ‹ufo_particular_theory_sig.Γ ((`) φ ` phi.src.𝒲) phi.img_inheres_in
            phi.tgt.𝒬𝒮 phi.img_assoc_quale phi.img_towards =
            MorphImg φ Γ›
    using phi.morph_image_tgt_struct phi.morph_image_worlds_src by auto

  have C[simp]: ‹φ ` w₂ = φ ` w₁ ⟷ w₂ = w₁› if ‹w₁ ∈ 𝒲› ‹w₂ ∈ 𝒲› for w₁ w₂
    using that phi.phi_inv_phi_world by fastforce
  have D[simp]: ‹possible_worlds_sig.𝒫 ((`) φ ` phi.src.𝒲) = φ ` 𝒫›
    using phi.morph_image_worlds_src phi.morph_is_surjective by auto
  have E: ‹inj_on (f ∘ φ) ℰ ⟷ inj_on f (φ ` ℰ)› for f :: ‹'p₁ ⇒ ZF›
    using inj_on_imageI[of f φ ℰ] comp_inj_on[OF phi.morph_is_injective[simplified]]
    by metis
  have F: ‹f ` φ ` w₂ = (λx. f (φ x)) ` w₂› for f :: ‹'p₁ ⇒ ZF› and w₂
    by auto
  have G[simp,intro!]: ‹inj_on f₂ (φ ` ℰ)›
    using f2 inj_on_subset by auto
  show ?thesis
    apply (intro iffI iso_instantiation_sig.iof_I[OF inj_on_subset[OF f2]]
      ; simp only: phi.morph_image_worlds_src D
      ; elim iso_instantiation_sig.iof_E1 exE conjE imageE
      ; hypsubst_thin? ; (simp only: A D)?
      ; simp)
```

```
    subgoal premises P for f w₂
      apply (intro exI[of _ x₁] ; simp? ; intro exI[of _ w₂] ; simp)
      apply (intro iof_I[of ‹f ∘ φ›,simplified E,OF P(1,3),of x₁,simplified])
      using P(2)[simplified,simplified F]  .
    subgoal premises P for x₂ w₂ f
      using
        phi.invariant_under_isomorphisms_B[OF iso_invariant_iof_predicate_axioms
          , simplified,OF P(1) _ P(3,2),OF G,
          THEN iffD1,OF P(4)]
      .
    done
qed

lemma phi_tgt_iof_iff:
  ‹iso_instantiation_sig.iof phi.𝒲φ phi.img_inheres_in phi.tgt.𝒬𝒮
      phi.img_assoc_quale phi.img_towards zf_iof x w u ⟷
    (∃x₁ w₁. x = φ x₁ ∧ w = φ ‘ w₁ ∧ x₁ ::w₁ u)› (is ‹?P ⟷ ?Q›)
proof -
  have A0: ‹x ∈ φ ‘ 𝒫 ∧ w ∈ ((‘) φ) ‘ 𝒲› (is ‹?A›)
    if ‹iso_instantiation_sig.iof phi.𝒲φ phi.img_inheres_in phi.tgt.𝒬𝒮
      phi.img_assoc_quale phi.img_towards zf_iof x w u ∨
      (∃x₁ w₁. x = φ x₁ ∧ w = φ ‘ w₁ ∧ x₁ ::w₁ u)›
    using that
    by (metis iso_instantiation_sig.iof_E1 phi.morph_image_simps(1) phi.morph_image_worlds_sr
phi.morph_preserves_particulars phi.world_preserve_img ufo_particular_theory_sig.Γ_simps(2))


  have A1: ‹P ⟷ Q› if ‹A ⟹ P ⟷ Q› ‹P ∨ Q ⟹ A›  for A  P Q
    using that by blast
  show ?thesis
    apply (rule A1[where A = ?A and P = ?P and Q = ?Q])
    subgoal using phi_tgt_iof_1 by metis
    using A0 by simp
qed

lemma phi_src_iof_iff:
  ‹iso_instantiation_sig.iof phi.𝒲φ phi.img_inheres_in phi.tgt.𝒬𝒮
      phi.img_assoc_quale phi.img_towards zf_iof (φ x)(φ ‘ w) u
   ∧ x ∈ 𝒫 ∧ w ∈ 𝒲  ⟷ x ::w u›
proof (simp only: phi_tgt_iof_iff ; intro iffI conjI ; (elim exE conjE)?)
  fix x₁ w₁
  assume as: ‹x ∈ ℰ› ‹w ∈ 𝒲› ‹φ x = φ x₁› ‹φ ‘ w = φ ‘ w₁› ‹x₁ ::w₁ u›
```

```
  obtain A: ‹x = x₁› ‹w = w₁›
    using that as(1,2,3,4) phi.morph_is_injective[simplified] worlds_are_made_of_particul
    by (metis as(5) inj_onD inj_on_image_eq_iff iof_E1
              worlds_are_made_of_particulars)
  show ‹x ::w u› using A as by simp
next
  assume as: ‹x ::w u›
  show ‹x ∈ ℰ› using as by blast
  show ‹w ∈ 𝒲› using as by blast
  show ‹∃x₁ w₁. φ x = φ x₁ ∧ φ ‘ w = φ ‘ w₁ ∧ x₁ ::w₁ u›
    by (intro exI[of _ x] exI[of _ w] as conjI ; simp)
qed

lemma phi_inheres_in_tranclp:
  ‹phi.tgt.inheresIn++ x y ⟷ (∃x₁ y₁. x₁ ◁◁ y₁ ∧ x = φ x₁ ∧ y = φ y₁)›
  apply (intro iffI)
  subgoal
    apply (rule exI[of _ ‹phi.inv_morph x›] ; rule exI[of _ ‹phi.inv_morph y›])
    apply (induct rule: tranclp.induct ; simp? ; intro conjI ; (elim conjE)?
          ; simp?)
    subgoal G1 for a b
      apply (intro phi.morph_reflects_inherence[of ‹phi.inv_morph a› ‹phi.inv_morph
b›
          , THEN iffD1, THEN
          tranclp.intros(1)[of ‹phi.src_inheres_in›],simplified])
      subgoal G1_1 using phi.morph_image_def phi.phi_inv_scope by auto
      subgoal G1_2 using phi.phi_inv_scope by auto
      by (metis Γ_simps(3) phi.inv_inheres_in_reflects phi.tgt.inherence_scope)

    subgoal G2 using phi.inv_morph_morph phi.tgt.inherence_scope by presburger
    subgoal G3 by auto
    subgoal G4 premises P for a b c
      apply (intro tranclp.intros(2)[OF P(3)])
      using P(1,2) by (metis Γ_simps(3) phi.inv_inheres_in_reflects phi.tgt.inherence_sco

    using G3 by blast
  subgoal
    apply (elim exE conjE ; hypsubst_thin)
    subgoal for x₁ y₂
      apply (induct rule: tranclp.induct)
      subgoal G1 for a b
        apply (intro tranclp.intros(1))
```

```
        using phi.morph_reflects_inherence by auto
      subgoal G2 premises P for a b c
        apply (intro tranclp.intros(2)[OF P(2)])
        using P(3) phi.morph_reflects_inherence by auto
      done
    done
  done

interpretation inst2:
  iso_universals ‹phi.tgt.𝒲› ‹phi.tgt_inheres_in› ‹phi.tgt.𝒬𝒮›
        ‹phi.tgt.assoc_quale› ‹phi.tgt.towards› ‹zf_iof› ‹TYPE('p₁)›
proof -
  have A: ‹x₂ ::phi.inv_morph ‘ w₂ u›
    if as: ‹φ x₁ ∈ phi.tgt.ℳ› ‹x₁ ::w₁ u›
          ‹w₂ ∈ phi.𝒲φ› ‹φ x₂ ∈ w₂› ‹x₂ ::w₃ u› for x₁ x₂ w₁ w₂ w₃ u
  proof -
    have AA: ‹x₁ ∈ ℰ›  using iof_E[OF as(2)] by metis
    obtain g :: ‹'p ⇒ ZF› where  BB: ‹inj_on g ℰ› ‹x₂ ∈ ℰ›
      using iof_E[OF as(5)] by blast

    interpret g: particular_struct_bijection_1 Γ g
      apply (intro inj_morph_img_isomorphism[of g] BB)
      using inj_on_id by blast

    have F: ‹x₁ ∈ ℳ› using as(1) AA by auto
    then have G: ‹u ∈ 𝒰ℳ› using as(2) by blast
    obtain H: ‹u ∈ 𝒰› ‹⋀x w₁ w₂. ⟦ x ::w₁ u ; w₂ ∈ 𝒲 ; x ∈ w₂ ⟧ ⟹ x ::w₂ u›
      using moment_universals_are_rigid[OF G,THEN rigidE] by metis


    show ‹x₂ ::phi.inv_morph ‘ w₂ u›
      apply (rule H(2)[of _ w₃,OF as(5)])
      subgoal  using Γ_simps(2) that(3) by blast
      using BB(2) phi.tgt_world_corresp_inv_image that(3) that(4) by force
  qed

  interpret phi: iso_instantiation
    where 𝒲 = phi.tgt.𝒲 and
          inheresIn = phi.tgt_inheres_in and
          𝒬𝒮 = phi.tgt.𝒬𝒮 and
          assoc_quale = phi.tgt.assoc_quale and
          towards = phi.tgt.towards and
```

```
        zf_iof = zf_iof and
        Typ_p = ‹TYPE('p_1)›
    by (unfold_locales)

  show ‹iso_universals phi.tgt.𝒲 phi.tgt_inheres_in phi.tgt.𝒬𝒮
      phi.tgt.assoc_quale phi.tgt.towards zf_iof›
  apply (unfold_locales)
  subgoal
    apply (auto simp: instantiation_sig.𝒰_𝒮_def instantiation_sig.𝒰_ℳ_def
                phi_tgt_iof_iff[simplified])
    subgoal premises P for U x_1 x_2 w_1 w_2
      supply R1 = 𝒰_ℳ_I[OF P(5)]
      supply R2 = 𝒰_𝒮_I[OF P(4)]
      supply R3 = substantial_moment_univs_separate[simplified disjoint_eq_subset_Compl
          , THEN subsetD,simplified,THEN notE]
      apply (rule R3[OF R2 R1])
      subgoal
        using phi.tgt.𝒮_I[OF P(1,2), simplified phi.morph_image_substantials[simplified]]
        using P(4)[THEN iof_scope(1)]
        using phi.morph_image_substantials phi.morph_preserves_substantials by auto
      subgoal
        using P(3)
        using P(5)[THEN iof_scope(1)]
        using phi.morph_image_substantials phi.morph_preserves_substantials by auto
      done
    done
  subgoal for u
    apply (auto simp: instantiation_sig.𝒰_𝒮_def instantiation_sig.𝒰_ℳ_def
                phi_tgt_iof_iff[simplified]
                instantiation_sig.rigid_def
                instantiation_sig.𝒰_def)
    subgoal for x_1 w_1 w_2 x_2 w_3
      apply (intro exI[of _ ‹x_2›] conjI exI[of _ ‹phi.inv_morph ` w_2›]
          ; simp)
      subgoal using A by metis
      done
    done
  done
qedlemma detailingMoments_invariant:
  ‹inst2.detailingMoments U = φ ` detailingMoments U›
proof -
  have A: ‹inst2.𝒰_𝒮 = 𝒰_𝒮›
```

```
    apply (simp only: inst2.𝒰_𝒮_def phi_tgt_iof_iff)
    apply (auto ; (elim 𝒰_𝒮_E)?)
    subgoal for u x w by (intro 𝒰_𝒮_I[of x w] ; auto)
    subgoal for u x w apply (rule exI[of _ ‹φ x›] ; intro conjI
            ; (rule exI[of _ ‹φ ‘ w›] ; rule exI[of _ x])?
            ; (intro conjI)?
            ; (rule exI[of _ w])? ; simp?)
      by auto
    done
  have B: ‹inst2.Insts U = φ ‘ Insts U›
    apply (intro set_eqI iffI ; (elim inst2.InstsE InstsE imageE)?
            ; simp only: phi_tgt_iof_iff
            ; (elim exE conjE)?
            ; hypsubst_thin)
    subgoal for _ _ x₁ w₁  by (intro imageI ; blast)
    subgoal for _ x w
      apply (intro inst2.InstsI[of _ ‹φ ‘ w›])
      subgoal premises P
        apply (rule iof_E[OF P])
        subgoal premises Q
          using phi_src_iof_iff[of x w U,simplified Q(2,3), simplified simp_thms]
P by simp
        done
      done
    done
  have C: ‹phi.img_inheres_in y x ⟷ (∃y₁ x₁. y₁ ◁ x₁ ∧ y = φ y₁ ∧ x = φ x₁)›
for x y
    by (metis (no_types, hide_lams) Γ_simps(2) Γ_simps(3) phi.I_img_eq_tgt_I phi.morph_image_I
phi.morph_image_def phi.morph_image_iff phi.morph_image_inheres_in_D(1) phi.morph_preserves_in
phi.morph_reflects_inherence phi.tgt.endurantI2)
  have D: ‹inst2.𝒰 = 𝒰›
    using 𝒰_def inst2.𝒰_def phi_tgt_iof_iff by auto
  have D1: ‹w₁ = w₂› if ‹w₁ ∈ 𝒲› ‹w₂ ∈ 𝒲› ‹φ ‘ w₁ = φ ‘ w₂› for w₁ w₂
    using that phi.morph_is_injective[simplified] worlds_are_made_of_particulars

    by (meson inj_on_image_eq_iff)

  have E: ‹inst2.char_by = char_by›
    apply (intro ext iffI char_by_I inst2.char_by_I ;
            elim char_by_E inst2.char_by_E ; (simp only: D)?)
    subgoal premises P for u₁ u₂ x w
      apply (rule P(4)[simplified phi_tgt_iof_iff, THEN exE,simplified C,of ‹φ x›
```

```
‹φ ‘ w›
                    , simplified] ; (elim exE conjE)?; hypsubst?)
      subgoal by (intro exI[of _ x] ; simp; intro exI[of _ w] ; simp add: P)
      subgoal premises Q for a x1 x2 x3 w1
        apply (rule P(1)[THEN iof_E])
        apply (rule Q(3)[THEN iof_E])
        subgoal premises T for f g
          supply R1 = phi.morph_is_injective[THEN inj_onD,simplified]
          supply R2 = Q(1)[THEN inherence_scope]
          supply R3 = R2[THEN conjunct1] R2[THEN conjunct2]
          supply R4 = Q(4)[THEN R1,OF T(7) R3(1)] Q(5)[THEN R1,OF T(2) R3(2)] D1[OF
T(3,8) Q(2)]
          using Q(1,3)[simplified R4[symmetric]] by blast
        done
      done
    subgoal premises P for u1 u2 x w
      using P(1) apply (simp only: phi_tgt_iof_iff)
      apply (elim exE conjE ; simp)
      subgoal premises Q for x1 w1
        using P(4)[OF Q(3)] apply (elim exE conjE)
        subgoal premises T for y
          apply (intro exI[of _ ‹φ y›] conjI exI[of _ y] exI[of _ w1] T(2); (simp
add: C)?)
          using T Q P by blast
        done
      done
    done
  have F: ‹x1 = x› if ‹y1 ◁ x1› ‹φ x1 = φ x› ‹x ∈ 𝒫› for x1 x y1
    using phi.morph_is_injective[THEN inj_onD,simplified,OF that(2)]
        inherence_scope[OF that(1)] that(3) by metis


  show ?thesis
  apply (intro set_eqI iffI)
  subgoal for x
    apply (induct x rule: inst2.detailingMoments.inducts ; (simp only: A B E phi_inheres_
    subgoal for x y
      apply (simp only: C[of y x] ; elim exE conjE)
      apply (simp only: phi_tgt_iof_iff ; elim imageE ; simp ; intro imageI)
      subgoal premises P for x2 x3 x4
        using P(7) apply (elim InstsE)
        subgoal premises Q for w
```

```
          using Q apply (elim iof_E)
          subgoal premises T for f
            supply R1 = F[OF P(3,6) T(2)]
            supply R2 = P(2)[simplified R1,rule_format,OF exI,OF conjI,OF _ exI,OF
_ conjI]
                        P(1,3,4,5)[simplified R1] Q[simplified R1] T[simplified R1]
R1
            apply (intro detailingMoments.intros(1)[OF P(1) P(7) R2(3)] allI impI
notI
                  ; elim char_by_E)
            subgoal premises O for w1 u
              using O(4)[OF ‹x4 ::w U›] apply (elim exE conjE)
              subgoal premises Z for z
                by (metis O(1) O(4) R2(1) T(4) instantiation_sig.char_by_I)
              done
            done
          done
        done
      done
    subgoal for x y
      apply (auto ; intro imageI ; rule detailingMoments.intros(2)[of ‹phi.inv_morph
x›] ; simp)
      subgoal by (metis Γ_simps(2) detailingMoments.cases inherence_scope inherence_sig.M_E

            particular_struct_bijection_1_def particular_struct_injection.inv_morph_morph

            phi.particular_struct_bijection_1_axioms trans_inheres_in_scope)
      subgoal premises P for x1 x2 x3
        using P(4)
        by (metis Γ_simps(2) particular_struct_injection.inv_morph_morph
                phi.particular_struct_injection_axioms trans_inheres_in_scope)
      done
    done
  subgoal
    apply (elim imageE ; hypsubst_thin)
    subgoal for x
      apply (induct x rule: detailingMoments.inducts)
      subgoal for x1 x2
        apply (intro inst2.detailingMoments.intros(1)[of _ ‹φ x1›] ; (simp only:
A B E phi_inheres_in_tranclp)?)
        subgoal G1 by auto
        subgoal G2
```

```
          apply (simp only: C)
          by blast
        subgoal G3
          apply (simp only: phi_tgt_iof_iff ; intro allI impI)
          subgoal premises P for w u
            using P(5) apply (elim exE conjE)
            subgoal premises Q for x3 w1
              apply (intro P(4)[rule_format,of w1 u])
              using Q(3) Q(3)[THEN iof_scope(1)] P(3) inherence_scope[OF P(3)]
                    phi.morph_is_injective[THEN inj_onD,simplified,OF Q(1)]
              by metis
            done
          done
        done
      subgoal for x1 x2
        apply (rule inst2.detailingMoments.intros(2)[of _ _ ‹φ x2›] ; (simp only:
A B E phi_inheres_in_tranclp)?)
        by blast
      done
    done
  done
qed

end


lemma trim_inheres_in_scope:
  assumes ‹trim_inheres_in U x y›
  shows ‹x ∈ 𝓜 - Δ_U› ‹y ∈ 𝓟 - Δ_U›
  subgoal using assms by (meson DiffI 𝓜_I trim_inheres_in_D(1) trim_inheres_in_D(2))
  using assms apply (elim trim_inheres_in_E)
  by blast

lemma trim_towards_scope:
  assumes ‹trim_towards U x y›
  shows ‹x ∈ 𝓜 - Δ_U› ‹y ∈ 𝓟 - Δ_U›
  subgoal using assms using trim_towards_def by auto
  using assms trim_towards_E by blast


lemma trim_assoc_quale_scope_E:
  assumes ‹trim_assoc_quale U x q›
```

```
  obtains ‹x ∈ 𝓜 - Δ_U› ‹q ∈ quality_space_sig.qualia (trim_quality_spaces U) ›
  using assms apply (elim trim_assoc_quale_E)
  apply (auto simp: quality_space_sig.qualia_def)
  subgoal premises P
    apply (rule P(1))
    subgoal using P(2,3) using assoc_quale_scopeD(3) by blast
    subgoal premises Q
      using assoc_quale_scopeD(2)[OF Q P(2)] apply (elim 𝒬_E)
      subgoal for A
        apply (rule bexI[of _ A] ; simp?)
        apply (rule trim_quality_spaces_I[of A q U x] ; simp?)
        by (intro trim_assoc_quale_I P Q)
      done
    done
  done

lemma trim_assoc_quale_scope_E_1:
  assumes ‹trim_assoc_quale U x q›
  obtains ‹x ∈ 𝓜 - Δ_U› ‹q ∈ quality_space_sig.qualia (ps_quality_spaces (trim U))
›
  using assms trim_assoc_quale_scope_E that[simplified trim_simps] by metis

lemma trim_substantials[simp]:
   ‹inherence_sig.𝒮 (ps_worlds (trim U)) (ps_inheres_in (trim U))  = 𝒮›
  apply (auto simp: inherence_sig.𝒮_def)
  subgoal premises P for x
    using P(3,1,2)
    apply (induct x rule: detailingMoments.induct)
    subgoal using inherence_scope by blast
    using trans_inheres_in_scope by blast
  done

lemma trim_substantials_1[simp]:
   ‹inherence_sig.𝒮 (trim_worlds U) (trim_inheres_in U) = 𝒮›
  using trim_substantials[simplified trim_simps] .

lemma trim_inheres_in_trancl[simp]:
  ‹(trim_inheres_in U)⁺⁺ x y ⟷ (◁)⁺⁺ x y ∧ x ∉ Δ_U ∧ y ∉ Δ_U›
  apply (intro iffI conjI ; (elim conjE)?)
  subgoal G1
    apply (induct rule: tranclp.induct)
    subgoal G1_1 by (simp add: tranclp.r_into_trancl trim_inheres_in_D(1))
```

```
    subgoal G1_2 premises P for a b c
      using P(3) apply (elim trim_inheres_in_E)
      subgoal premises Q
        using P(2) Q(1) tranclp.intros(2) by metis
      done
    done
  subgoal G2
    by (induct rule: tranclp.induct ; elim trim_inheres_in_E ; simp)
  subgoal G3
    by (induct rule: tranclp.induct ; elim trim_inheres_in_E ; simp)
  subgoal premises P
    using P apply (induct rule: tranclp.induct)
    subgoal premises Q for a b
      using trim_inheres_in_I[OF Q, THEN tranclp.intros(1)[of ‹trim_inheres_in U›]]
.
    subgoal premises Q for a b c
      using Q
      by (metis (full_types) detailingMoments.simps
            trim_inheres_in_I tranclp.r_into_trancl tranclp_trans)
    done
  done

context
  fixes U :: 'u
begin

interpretation trim: possible_worlds ‹trim_worlds U›
  apply (unfold_locales ; simp? ; (elim conjE)?)
  subgoal using injection_to_ZF_exist by blast
  subgoal using at_least_one_possible_world trim_worlds_I by fastforce
  subgoal by (meson Diff_iff instantiation_sig.trim_worlds_I particulars_do_not_exist_in_
  done

lemma trim_ed[simp]: ‹trim.ed x y ⟷ ed x y ∧ x ∉ Δ_U ∧ y ∉ Δ_U›
  apply (intro iffI conjI ; elim trim.edE conjE edE ; (intro edI trim.edI)?
          ; simp? ; (elim conjE)? ; simp?)
  subgoal premises P for w
    using P(1)[OF trim_worlds_I,OF P(2), of ‹w - Δ_U›,simplified,
              simplified P,simplified] .
  subgoal premises P for w
    using P(6) apply (elim trim_worlds_E)
    subgoal premises Q for w_1
```

```
      supply R1 = P(7)[simplified Q(2),simplified,THEN conjunct1]
      supply R2 = P(5)[OF Q(1)]
      by (simp add: Q(2) R2[OF R1] P(2))
    done
  done

interpretation trim: inherence_base ‹trim_worlds U› ‹trim_inheres_in U›
  apply (unfold_locales ; (intro conjI)? ; simp?
          ; (intro conjI)? ; (elim trim_inheres_in_E)? ; simp?)
  subgoal by blast
  subgoal by blast
  subgoal by (simp add: inherence_imp_ed)
  subgoal by (simp add: moment_non_migration)
  done

interpretation trim: inherence ‹trim_worlds U› ‹trim_inheres_in U›
  apply (unfold_locales)
  subgoal
    apply (rule wf_subset[to_pred,OF inherence_is_noetherian] ; simp)
    by (auto elim: trim_inheres_in_E)
  subgoal
    apply (rule wf_subset[to_pred,OF inherence_is_wf])
    by (auto elim: trim_inheres_in_E)
  done

interpretation trim: quality_space ‹trim_quality_spaces U›
  apply (unfold_locales ; (intro notI)? ; (elim trim_quality_spaces_E trim_assoc_quale_E)?
; simp?)
  using quality_spaces_are_disjoint by auto

interpretation trim: qualified_particulars ‹trim_worlds U› ‹trim_inheres_in U› ‹trim_quality_spa
U›
                        ‹trim_assoc_quale U›
  apply (unfold_locales ; simp? ; (elim conjE)?)
  subgoal G1 for x q
    apply (elim trim_assoc_quale_E ; intro conjI)
    subgoal using assoc_quale_scopeD(3) by blast
    subgoal by blast
    subgoal by (metis instantiation_sig.trim_assoc_quale_def trim_assoc_quale_scope_E)
    done
  subgoal G2 for x q1 q2
    apply (elim trim_assoc_quale_E)
```

```
    by (simp add: assoc_quale_unique)
  subgoal G3 for w y1 y2 x q1 q2 Q
    apply (elim trim_assoc_quale_E trim_inheres_in_E trim_worlds_E trim_quality_spaces_E
; simp)
    by (simp add: quality_moment_unique_by_quality_space)
  subgoal G4 for Q
    by (elim trim_quality_spaces_E ; blast)
  subgoal G5 for y1 x y2 q
    apply (elim trim_assoc_quale_E trim_inheres_in_E ; simp)
    by (simp add: quale_determines_moment)
  done

lemma trim_ultimate_bearer[simp]:
  assumes ‹x ∈ trim.𝒫›
  shows ‹trim.ultimateBearer x = !β x›
proof -
  have A: ‹x ∈ 𝒮 ⟹ x ∉ ℳ› using assms by blast
  show ?thesis
    apply (rule trim.ultimate_bearer_eq_simp[THEN iffD2] ; simp)
    using assms apply (cases x rule: trim.endurant_cases)
    subgoal substantials
      apply simp
      subgoal premises P
        using assms by force
      done
    subgoal premises P
      using trim.ultimate_bearer_ex1I[OF P] apply (simp)
      apply (elim ex1E conjE ; simp)
      by (meson endurantI1 trans_inheres_in_scope)
    subgoal premises P
      apply (intro conjI)
      subgoal G1
        using assms by auto
      subgoal G2
        by (smt (z3) assms endurantI3 inherence_sig.𝒮_I iso_universals.trim_substantials_
iso_universals_axioms noetherian_inherence.ultimate_bearer_eqI1 noetherian_inherence.ulti
noetherian_inherence_axioms rtranclp.rtrancl_refl trim.endurantI3 trim.noetherian_inheren
trim.ultimate_bearer_ex1I trim_inheres_in_trancl)
      done
    done
qed
```

```
interpretation trim: towardness ‹trim_worlds U› ‹trim_inheres_in U› ‹trim_towards U›
  apply (unfold_locales)
  subgoal for x y
    apply (elim trim_towards_E ;  simp)
    using towardness_scope by simp
  subgoal for x y
    apply (elim trim_towards_E ;  simp)
    using towardness_imp_ed by simp
  subgoal for x y
    apply (elim trim_towards_E)
    subgoal premises P
      supply R1 = towardness_scopeD[OF P(1)]
      supply R2 = trim_ultimate_bearer[of x, simplified P(1) trim_particulars_1,OF
DiffI, OF R1(2) P(2)]
      using  P R1 R2
      apply (simp only: R2)
      by blast
    done
  subgoal for x y1 y2
    apply (elim trim_towards_E)
    by (simp add: towardness_single)
  done

interpretation trim: ufo_particular_theory ‹trim_worlds U› ‹trim_inheres_in U›
    ‹trim_quality_spaces U› ‹trim_assoc_quale U› ‹trim_towards U›
  apply (unfold_locales)
  apply (elim trim.qualifiedParticularsE trim_assoc_quale_E ; intro notI
          ; elim trim_inheres_in_E)
  using qualified_particulars_are_not_bearers by blastlemma trim_particular_struct[intro!,si

  ‹particular_struct (trim U)›
  apply (simp only: particular_struct_def trim_simps)
  by intro_locales

interpretation trim_morph: pre_particular_struct_morphism ‹trim U› Γ id
  apply (simp only: pre_particular_struct_morphism_def ; intro conjI)
  subgoal by simp
  subgoal by blast
  apply (unfold_locales ; simp add: trim_simps)
  subgoal by (simp add: trim_quality_spaces_def)
  subgoal by (simp add: trim_inheres_in_def)
```

```
  subgoal for x z
    apply auto
    by (metis instantiation_sig.detailingMoments.simps tranclp.r_into_trancl)
  subgoal by (simp add: trim_towards_def)
  subgoal for x z
    apply auto
    using trim.endurantI3 by auto
  by (simp add: trim_assoc_quale_def)


interpretation trim_morph: particular_struct_morphism ‹trim U› Γ id
  apply (unfold_locales ; simp add: trim_simps)
  subgoal
    apply (elim trim_worlds_E ; hypsubst_thin)
    subgoal for w
      apply (intro exI[of _ ‹w›])
      apply (intro trim_morph.world_corresp_I[simplified] ; (simp add: trim_simps)?)
      by (intro trim_worlds_I[of  w] ; simp)
    done
  subgoal for w_t
    apply (intro exI[of _ ‹w_t - Δ_U›] trim_morph.world_corresp_I[simplified]
           ; (simp add: trim_simps)? )
    by (intro trim_worlds_I[of  w_t] ; simp)
  donelemma trim_injective[intro!,simp]:
  ‹particular_struct_injection (trim U) Γ id›
  by (unfold_locales ; intro inj_on_id)


lemma trim_worlds_img: ‹∃ w_1 ∈ 𝒲. w = w_1 - Δ_U› if ‹w ∈ trim_morph.src.𝒲›
  using that apply (simp add: trim_simps)
  by (elim trim_worlds_E ; blast)

end

lemma detailingMoments_mono[intro]:
  assumes ‹U_1 ⊑ U_2›
  shows ‹Δ_U_1 ⊆ Δ_U_2›
proof -
  have A: ‹x ∈ Insts U_2› if ‹x ∈ Insts U_1› for x
    using that assms subsumesE InstsI InstsE by metis
  have B: ‹x ::_w U_2› if ‹x ::_w U_1› for x w
    using that assms subsumesE  by metis
```

```
  have C: ‹U₂ ∈ 𝒰_𝒮› if ‹U₁ ∈ 𝒰_𝒮›
  proof -
    obtain x w where ‹x ::_w U₂› ‹x ∈ 𝒮›
      using assms
      by (meson B 𝒰_𝒮_E ‹U₁ ∈ 𝒰_𝒮›)
    then show ?thesis using 𝒰_𝒮_I by blast
  qed
  obtain D: ‹U₁ ∈ 𝒰› ‹U₂ ∈ 𝒰›
    using assms by blast
  show ?thesis
    apply (intro subsetI)
    subgoal for u₁
      apply (induct rule: detailingMoments.induct)
      subgoal premises P for x y
        apply (intro detailingMoments.intros(1)[of U₂ x y] P A C allI impI notI
; elim char_by_E)
        subgoal premises Q for w u
          apply (rule P(4)[rule_format,THEN notE,of w u]
                ; (intro Q char_by_I D)?)
          using B by blast
        done
      subgoal premises P for x y
        by (intro detailingMoments.intros(2)[of x U₂ y] P)
      done
    done
qed


end

end
```

E.5 Formal Characterization of Sortality

```
theory Sortality
  imports Trimming FiniteInstantiation "../Identity/Identity"
begincontext finite_instantiation
beginlemma  permutables_preserved_under_trimming:
  fixes x and U :: 'u
  assumes A: ‹x ∈ 𝒮› ‹x ∉ 𝒫₁!›
  shows ‹x ∉ trim_non_permutables U›
```

```
proof -
  have A1: ‹x ∈ 𝒫› using A by blast
  obtain φ y where  B: ‹φ ∈ EndoMorphsΓ› ‹y ∈ ℰ› ‹(φ y = x) ≠ (y = x)›
    using A(2)[THEN notE, OF non_permutables_I,OF A1 non_permutable_I] by blast
  have B1: ‹y ∈ 𝒮›
    using B(3,2) A(1) apply auto
    subgoal premises P
      using P(3,4)
      by (metis B(1) endomorphisms_iff particular_struct_endomorphism_def
          particular_struct_morphism_def pre_particular_struct_morphism.morph_preserves_m

          ufo_particular_theory_sig.Γ_simps(3))
    done
  have ‹φ ∈ Permstrim U› using phi_in_trim_perm[OF B(1),of U] by simp
  then interpret I1: particular_struct_permutation ‹trim U› φ by simp
  obtain B2: ‹x ∈ I1.endurants› ‹y ∈ I1.endurants›
    using A(1) B1 I1.endurantI3 by auto
  let ?I1_Γ = ‹trim U›
  have I1_src_Γ: ‹I1.src.Γ = trim U› using I1.src.Γ_simps by blast
  have C: ‹φ ∈ EndoMorphs?I1_Γ›
    using I1.particular_struct_endomorphism_axioms by force
  have D: ‹trim_non_permutables U = I1.src.non_permutables›
    by (simp only: trim_simps)
  show ?thesis
    apply (simp only: D)
    apply (intro notI ; elim I1.src.non_permutables_E)
    subgoal premises P
      using I1.src.non_permutable_E[of x,simplified I1_src_Γ,OF P(2) C,of y,OF B2(2)]
B(3) by simp
    done
qedlemma  non_permutables_under_trimming_are_the_identifiable:
  fixes x and U
  assumes ‹x ∈ 𝒮› ‹x ∈ trim_non_permutables U›
  shows ‹x ∈ 𝒫=›
proof -
  have ‹x ∈ 𝒫1!›
    using permutables_preserved_under_trimming[OF assms(1)] assms(2) by metis
  then show ?thesis
    using identifiable_particulars_are_the_non_permutables
    by simp
qed
```

```
end

context iso_instantiation_sig
begindefinition ‹Sortals ≡ { U . U ∈ 𝒰_𝒮 ∧ Insts U ⊆ trim_non_permutables U}›lemma
SortalsI[intro!]:
  assumes ‹U ∈ 𝒰_𝒮› ‹⋀x w. x ::_w U ⟹ x ∈ trim_non_permutables U›
  shows ‹U ∈ Sortals›
  using assms by (auto simp: Sortals_def)

lemma SortalsE[elim!]:
  assumes ‹U ∈ Sortals›
  obtains ‹U ∈ 𝒰_𝒮› ‹⋀x w. x ::_w U ⟹ x ∈ trim_non_permutables U›
  using assms by (auto simp: Sortals_def)

lemma SortalsD:
  assumes ‹U ∈ Sortals›
  shows ‹U ∈ 𝒰_𝒮› ‹⋀x w. x ::_w U ⟹ x ∈ trim_non_permutables U›
  using assms by auto

lemma sortals_are_universals: ‹Sortals ⊆ 𝒰›
  by blast

lemma sortals_are_subst_universals: ‹Sortals ⊆ 𝒰_𝒮›
  by blast

end

context finite_instantiation
beginlemma  sortals_instances_are_identifiable:
  assumes ‹U ∈ Sortals› ‹x ∈ Insts U›
  shows ‹x ∈ 𝒫_=›
proof -
  obtain A: ‹U ∈ 𝒰_𝒮› ‹U ∈ 𝒰›  using assms(1) by blast
  then have B: ‹x ∈ 𝒮› using assms(2)  𝒰_𝒮_insts by blast
  show ?thesis
    apply (intro non_permutables_under_trimming_are_the_identifiable[of x U] A B)
    using assms(2) SortalsD[OF assms(1)] by blast
qed

end

end
```

APPENDIX F – Auxiliary Theories

```
theory Common
  imports Main "HOL-ZF.MainZF" "HOL-Library.LaTeXsugar"
begin


translations
  "xs" <= "CONST set xs"


translations
  "n" <= "CONST of_nat n"
  "n" <= "CONST int n"


syntax (latex output)
  "_appendL" :: "'a list ⇒ 'a list ⇒ 'a list" (infixl "@" 65)
translations
  "_appendL xs ys" <= "xs @ ys"
  "_appendL (_appendL xs ys) zs" <= "_appendL xs (_appendL ys zs)"



translations
  "_bind (p, CONST DUMMY) e" <= "_bind p (CONST fst e)"
  "_bind (CONST DUMMY, p) e" <= "_bind p (CONST snd e)"

  "_tuple_args x (_tuple_args y z)" ==
    "_tuple_args x (_tuple_arg (_tuple y z))"

  "_bind (CONST Some p) e" <= "_bind p (CONST the e)"
  "_bind (p # CONST DUMMY) e" <= "_bind p (CONST hd e)"
  "_bind (CONST DUMMY # p) e" <= "_bind p (CONST tl e)"


no_syntax (output)
  "_constrain" :: "logic => type => logic"  ("_::_" [4, 0] 3)
  "_constrain" :: "prop' => type => prop'"  ("_::_" [4, 0] 3)
```

syntax (output)

```
  "_constrain" :: "logic => type => logic"  ("_ :: _" [4, 0] 3)
  "_constrain" :: "prop' => type => prop'"  ("_ :: _" [4, 0] 3)
```

syntax (output)

```
  "_topsort" :: "sort"  ("⊤" 1000)
  "_sort" :: "classes => sort"  ("'(_')" 1000)
  "_classes" :: "id => classes => classes"  ("_ ∩ _" 1000)
  "_classes" :: "longid => classes => classes"  ("_ ∩ _" 1000)
```

setup ‹

```
  Thy_Output.antiquotation_pretty_source_embedded binding‹myterm_type›
    (Term_Style.parse -- Args.term)
    (fn ctx => fn (_,t) =>
        Pretty.block [Thy_Output.pretty_term ctx t, Pretty.str " ::",
            Pretty.brk 1, Syntax.pretty_typ ctx (fastype_of t)])
```

›

setup ‹

```
 let
   fun proc_loc_name (ctxt, (name, pos)) =
      Locale.check (Proof_Context.theory_of ctxt) (name, pos) ;
   fun output_locale {context = ctxt, source = src, argument = loc_name : string}
: Latex.text =
    let
      val thy = Proof_Context.theory_of ctxt ;
      val loc_hyp_spec = Locale.hyp_spec_of thy loc_name ;
      fun filter_assumption (Element.Assumes x) = let val ((b,(t,_)::_)::_) = x in
SOME (b,t) end
          | filter_assumption _                  = NONE
      val assumptions = List.mapPartial filter_assumption loc_hyp_spec
      val assms_latex = assumptions |>
            map (fn (b,prop) =>
                [
                  prop |> Thy_Output.pretty_term ctxt |> Thy_Output.pretty ctxt ,
                  Latex.string "(",
                  "" |> Attrib.pretty_binding ctxt b |> Pretty.block |> Thy_Output.pretty
```

```
ctxt,
                  Latex.string ")\n\n"
                ]
            ) |> List.concat
    in
      Latex.environment_block "locale" (
       [Latex.string "\\localename{", Latex.string loc_name, Latex.string "}\n"]
@
       [Latex.environment_block "localeassms" assms_latex]
      )

    end
 in
   Document_Antiquotation.setup_embedded binding⟨locale_full⟩
    (Args.context -- Scan.lift Args.embedded_position >> proc_loc_name)
    output_locale
  end

⟩

setup ⟨
  Document_Antiquotation.setup_embedded binding⟨thm_name⟩
      (Scan.lift Args.embedded_position -- Attrib.thms)
      (fn {argument = ((name,pos),thm), context = ctxt, source = src} =>
          let
             val ctxt1 = Config.put Name_Space.names_short true ctxt
          in
            Proof_Context.markup_extern_fact ctxt1 name |> Pretty.marks_str |> Thy_Output
ctxt1
          end)
⟩

bundle all_trace = [[simp_trace, linarith_trace, metis_trace, smt_trace]]

sledgehammer_params [timeout=60,provers= cvc4 vampire z3 e spass]

end
theory WellfoundedExtra
  imports Main
begin

lemma wfP_inj_wfP:
```

```
  fixes R :: ‹'a ⇒ 'a ⇒ bool› and f :: ‹'a ⇒ 'b›
  assumes ‹wfP R› ‹inj_on f { x . ∃y. R x y ∨ R y x }›
  shows ‹wfP (λx y. ∃x1 y1. R x1 y1 ∧ x = f x1 ∧ y = f y1)›
proof -
  define X where ‹X ≡ { x . ∃y. R x y ∨ R y x }›
  define X' where ‹X' ≡ f ` X›
  define P where ‹P ≡ λx y. ∃x1 y1. R x1 y1 ∧ x = f x1 ∧ y = f y1›
  define g where ‹g ≡ inv_into X f›
  have f_inj: ‹inj_on f X›
    using assms(2) by (auto simp: X_def)
  then have f1: ‹x = y› if ‹x ∈ X› ‹y ∈ X› ‹f x = f y›  for x y
    using that inj_onD by metis
  have g_inj: ‹inj_on g X'›
    apply (auto simp: g_def X'_def)
    using f_inj by (simp add: inj_on_def)
  then have g1: ‹x = y› if ‹x ∈ X'› ‹y ∈ X'› ‹g x = g y›  for x y
    using that inj_onD by metis
  have gf[simp]: ‹g (f x) = x› if ‹x ∈ X› for x
    using that f_inj by (auto simp: g_def)
  have fg[simp]: ‹f (g x) = x› if ‹x ∈ X'› for x
    using that f_inj X'_def by (auto simp: g_def)
  have fg_img[simp]: ‹f ` g ` Y = Y› if ‹Y ⊆ X'› for Y
    using that apply (auto intro!: that simp: image_def)
    subgoal premises P for x
      apply (intro exI[of _ ‹g x›] conjI bexI[of _ ‹x›])
      using P fg by auto
    done
  have P1: ‹P x y ⟹ x ∈ X' ∧ y ∈ X'› for x y
    by (auto simp: X_def X'_def P_def)
  have ‹∃z. z ∈ Q ∧ (∀y. R y z ⟶ y ∉ Q)› if as: ‹Q ≠ {}› for Q
  proof -
    have AA: ‹∃x. x ∈ Q› using as by blast
    note assms(1)[simplified wf_eq_minimal[to_pred],simplified,rule_format,OF AA]
    then show ‹∃z. z ∈ Q ∧ (∀y. R y z ⟶ y ∉ Q)›
      by blast
  qed
  then obtain pick where
        pick: ‹⋀Q. Q ≠ {} ⟹ pick Q ∈ Q›
              ‹⋀Q y. ⟦ Q ≠ {} ; R y (pick Q) ⟧ ⟹ y ∉ Q›
    by moura
  show ‹?thesis›
  proof (clarsimp simp add: wf_eq_minimal[to_pred])
```

```
fix Q and x :: ‹'b›
assume as: ‹x ∈ Q›
define Q' where Q1: ‹Q' ≡ g ` Q›
define z where ‹z ≡ if Q ∩ X' ≠ {} then f (pick (g ` (Q ∩ X'))) else x›
have ‹Q ∩ X' ⊆ X'› by blast
then have fgQX'[simp]: ‹f ` g ` (Q ∩ X') = Q ∩ X'› by simp
show ‹∃z∈Q. ∀y. (∃x1 y1. R x1 y1 ∧ y = f x1 ∧ z = f y1) ⟶ y ∉ Q›
proof (intro bexI[of _ ‹z›] allI impI ; (elim exE conjE)?)
  show ‹z ∈ Q›
  proof (cases ‹g ` (Q ∩ X') = {}›)
    assume as1: ‹g ` (Q ∩ X') = {}›
    then have ‹Q ∩ X' = {}› by auto
    then have ‹z = x› by (auto simp: z_def)
    then show ‹z ∈ Q› using as by simp
  next
    assume as1: ‹g ` (Q ∩ X') ≠ {}›
    then have ‹Q ∩ X' ≠ {}› by auto
    then have z: ‹z = f (pick (g ` (Q ∩ X')))› by (auto simp: z_def)
    have ‹pick (g ` (Q ∩ X')) ∈ g ` (Q ∩ X')›
      using pick[of ‹g ` (Q ∩ X')›] as1 by blast
    then have ‹f (pick (g ` (Q ∩ X'))) ∈ Q›
      by (auto simp: z_def)
    then show ‹z ∈ Q› using z by simp
  qed
  fix y x1 y1
  assume as1: ‹R x1 y1› ‹y = f x1› ‹z = f y1›
  then have x1y1: ‹x1 ∈ X› ‹y1 ∈ X› using X_def by auto
  then have yz: ‹y ∈ X'› ‹z ∈ X'› using X'_def as1 by auto
  show ‹y ∉ Q›
  proof
    assume as2: ‹y ∈ Q›
    then have QX': ‹Q ∩ X' ≠ {}› using yz by blast
    then have z: ‹z = f (pick (g ` (Q ∩ X')))› using z_def by auto
    have gQX'_ss: ‹g ` (Q ∩ X') ⊆ X› using X'_def by auto
    have ‹g ` (Q ∩ X') ≠ {}› using QX' by blast
    then obtain pick1: ‹pick (g ` (Q ∩ X')) ∈ g ` (Q ∩ X')›
        ‹⋀y. R y (pick (g ` (Q ∩ X'))) ⟹ y ∉ g ` (Q ∩ X')›
      using pick by metis
    then have ‹pick (g ` (Q ∩ X')) ∈ X› using gQX'_ss by blast
    then have ‹y1 = pick (g ` (Q ∩ X'))›
      using f1  x1y1(2) z as1(3) by metis
    then have pick2: ‹⋀y. R y y1 ⟹ y ∉ g ` (Q ∩ X')›
```

```
          using pick1(2) by auto
        then have ‹x₁ ∉ g ` (Q ∩ X')› using as1(1) by metis
        then show ‹False›
          by (metis IntI as1(2) as2 gf imageI x1y1(1) yz(1))
      qed
    qed
  qed
qed

end
```