A First-Order Logic Formalization of the Unified Foundational Ontology

Daniele Porelo, Claudenir M. Fonseca, João Paulo A. Almeida, Giancarlo Guizzardi, Tiago Prince Sales

December 1, 2021

Abstract

This document presents a formalization of the Unified Foundational Ontology (UFO) in first-order logic. This formalization is documented by means of three complementary representations: (i) a representation in standard Common Logic using the CLIF syntax; (ii) a representation in natural language; and, when applicable, (iii) a UML-based diagrammatic representation. The presented formalization is supported by consistency and satisfiability checks performed through automated proofing tools.

1 Introduction

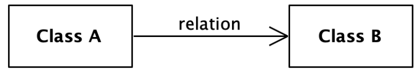
This document presents a formalization of the Unified Foundational Ontology (UFO) in first-order logic. This formalization is documented by means of three complementary representations: (i) a representation in standard Common Logic using the CLIF syntax; (ii) a representation in natural language; and, when applicable, (iii) a UML-based diagrammatic representation. The presented formalization is supported by consistency and satisfiability checks performed through automated proofing tools.

The remainder of this document is organized as a single formalization section (Section 2), which contains subsections for each submodule of the ontology.

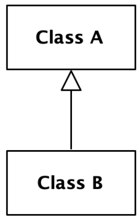
2 Formalization

This section contains the formalization of the Unified Foundational Ontology (UFO) in first-order logics. This formalization is organized in several subsections where each presents the formalization of a portion of the whole ontology. The formalization is presented through different equivalent representations, designed to support the understanding of its contents: (i) a representation in standard Common Logic using the CLIF syntax; (ii) a representation in natural language; and, when applicable, (iii) a UML-based diagrammatic representation.

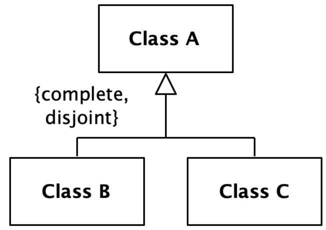
(a) Rectangle shape.



(b) Labeled open arrow.



(c) Closed arrow.



(d) Labeled closed arrows.

Figure 1: UML-based representation of first-order logic axioms.

The UML-based diagrammatic representation serves as a visual representation certain predicates and axioms, being each element in Figure 1 being translated as follows:

- Rectangle shape (Figure 1a): visual representation of unary predicates associated to types in the ontology; the associated predicate is shown in lower camel case with no spaces.

  $classA(x)$

- Open arrow (Figure 1b): visual representation of binary predicates; the predicate associated to the arrows' label is shown in lower camel case with no spaces; the predicate can only be true for any x and y if it is also true predicates associated to the types of each end (keeping the order of the arrow in the binary predicate's positions); this representation may also be associated to ternary predicates ifif its third position represents a time-index.

  $\forall x, y(relation(x, y) \rightarrow (classA(x) \wedge classB(y)))$

  $\forall x, y, w(relation(x, y, w) \rightarrow (classA(x) \wedge classB(y) \wedge world(w)))$

- Closed arrow (Figure 1c): visual representation of specializations between ontology's types, where the type in the tail of the arrow is a subtype of the type in the head of the arrow.

  $\forall x(classB(x) \rightarrow classA(x))$

- Labeled closed arrow (Figure 1d): visual representation of disjoint and/or complete constraints over sets specializations between ontology's types.

  $\forall x(classB(x) \rightarrow classA(x))$
  $\forall x(classC(x) \rightarrow classA(x))$
  $\forall x(classA(x) \rightarrow (classB(x) \vee classC(x)))$ {complete}
  $\neg\exists x(classB(x) \wedge classC(x))$ {disjoint}

2.1 Partial Taxonomy of UFO: Thing

This subsection presents most general types of UFO's taxonomy specializing the type Thing (Figure 2).

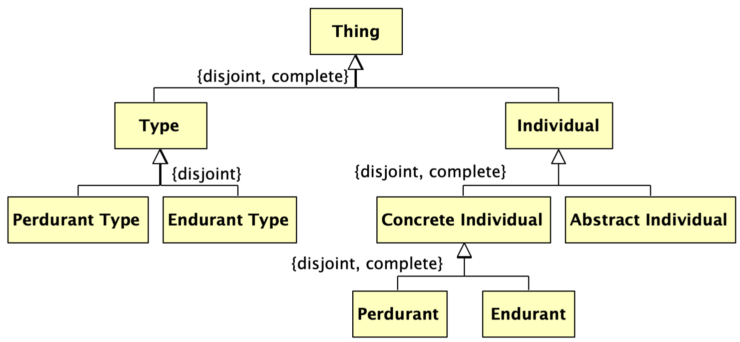


Figure 2: Visual representation of UFO's taxonomy of Thing.

a1 For every x, x is a Thing iff x is either a Type or an Individual.

$$\forall x(\mathsf{type\_}(x) \lor \mathsf{individual}(x) \leftrightarrow \mathsf{thing}(x))$$

```
(cl-text  ax_thing_taxonomy
(forall (x)
  (iff (or (type_ x) (individual x))
    (thing x))
  )
)
```

a2 There is no x such that it is a Type and an Individual.

$$\neg\exists x(\mathsf{type\_}(x) \land \mathsf{individual}(x))$$

```
(cl-text  ax_thing_partition
(not (exists (x)
  (and (type_ x) (individual x)))
)
)
```

a3 For every x, x is an Individual iff x is either a Concrete Individual or an Abstract Individual.

$$\forall x(\mathsf{concreteIndividual}(x) \lor \mathsf{abstractIndividual}(x) \leftrightarrow \mathsf{individual}(x))$$

```
(cl-text  ax_individual_taxonomy
(forall (x)
  (iff (or (concreteIndividual x) (abstractIndividual x))
    (individual x))
  )
)
```

a4 There is no x such that it is a Concrete Individual and an Abstract Individual.

$\neg\exists x(\mathsf{concreteIndividual}(x) \wedge \mathsf{abstractIndividual}(x))$

```
(cl-text  ax_individual_partition
(not (exists (x)
  (and (concreteIndividual x) (abstractIndividual x)))
)
)
```

a5 For every x, x is a Concrete Individual iff x is either a Perdurant or an Endurant.

$\forall x(\mathsf{endurant}(x) \vee \mathsf{perdurant}(x) \leftrightarrow \mathsf{concreteIndividual}(x))$

```
(cl-text  ax_concreteIndividual_taxonomy
(forall (x)
  (iff (or (endurant x) (perdurant x))
    (concreteIndividual x))
  )
)
```

a6 There is no x such that it is a Perdurant and an Endurant.

$\neg\exists x(\mathsf{endurant}(x) \wedge \mathsf{perdurant}(x))$

```
(cl-text  ax_concreteIndividual_partition
(not (exists (x)
  (and (endurant x) (perdurant x)))
)
)
```

a7 For every x, x is a Concrete Individual iff x is either a Perdurant or an Endurant.

$\forall x(\mathsf{endurantType}(x) \vee \mathsf{perdurantType}(x) \rightarrow \mathsf{type\_}(x))$

```
(cl-text  ax_type_taxonomy
(forall (x)
  (if (or (endurantType x) (perdurantType x))
    (type_ x))
  )
)
```

a8 There is no x such that it is a Perdurant Type and an Endurant Type.

$\neg\exists x(\mathsf{endurantType}(x) \wedge \mathsf{perdurantType}(x))$

```
(cl-text  ax_type_partition
(not (exists (x)
  (and (endurantType x) (perdurantType x)))
)
)
```

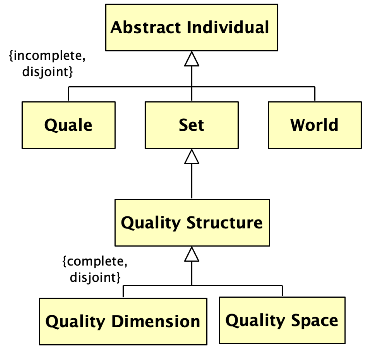


Figure 3: Visual representation of UFO's taxonomy of Abstract Individual.

2.2 Partial Taxonomy of UFO: Abstract Individual

This subsection presents a portion of UFO's taxonomy specializing the type Abstract Individual (Figure 3).

a9 Every x that is a Quale is also an Abstract Individual.

$\forall x(\mathsf{quale}(x) \rightarrow \mathsf{abstractIndividual}(x))$

```
(cl-text  ax_abstractIndividual_taxonomy_quale
(forall (x)
  (if (quale x)
    (abstractIndividual x))
  )
)
```

a10 Every x that is a Set is also an Abstract Individual.

$\forall x(\mathsf{set\_}(x) \rightarrow \mathsf{abstractIndividual}(x))$

```
(cl-text  ax_abstractIndividual_taxonomy_set
(forall (x)
  (if (set_ x)
    (abstractIndividual x))
  )
)
```

a11 Every x that is a World is also an Abstract Individual.

$\forall x(\mathsf{world}(x) \rightarrow \mathsf{abstractIndividual}(x))$

```
(cl-text  ax_abstractIndividual_taxonomy_world
(forall (x)
  (if (world x)
    (abstractIndividual x))
  )
)
```

a12 There is no x such that it is a Quale, a Set, and a World (pairwise disjoint).

$\neg\exists x((\mathsf{quale}(x) \land \mathsf{set\_}(x)) \lor (\mathsf{quale}(x) \land \mathsf{world}(x)) \lor (\mathsf{set\_}(x) \land \mathsf{world}(x)))$

```
(cl-text  ax_abstractIndividual_pairwiseDisjoint
(not (exists (x)
  (or (and (quale x) (set_ x)) (and (quale x) (world x)) (and
    (set_ x) (world x))))
)
)
```

a13 Every x that is a Quality Structure is also a Set.

$\forall x(\mathsf{qualityStructure}(x) \rightarrow \mathsf{set\_}(x))$

```
(cl-text  ax_set_taxonomy_qualityStructure
(forall (x)
  (if (qualityStructure x)
    (set_ x))
  )
)
```

a14 For every x, x is a Quality Structure iff x is either a Quality Dimension or a Quality Space.

$\forall x(\mathsf{qualityDimension}(x) \lor \mathsf{qualitySpace}(x) \leftrightarrow \mathsf{qualityStructure}(x))$

```
(cl-text  ax_qualityStructure_taxonomy
(forall (x)
  (iff (or (qualityDimension x) (qualitySpace x))
    (qualityStructure x))
  )
)
```

a15 There is no x such that it is a Quality Dimension and a Quality Space.

$\neg\exists x(\mathsf{qualityDimension}(x) \land \mathsf{qualitySpace}(x))$

```
(cl-text  ax_qualityStructure_partition
(not (exists (x)
  (and (qualityDimension x) (qualitySpace x)))
)
)
```

2.3 UFO Taxonomy

2.3.1 Partial Taxonomy of Abstract Individual

```
% Abstract Individual

fof(ax_abstractIndividual_taxonomy_quale, axiom, (
  ![X]: (quale(X) => (abstractIndividual(X)))
)).

fof(ax_abstractIndividual_taxonomy_set, axiom, (
  ![X]: (set_(X) => (abstractIndividual(X)))
```

```
)).

fof(ax_abstractIndividual_taxonomy_world, axiom, (
  ![X]: (world(X) => (abstractIndividual(X)))
)).

fof(ax_abstractIndividual_pairwiseDisjoint, axiom, (
  ~?[X]: ((quale(X) & set_(X)) | (quale(X) & world(X)) | (set_(X) &
     world(X)))
)).

% Set

fof(ax_set_taxonomy_qualityStructure, axiom, (
  ![X]: (qualityStructure(X) => (set_(X)))
)).

% Quality Structure

fof(ax_qualityStructure_taxonomy, axiom, (
  ![X]: ((qualityDimension(X) | qualitySpace(X)) <=> (
    qualityStructure(X)))
)).

fof(ax_qualityStructure_partition, axiom, (
  ~?[X]: (qualityDimension(X) & qualitySpace(X))
)).

% Abstract Individual partial taxonomy instances
% (tested to rule out trivial models)
```

2.3.2 Partial Taxonomy of Endurant

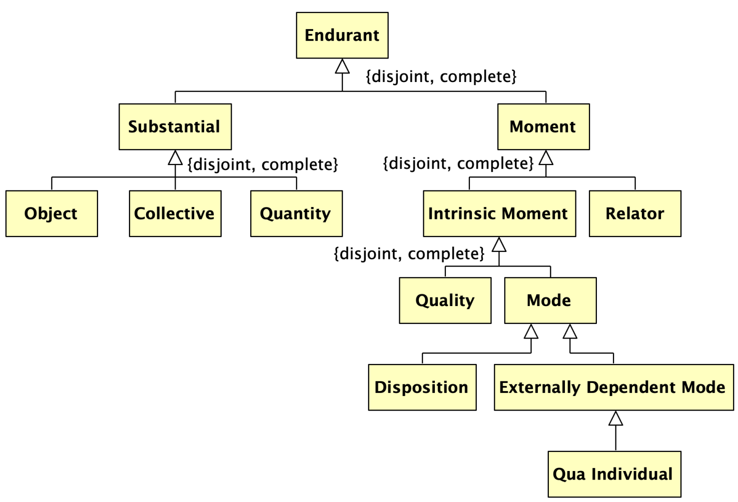


Figure 4: Partial Taxonomy of UFO – Endurant.

```
% fof(ax_abstractIndividual_instances, axiom, (
%   set_(set1) & quale(quale1) & qualityStructure(qualityStructure1
    ) & qualityDimension(qualityDimension1) & qualitySpace(
    qualitySpace1) & world(world1)
% )).

% Endurant

fof(ax_endurant_taxonomy, axiom, (
  ![X]: ((substantial(X) | moment(X)) <=> (endurant(X)))
)).

fof(ax_endurant_partition, axiom, (
  ~?[X]: (substantial(X) & moment(X))
)).

% Substantial

fof(ax_substantial_taxonomy, axiom, (
  ![X]: ((object(X) | collective(X) | quantity(X)) <=> (substantial
    (X)))
)).

fof(ax_substantial_partition, axiom, (
  ~?[X]: ((object(X) & collective(X)) | (object(X) & quantity(X)) |
     (collective(X) & quantity(X)))
)).

% Moment

fof(ax_moment_taxonomy, axiom, (
  ![X]: ((intrinsicMoment(X) | relator(X)) <=> (moment(X)))
)).

fof(ax_moment_partition, axiom, (
  ~?[X]: (intrinsicMoment(X) & relator(X))
)).

% Intrinsic Moment

fof(ax_intrinsicMoment_taxonomy, axiom, (
  ![X]: ((quality(X) | mode(X)) <=> (intrinsicMoment(X)))
)).

fof(ax_intrinsicMoment_partition, axiom, (
  ~?[X]: (quality(X) & mode(X))
)).

% Mode

fof(ax_mode_taxonomy_externallyDependentMode, axiom, (
  ![X]: (externallyDependentMode(X) => (mode(X)))
)).

% Externally Dependent Mode

fof(ax_externallyDependentMode_taxonomy_quaIndividual, axiom, (
```

```
  ![X]: (quaIndividual(X) => (externallyDependentMode(X)))
)).

% Endurant partial taxonomy instances
% (tested to rule out trivial models)
```

2.3.3 Partial Taxonomy of Endurant Type (on ontological natures)

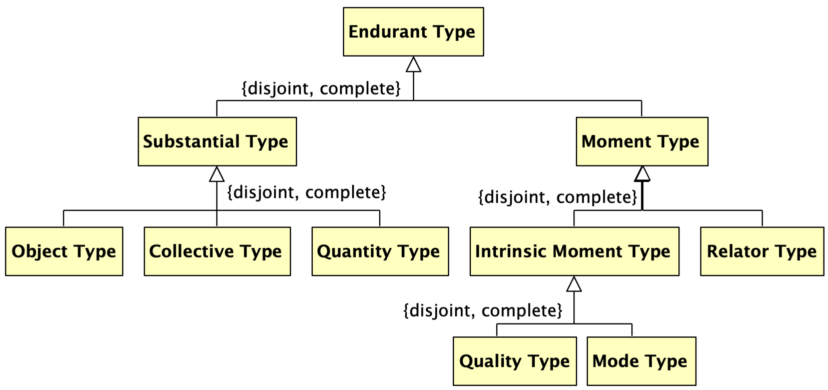


Figure 5: Partial Taxonomy of UFO – Endurant Types (by ontological nature).

```
% fof(ax_endurant_instances, axiom, (
%   substantial(substantial1) & moment(moment1) & object(object1) &
     collective(collective1) & quantity(quantity1) &
    intrinsicMoment(intrinsicMoment1) & relator(relator1) & quality
    (quality1) & mode(mode1) & disposition(disposition1) &
    externallyDependentMode(externallyDependentMode1) &
    quaIndividual(quaIndividual1)
% )).

% Endurant Type (by ontological nature)

fof(ax_endurantType_taxonomy_nature, axiom, (
  ![X]: ((substantialType(X) | momentType(X)) <=> (endurantType(X))
    )
)).

fof(ax_endurantType_partition_nature, axiom, (
  ~?[X]: (substantialType(X) & momentType(X))
)).

% Substantial Type

fof(ax_substantialType_taxonomy, axiom, (
  ![X]: ((objectType(X) | collectiveType(X) | quantityType(X)) <=>
    (substantialType(X)))
)).

fof(ax_substantialType_partition, axiom, (
```

```
  ~?[X]: ((objectType(X) & collectiveType(X)) | (objectType(X) &
    quantityType(X)) | (collectiveType(X) & quantityType(X)))
)).

% Moment Type

fof(ax_momentType_taxonomy, axiom, (
  ![X]: ((intrinsicMomentType(X) | relatorType(X)) <=> (momentType(
    X)))
)).

fof(ax_momentType_partition, axiom, (
  ~?[X]: (intrinsicMomentType(X) & relatorType(X))
)).

% Intrinsic Moment Type

fof(ax_intrinsicMomentType_taxonomy, axiom, (
  ![X]: ((qualityType(X) | modeType(X)) <=> (intrinsicMomentType(X)
    ))
)).

fof(ax_intrinsicMomentType_partition, axiom, (
  ~?[X]: (qualityType(X) & modeType(X))
)).

% Endurant Type (by ontological nature) partial taxonomy instances
% (tested to rule out trivial models)
```

2.3.4 Partial Taxonomy of Endurant Type (on modal properties of types)

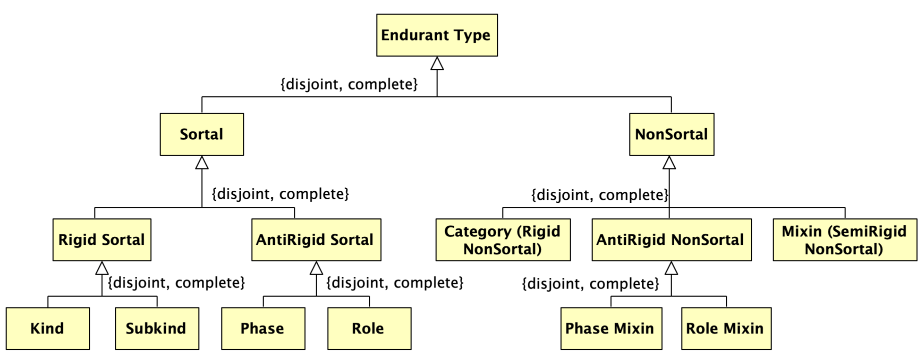


Figure 6: Partial Taxonomy of UFO – Endurant Types (by modal properties of types).

```
% fof(ax_endurantType_instances_natures, axiom, (
%   substantialType(substantialType1) & momentType(momentType1) &
    objectType(objectType1) & collectiveType(collectiveType1) &
    quantityType(quantityType1) & intrinsicMomentType(
```

```
    intrinsicMomentType1) & relatorType(relatorType1) & qualityType
    (qualityType1) & modeType(modeType1)
% )).

% Endurant Type (by modal properties of types)

fof(ax_endurantType_taxonomy_properties, axiom, (
  ![X]: ((sortal(X) | nonSortal(X)) <=> (endurantType(X)))
)).

fof(ax_endurantType_partition_properties, axiom, (
  ~?[X]: (sortal(X) & nonSortal(X))
)).

% Sortal

fof(ax_sortal_taxonomy, axiom, (
  ![X]: ((rigidSortal(X) | antiRigidSortal(X)) <=> (sortal(X)))
)).

fof(ax_sortal_partition, axiom, (
  ~?[X]: (rigidSortal(X) & antiRigidSortal(X))
)).

% Rigid Sortal

fof(ax_rigidSortal_taxonomy, axiom, (
  ![X]: ((kind(X) | subkind(X)) <=> (rigidSortal(X)))
)).

fof(ax_rigidSortal_partition, axiom, (
  ~?[X]: (kind(X) & subkind(X))
)).

% Anti-Rigid Sortal

fof(ax_antiRigidSortal_taxonomy, axiom, (
  ![X]: ((phase(X) | role(X)) <=> (antiRigidSortal(X)))
)).

fof(ax_antiRigidSortal_partition, axiom, (
  ~?[X]: (phase(X) & role(X))
)).

% Non-Sortal

fof(ax_nonSortal_taxonomy, axiom, (
  ![X]: ((rigidNonSortal(X) | semiRigidNonSortal(X) |
    antiRigidNonSortal(X)) <=> (nonSortal(X)))
)).

fof(ax_nonSortal_partition, axiom, (
  ~?[X]: ((rigidNonSortal(X) & semiRigidNonSortal(X)) | (
    rigidNonSortal(X) & antiRigidNonSortal(X)) | (
    semiRigidNonSortal(X) & antiRigidNonSortal(X)))
)).

```

```
% Category

fof(ax_rigidNonSortal_taxonomy, axiom, (
  ![X]: (rigidNonSortal(X) <=> (category(X)))
)).

% Mixin

fof(ax_semiRigidNonSortal_taxonomy, axiom, (
  ![X]: (semiRigidNonSortal(X) <=> (mixin(X)))
)).

% Anti-Rigid Non-Sortal

fof(ax_antiRigidNonSortal_taxonomy, axiom, (
  ![X]: ((phaseMixin(X) | roleMixin(X)) <=> (antiRigidNonSortal(X)))
)).

fof(ax_antiRigidNonSortal_partition, axiom, (
  ~?[X]: (phaseMixin(X) & roleMixin(X))
)).

% Endurant Type (by modal properties of types) partial taxonomy instances
% (tested to rule out trivial models)
```

2.3.5 Defining Types, Individuals, and Specialization

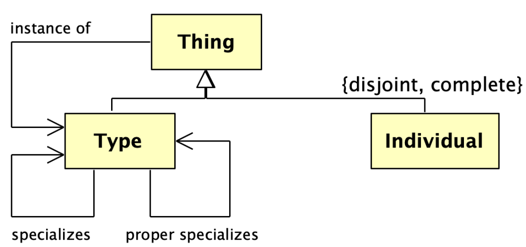


Figure 7: Types, individuals, instantiation, and specialization.

```
% fof(ax_endurantType_instances_properties, axiom, (
%   sortal(sortal1) & nonSortal(nonSortal1) & rigidSortal(rigidSortal1) & antiRigidSortal(antiRigidSortal1) & kind(kind1) & subkind(subkind1) & phase(phase1) & role(role1) & category(category1) & mixin(mixin1) & antiRigidNonSortal(antiRigidNonSortal1) & phaseMixin(phaseMixin1) & roleMixin(roleMixin1)
% )).

%%%%%%%%%%% Instance of, Types, and Individuals %%%%%%%%%%

fof(ax_dIof, axiom, (
  ![X,Y,W]: (iof(X,Y,W) => (type_(Y) & world(W)))
```

```
)).

fof(ax_dType_a1, axiom, (
  ![X]: (type_(X) <=> (?[Y,W]: iof(Y,X,W)))
)).

fof(ax_dIndividual_a2, axiom, (
  ![X]: (individual(X) <=> (~?[Y,W]: iof(Y,X,W)))
)).

fof(ax_multiLevel_a3, axiom, (
  ![X,Y,W]: (iof(X,Y,W) => (type_(X) | individual(X)))
)).

fof(ax_twoLevelConstrained_a4, axiom, (
  ~?[X,Y,Z,W]: (type_(X) & iof(X,Y,W) & iof(Y,Z,W))
)).

% Instantiation relations
% (tested to rule out trivial models)

% fof(ax_iofInUse, axiom, (
%   type_(t2) & individual(i2) & world(w2) & iof(i2,t2,w2)
% )).

% Ax |= "th_everythingIsAThing_t1"; conjecture commented for convenience

% fof(th_everythingIsAThing_t1, conjecture, (
%   ![X]: (type_(X) | individual(X))
% )).

% Ax |= "th_thingPartition_t2"; conjecture commented for convenience

% fof(th_thingPartition_t2, conjecture, (
%   ~?[X]: (type_(X) & individual(X))
% )).

%%%%%%%% Specialization and Proper Specialization %%%%%%%%

fof(ax_dSpecializes, axiom, (
  ![X,Y]: (specializes(X,Y) => (type_(X) & type_(Y)))
)).

fof(ax_specialization_a5, axiom, (
  ![T1,T2]: (specializes(T1,T2) <=> (
    type_(T1) & type_(T2) & ![W]: (world(W) => ![E]: (iof(E,T1,W) => iof(E,T2,W)))
  ))
)).

fof(ax_properSpecializes_d1, axiom, (
  ![X,Y]: (properSpecializes(X,Y) <=> (specializes(X,Y) & ~specializes(Y,X)))
)).

```

```
% Ax |= "th_cyclicSpecializations_t3"; conjecture commented for convenience

% fof(th_cyclicSpecializations_t3, conjecture, (
%   ![X,Y]: (specializes(X,Y) => (specializes(X,X) & specializes(Y,Y)))
% )).

% Ax |= "th_transitiveSpecializations_t4"; conjecture commented for convenience

% fof(th_transitiveSpecializations_t4, conjecture, (
%   ![X,Y,Z]: ((specializes(X,Y) & specializes(Y,Z)) => (specializes(X,Z)))
% )).

fof(ax_sharedSpecializations_a6, axiom, (
  ![T1,T2]: (![X,W]: ((iof(X,T1,W) & iof(X,T2,W) & ~specializes(T1,T2) & ~specializes(T2,T1)) => (
      (?[T3]: (specializes(T1,T3) & specializes(T2,T3) & iof(X,T3,W)))
      | (?[T3]: (specializes(T3,T1) & specializes(T3,T2) & iof(X,T3,W)))
  )))
)).

% Specialization relations
% (tested to rule out trivial models)

% fof(ax_specializesInUse, axiom, (
```

2.3.6 Defining Rigidity and Sortality

```
%%%%%%%%%%%%%%%% Sortality and Rigidity %%%%%%%%%%%%%%%%

% Rigidity

fof(ax_dRigid_a18, axiom, (
  ![T]: (rigid(T) <=> (endurantType(T) & (
    ![X]: ((?[W1]: (world(W1) & iof(X,T,W1))) => (![W2]: (world(W2) => iof(X,T,W2))))
  )))
)).

fof(ax_dAntiRigid_a19, axiom, (
  ![T]: (antiRigid(T) <=> (endurantType(T) & (
    ![X]: ((?[W1]: (world(W1) & iof(X,T,W1))) => (?[W2]: (world(W2) & ~iof(X,T,W2)))
  ))))
)).

fof(ax_dSemiRigid_a20, axiom, (
  ![T]: (semiRigid(T) <=> (endurantType(T) & ~rigid(T) & ~antiRigid(T)))
)).

```

```
% Ax |= "th_thEndurantTypeHaveRigidity_t5"; conjecture commented for convenience

% fof(th_thEndurantTypeHaveRigidity_t5, conjecture, (
%   ![T]: (endurantType(T) <=> (rigid(T) | semiRigid(T) | antiRigid(T)))
% )).

% Ax |= "th_pairwiseDisjointRigidities_t6"; conjecture commented for convenience

% fof(th_pairwiseDisjointRigidities_t6, conjecture, (
%   ~![T]: ((rigid(T) & semiRigid(T)) | (semiRigid(T) & antiRigid(T)) | (rigid(T) & antiRigid(T)))
% )).

% Ax |= "th_rigidAntiRigidSpecializationConstraint_t7"; conjecture commented for convenience

% fof(th_rigidAntiRigidSpecializationConstraint_t7, conjecture, (
%   ~![T1,T2]: (rigid(T1) & antiRigid(T2) & specializes(T1,T2))
% )).

% Ax |= "th_semiRigidAntiRigidSpecializationConstraint_t8"; conjecture commented for convenience

% fof(th_semiRigidAntiRigidSpecializationConstraint_t8, conjecture, (
%   ~![T1,T2]: (semiRigid(T1) & antiRigid(T2) & specializes(T1,T2))
% )).

% Rigidity properties
% (tested to rule out trivial models)

% fof(ax_rigidityInUse, axiom, (
%   endurantType(t4_1) & endurantType(t4_2) & endurantType(t4_3) & rigid(t4_1) & semiRigid(t4_2) & antiRigid(t4_3) & properSpecializes(t4_1,t4_2) & properSpecializes(t4_3,t4_1)
% )).

% Sortality

fof(ax_endurantsKind_a21, axiom, (
  ![E]: (endurant(E) => (
    ?[U]: (kind(U) & (![W]: (world(W) => iof(E,U,W))))
  ))
)).

fof(ax_uniqueKind_a22, axiom, (
  ![E,U,W]: ((world(W) & kind(U) & iof(E,U,W)) => (
    ~?[U2,W2]: (kind(U2) & iof(E,U2,W2) & ~(U = U2))
  ))
)).

% Changing "ax_dSortal_a23" from the form it was defined in the paper to "sortals are endurant types that specialize some ultimate sortal" seem to express the same concept while
```

```
    speeding up the execution of SPASS considerably

% fof(ax_dSortal_a23, axiom, (
%   ![S]: (sortal(S) <=> (endurantType(S) & (?[U]: (kind(U) & (![E,W]: (iof(E,S,W) => iof(E,U,W)))))))
% )).

fof(ax_dSortal_a23, axiom, (
  ![S]: ((sortal(S)) <=> (endurantType(S) & (?[U]: (kind(U) & specializes(S,U)))))
)).

% If we have the taxonomy's axiomatization, then a24 becomes a theorem
% Ax |= "th_nonSortalsAreEndurantsThatAreNotSortals_a24"; conjecture commented for convenience

% fof(th_nonSortalsAreEndurantsThatAreNotSortals_a24, conjecture, (
%   ![NS]: ((nonSortal(NS)) <=> (endurantType(NS) & ~sortal(NS)))
% )).

% Ax |= "th_kindsAreRigid_t9"; conjecture commented for convenience

% fof(th_kindsAreRigid_t9, conjecture, (
%   ![U]: ((kind(U)) => (rigid(U)))
% )).

% Ax |= "th_kindsHaveDisjointExtensions_t10"; conjecture commented for convenience

% fof(th_kindsHaveDisjointExtensions_t10, conjecture, (
%   ![K1,K2]: ((kind(K1) & kind(K2) & ~(K1=K2)) => (
%     ~?[X,W1,W2]: (world(W1) & world(W2) & iof(X,K1,W1) & iof(X,K2,W2)))
%   )
% )).

% Ax |= "th_kindsHaveDisjointTaxonomies_t11"; conjecture commented for convenience

% fof(th_kindsHaveDisjointTaxonomies_t11, conjecture, (
%   ![K1,K2]: ((kind(K1) & kind(K2) & ~(K1=K2)) => (
%     ~?[T]: (specializes(T,K1) & specializes(T,K2)))
%   )
% )).

% Ax |= "th_kindsAreSortal_t12"; conjecture commented for convenience

% fof(th_kindsAreSortal_t12, conjecture, (
%   ![K]: ((kind(K)) => (sortal(K)))
% )).

% Ax |= "th_sortalSpecializeKinds_t13"; conjecture commented for convenience

% fof(th_sortalSpecializeKinds_t13, conjecture, (
```

```
%   ![S]: ((sortal(S)) => (?[K]: (kind(K) & specializes(S,K))))
% )).

% Ax |= "th_sortalsSpecializeAUniqueKind_t14"; conjecture commented for convenience

% fof(th_sortalsSpecializeAUniqueKind_t14, conjecture, (
%   ![S]: ((sortal(S)) => (~?[U,U2]: (kind(U) & kind(U2) & specializes(S,U) & specializes(S,U2) & ~(U=U2))))
% )).

% Sortality properties
% (tested to rule out trivial models)

% fof(ax_sortalityInUse, axiom, (
%   endurant(e5_1) & endurant(e5_2) & world(w5) & kind(k5_1) & kind(k5_2) & iof(e5_1,k5_1,w5) & iof(e5_1,k5_1,w5) & ~(k5_1=k5_2)
% )).

% Sortality + Rigidity

fof(ax_rigidSortalsAreRigidAndSortal_xx, axiom, (
  ![T]: ((rigidSortal(T)) <=> (rigid(T) & sortal(T)))
)).

fof(ax_antiRigidSortalsAreAntiRigidAndSortal_xx, axiom, (
  ![T]: ((antiRigidSortal(T)) <=> (antiRigid(T) & sortal(T)))
)).

fof(ax_rigidNonSortalsAreRigidAndNonSortal_xx, axiom, (
  ![T]: ((rigidNonSortal(T)) <=> (rigid(T) & nonSortal(T)))
)).

fof(ax_antiRigidNonSortalsAreAntiRigidAndNonSortal_xx, axiom, (
  ![T]: ((antiRigidNonSortal(T)) <=> (antiRigid(T) & nonSortal(T)))
)).

fof(ax_semiRigidNonSortalsAreSemiRigidAndNonSortal_xx, axiom, (
  ![T]: ((semiRigidNonSortal(T)) <=> (semiRigid(T) & nonSortal(T)))
)).

% If we have the taxonomy's axiomatization, then a25 becomes a theorem
% Ax |= "th_kindAndSubkindAreDisjoint_a25"; conjecture commented for convenience

% fof(th_kindAndSubkindAreDisjoint_a25, conjecture, (
%   ~?[T]: (kind(T) & subkind(T))
% )).

% If we have the taxonomy's axiomatization, then a26 becomes a theorem
% Ax |= "th_kindAndSubkindAreRigidSortals_a26"; conjecture commented for convenience

% fof(th_kindAndSubkindAreRigidSortals_a26, conjecture, (
%   ![T]: ((kind(T) | subkind(T)) <=> (rigid(T) & sortal(T)))
```

```
% )).

% If we have the taxonomy's axiomatization, then a27 becomes a theorem
% Ax |= "th_phaseAndRoleAreDisjoint_a27"; conjecture commented for convenience

% fof(th_phaseAndRoleAreDisjoint_a27, conjecture, (
%   ~?[T]: (phase(T) & role(T))
% )).

% If we have the taxonomy's axiomatization, then a28 becomes a theorem
% Ax |= "th_phaseAndRoleAreAntiRigidSortals_a28"; conjecture commented for convenience

% fof(th_phaseAndRoleAreAntiRigidSortals_a28, conjecture, (
%   ![T]: ((phase(T) | role(T)) <=> (antiRigid(T) & sortal(T)))
% )).

% Skipping (a29) because we leave the concept of semi-rigid sortals out of this ontology.

% If we have the taxonomy's axiomatization, then a30 becomes a theorem
% Ax |= "th_categoriesAreRigidNonSortals_a30"; conjecture commented for convenience

% fof(th_categoriesAreRigidNonSortals_a30, conjecture, (
%   ![T]: ((category(T)) <=> (rigid(T) & nonSortal(T)))
% )).

% If we have the taxonomy's axiomatization, then a31 becomes a theorem
% Ax |= "th_mixinsAreSemiRigidNonSortals_a31"; conjecture commented for convenience

% fof(th_mixinsAreSemiRigidNonSortals_a31, conjecture, (
%   ![T]: ((mixin(T)) <=> (semiRigid(T) & nonSortal(T)))
% )).

% If we have the taxonomy's axiomatization, then a32 becomes a theorem
% Ax |= "th_phaseMixinAndRoleMixinAreDisjoint_a32"; conjecture commented for convenience

% fof(th_phaseMixinAndRoleMixinAreDisjoint_a32, conjecture, (
%   ~?[T]: (phaseMixin(T) & roleMixin(T))
% )).

% If we have the taxonomy's axiomatization, then a33 becomes a theorem
% Ax |= "ax_phaseMixinAndRoleMixinAreAntiRigidSortals_a33"; conjecture commented for convenience

% fof(th_phaseMixinAndRoleMixinAreAntiRigidSortals_a33, conjecture, (
```

```
%   ![T]: ((phaseMixin(T) | roleMixin(T)) <=> (antiRigid(T) &
    nonSortal(T)))
% )).

% Ax |= "th_leafCategoriesArePairwiseDisjoint_t18"; conjecture
    commented for convenience

% fof(th_leafCategoriesArePairwiseDisjoint_t18, conjecture, (
%   ~?[T]: (endurantType(T) & (
%     (
%       (kind(T) & subkind(T))
%       | (kind(T) & phase(T))
%       | (kind(T) & role(T))
%       | (kind(T) & category(T))
%       | (kind(T) & mixin(T))
%       | (kind(T) & phaseMixin(T))
%       | (kind(T) & roleMixin(T))
%     ) | (
%       (subkind(T) & phase(T))
%       | (subkind(T) & role(T))
%       | (subkind(T) & category(T))
%       | (subkind(T) & mixin(T))
%       | (subkind(T) & phaseMixin(T))
%       | (subkind(T) & roleMixin(T))
%     ) | (
%       (phase(T) & role(T))
%       | (phase(T) & category(T))
%       | (phase(T) & mixin(T))
%       | (phase(T) & phaseMixin(T))
%       | (phase(T) & roleMixin(T))
%     ) | (
%       (role(T) & category(T))
%       | (role(T) & mixin(T))
%       | (role(T) & phaseMixin(T))
%       | (role(T) & roleMixin(T))
%     ) | (
%       (category(T) & mixin(T))
%       | (category(T) & phaseMixin(T))
%       | (category(T) & roleMixin(T))
%     ) | (
%       (mixin(T) & phaseMixin(T))
%       | (mixin(T) & roleMixin(T))
%     ) | (
%       (phaseMixin(T) & roleMixin(T))
%     )
%   ))
% )).

% Ax |= "th_leafCategoriesCompletelyCategorizeAllEndurantTypes_t19
    "; conjecture commented for convenience

% fof(th_leafCategoriesCompletelyCategorizeAllEndurantTypes_t19,
    conjecture, (
%   ![T]: (endurantType(T) => (
%     kind(T) | subkind(T) | phase(T) | role(T) | category(T) |
    mixin(T) | phaseMixin(T) | roleMixin(T)
%   ))
```

```
% )).

% Sortality and rigidity properties combined
% (tested to rule out trivial models)

% fof(ax_sortalityAndRigidityInUse, axiom, (
%   endurant(e6_1) & endurant(e6_2) & world(w6) & kind(k6_1) & kind
    (k6_2) & iof(e6_1,k6_1,w6) & iof(e6_1,k6_1,w6) & ~(k6_1=k6_2)
% )).
```

2.3.7 Defining Endurant Types

```

% Defining the taxonomy of types of ontological natures through the
     categorization of the taxonomy of concrete individuals

fof(ax_perdurantTypeDefinition_a44, axiom, (
  ![T]: (perdurantType(T) <=> (
    type_(T) & (![P,W]: ((world(W) & iof(P,T,W)) => (perdurant(P)))
    )
  ))
)).

fof(ax_endurantTypeDefinition_a44, axiom, (
  ![T]: (endurantType(T) <=> (
    type_(T) & (![E,W]: ((world(W) & iof(E,T,W)) => (endurant(E))))
  ))
)).

fof(ax_substantialTypeDefinition_a44, axiom, (
  ![T]: (substantialType(T) <=> (
    type_(T) & (![E,W]: ((world(W) & iof(E,T,W)) => (substantial(E)
    )))
  ))
)).

fof(ax_momentTypeDefinition_a44, axiom, (
  ![T]: (momentType(T) <=> (
    type_(T) & (![E,W]: ((world(W) & iof(E,T,W)) => (moment(E))))
  ))
)).

fof(ax_objectTypeDefinition_a44, axiom, (
  ![T]: (objectType(T) <=> (
    type_(T) & (![E,W]: ((world(W) & iof(E,T,W)) => (object(E))))
  ))
)).

fof(ax_collectiveTypeDefinition_a44, axiom, (
  ![T]: (collectiveType(T) <=> (
    type_(T) & (![E,W]: ((world(W) & iof(E,T,W)) => (collective(E))
    ))
  ))
)).

fof(ax_quantityTypeDefinition_a44, axiom, (
  ![T]: (quantityType(T) <=> (
    type_(T) & (![E,W]: ((world(W) & iof(E,T,W)) => (quantity(E))))
```

```
  ))
)).

fof(ax_intrinsicMomentTypeDefinition_a44, axiom, (
  ![T]: (intrinsicMomentType(T) <=> (
    type_(T) & (![E,W]: ((world(W) & iof(E,T,W)) => (intrinsicMoment(E))))
  ))
)).

fof(ax_relatorTypeDefinition_a44, axiom, (
  ![T]: (relatorType(T) <=> (
    type_(T) & (![E,W]: ((world(W) & iof(E,T,W)) => (relator(E))))
  ))
)).

fof(ax_qualityTypeDefinition_a44, axiom, (
  ![T]: (qualityType(T) <=> (
    type_(T) & (![E,W]: ((world(W) & iof(E,T,W)) => (quality(E))))
  ))
)).

fof(ax_modeTypeDefinition_a44, axiom, (
  ![T]: (modeType(T) <=> (
    type_(T) & (![E,W]: ((world(W) & iof(E,T,W)) => (mode(E))))
  ))
)).

% Types Definition
% (tested to rule out trivial models)
% TODO: investigate why we cannot list four different endurant types (it may have something to do with "intrinsicMoment" and "intrinsicMomentType")

% fof(ax_typesDefinitionsInstances, axiom, (
%   objectType(ot7) & collectiveType(ct7) & modeType(mt7)
% )).

% Ax |= "th_leafCategoriesArePairwiseDisjoint_t21"; conjecture commented for convenience
% Having the previously defined taxonomy, this should be quite trivial

% fof(th_leafCategoriesArePairwiseDisjoint_t21, conjecture, (
%   ~?[T]: (type_(T) & (
%     (
%       (objectType(T) & collectiveType(T)) | (objectType(T) & quantityType(T)) | (objectType(T) & modeType(T)) | (objectType(T) & qualityType(T)) | (objectType(T) & relatorType(T)) | (objectType(T) & perdurantType(T))
%     ) | (
%       (collectiveType(T) & quantityType(T)) | (collectiveType(T) & modeType(T)) | (collectiveType(T) & qualityType(T)) | (collectiveType(T) & relatorType(T)) | (collectiveType(T) & perdurantType(T))
%     ) | (
%       (quantityType(T) & modeType(T)) | (quantityType(T) &
```

```
    qualityType(T)) | (quantityType(T) & relatorType(T)) | (
    quantityType(T) & perdurantType(T))
%     ) | (
%       (modeType(T) & qualityType(T)) | (modeType(T) & relatorType
    (T)) | (modeType(T) & perdurantType(T))
%     ) | (
%       (qualityType(T) & relatorType(T)) | (qualityType(T) &
    perdurantType(T))
%     ) | (
%       relatorType(T) & perdurantType(T)
%     )
%   ))
% )).

% Ultimate Sortals Definitions (by ontological nature)

fof(ax_objectKindDefinition_a45, axiom, (
  ![T]: (objectKind(T) <=> (objectType(T) & kind(T)))
)).

fof(ax_collectiveKindDefinition_a45, axiom, (
  ![T]: (collectiveKind(T) <=> (collectiveType(T) & kind(T)))
)).

fof(ax_quantityKindDefinition_a45, axiom, (
  ![T]: (quantityKind(T) <=> (quantityType(T) & kind(T)))
)).

fof(ax_modeKindDefinition_a45, axiom, (
  ![T]: (modeKind(T) <=> (modeType(T) & kind(T)))
)).

fof(ax_qualityKindDefinition_a45, axiom, (
  ![T]: (qualityKind(T) <=> (qualityType(T) & kind(T)))
)).

fof(ax_relatorKindDefinition_a45, axiom, (
  ![T]: (relatorKind(T) <=> (relatorType(T) & kind(T)))
)).

% Ultimate sortals (by ontological nature) instances
% (tested to rule out trivial models)
% TODO: investigate why we cannot list all different types of
    ultimate sortals at once

% fof(ax_typesDefinitionsInstances, axiom, (
%   objectKind(ok9) & collectiveKind(ck9) & quantityKind(quank9) &
    relatorKind(rk9) & modeKind(mk9) & qualityKind(qualk9)
% )).

% Skipping (t22) because (a21) makes it trivial

% Ax |= "th_endurantsInstantiateEndurantKindsOfSomeNature_a46";
    conjecture commented for convenience
% This axiom is actually a theorem in this version of the
    axiomatization

```

```
% fof(th_endurantsInstantiateEndurantKindsOfSomeNature_a46, conjecture, (
%   ![E]: (endurant(E) => (
%     ?[K,W]: ((objectKind(K) | collectiveKind(K) | quantityKind(K) | modeKind(K) | qualityKind(K) | relatorKind(K))
%     & iof(E,K,W))
%   ))
% )).

% Ax |= "th_endurantSortalsCompleteness_t23"; conjecture commented for convenience
% Thanks to the taxonomy, we already have "sortal(T) => endurantType(T)", but I leave it like this to be consistent with the paper

% fof(th_endurantSortalsCompleteness_t23, conjecture, (
%   ![T]: ((endurantType(T) & sortal(T)) => (objectKind(T) | collectiveKind(T) | quantityKind(T) | qualityKind(T) | modeKind(T) | relatorKind(T) | phase(T) | role(T)))
% )).

% Ax |= "th_objectTypesSpecializeAKindOfSameNature_t24"; conjecture commented for convenience

% fof(th_objectTypesSpecializeAKindOfSameNature_t24, conjecture, (
%   ![T]: ((objectType(T) & sortal(T)) <=> (?[K]: (objectKind(K) & specializes(T,K))))
% )).

% Ax |= "th_collectiveTypesSpecializeAKindOfSameNature_t24"; conjecture commented for convenience

% fof(th_collectiveTypesSpecializeAKindOfSameNature_t24, conjecture, (
%   ![T]: ((collectiveType(T) & sortal(T)) <=> (?[K]: (collectiveKind(K) & specializes(T,K))))
% )).

% Ax |= "th_quantityTypesSpecializeAKindOfSameNature_t24"; conjecture commented for convenience

% fof(th_quantityTypesSpecializeAKindOfSameNature_t24, conjecture, (
%   ![T]: ((quantityType(T) & sortal(T)) <=> (?[K]: (quantityKind(K) & specializes(T,K))))
% )).

% Ax |= "th_modeTypesSpecializeAKindOfSameNature_t24"; conjecture commented for convenience

% fof(th_modeTypesSpecializeAKindOfSameNature_t24, conjecture, (
%   ![T]: ((modeType(T) & sortal(T)) <=> (?[K]: (modeKind(K) & specializes(T,K))))
% )).

% Ax |= "th_qualityTypesSpecializeAKindOfSameNature_t24"; conjecture commented for convenience
```

```
% fof(th_qualityTypesSpecializeAKindOfSameNature_t24, conjecture, (
%   ![T]: ((qualityType(T) & sortal(T)) <=> (?[K]: (qualityKind(K) & specializes(T,K))))
% )).

% Ax |= "th_relatorTypesSpecializeAKindOfSameNature_t24"; conjecture commented for convenience

% fof(th_relatorTypesSpecializeAKindOfSameNature_t24, conjecture, (
%   ![T]: ((relatorType(T) & sortal(T)) <=> (?[K]: (relatorKind(K) & specializes(T,K))))
% )).

% Ax |= "th_sortalLeafCategoriesAreDisjoint_t25"; conjecture commented for convenience

% fof(th_sortalLeafCategoriesAreDisjoint_t25, conjecture, (
%   ![T]: (objectKind(T) => (~(collectiveKind(T) | quantityKind(T) | modeKind(T) | qualityKind(T) | relatorKind(T) | category(T) | mixin(T) | phaseMixin(T) | roleMixin(T))))
%   & ![T]: (collectiveKind(T) => (~(objectKind(T) | quantityKind(T) | modeKind(T) | qualityKind(T) | relatorKind(T) | category(T) | mixin(T) | phaseMixin(T) | roleMixin(T))))
%   & ![T]: (quantityKind(T) => (~(objectKind(T) | collectiveKind(T) | modeKind(T) | qualityKind(T) | relatorKind(T) | category(T) | mixin(T) | phaseMixin(T) | roleMixin(T))))
%   & ![T]: (modeKind(T) => (~(objectKind(T) | quantityKind(T) | collectiveKind(T) | qualityKind(T) | relatorKind(T) | category(T) | mixin(T) | phaseMixin(T) | roleMixin(T))))
%   & ![T]: (qualityKind(T) => (~(objectKind(T) | quantityKind(T) | modeKind(T) | collectiveKind(T) | relatorKind(T) | category(T) | mixin(T) | phaseMixin(T) | roleMixin(T))))
%   & ![T]: (relatorKind(T) => (~(objectKind(T) | quantityKind(T) | modeKind(T) | qualityKind(T) | collectiveKind(T) | category(T) | mixin(T) | phaseMixin(T) | roleMixin(T))))
%   & ![T]: (category(T) => (~(objectKind(T) | quantityKind(T) | modeKind(T) | qualityKind(T) | relatorKind(T) | collectiveKind(T) | mixin(T) | phaseMixin(T) | roleMixin(T))))
%   & ![T]: (mixin(T) => (~(objectKind(T) | quantityKind(T) | modeKind(T) | qualityKind(T) | relatorKind(T) | category(T) | collectiveKind(T) | phaseMixin(T) | roleMixin(T))))
%   & ![T]: (phaseMixin(T) => (~(objectKind(T) | quantityKind(T) | modeKind(T) | qualityKind(T) | relatorKind(T) | category(T) | mixin(T) | collectiveKind(T) | roleMixin(T))))
%   & ![T]: (roleMixin(T) => (~(objectKind(T) | quantityKind(T) | modeKind(T) | qualityKind(T) | relatorKind(T) | category(T) | mixin(T) | phaseMixin(T) | collectiveKind(T))))
% )).

% Ax |= "th_sortalLeafCategoriesAreComplete_t26"; conjecture commented for convenience

% fof(th_sortalLeafCategoriesAreComplete_t26, conjecture, (
%   ![T]: ((endurantType(T)) => (objectKind(T) | collectiveKind(T) | quantityKind(T) | qualityKind(T) | modeKind(T) | relatorKind(T) | phase(T) | role(T) | category(T) | mixin(T) | phaseMixin(T
```

```
    ) | roleMixin(T)))
% )).
```

2.3.8 Mereology

```

% TODO: review whether it is necessary to reduce mereology to
    concrete individuals; I am leaving this axiom out for the
    moment

% fof(ax_partArguments, axiom, (
%   ![X,Y]: (part(X,Y) => (concreteIndividual(X) &
    concreteIndividual(Y)))
% )).

fof(ax_reflexiveParthood, axiom, (
  ![X]: (partOf(X,X))
)).

fof(ax_antiSymmetricParthood_a47, axiom, (
  ![X,Y]: ((partOf(X,Y) & partOf(Y,X)) => (X=Y))
)).

fof(ax_antiSymmetricParthood_a48, axiom, (
  ![X,Y]: ((partOf(X,Y) & partOf(Y,X)) => (X=Y))
)).

fof(ax_transitiveParthood_a49, axiom, (
  ![X,Y,Z]: ((partOf(X,Y) & partOf(Y,Z)) => (partOf(X,Z)))
)).

fof(ax_overlappingWholes_a50, axiom, (
  ![X,Y]: ((overlap(X,Y)) <=> (?[Z]: (partOf(Z,X) & partOf(Z,Y))))
)).

fof(ax_strongSupplementation_a51, axiom, (
  ![X,Y]: (~partOf(X,Y) <=> ?[Z]: (partOf(Z,X) & ~overlap(Z,Y)))
)).

fof(ax_properPart_a52, axiom, (
  ![X,Y]: (properPartOf(X,Y) <=> (partOf(X,Y) & ~partOf(Y,X)))
)).

fof(ax_binarySum_a53, axiom, (
  ![X,Y,Z]: (sum(Z,X,Y) <=> ![W]: (overlap(W,Z) <=> (overlap(W,X) |
     overlap(W,Y))))
)).

% Mereology in use
% (tested to rule out trivial models)

% fof(ax_mereologyInUse, axiom, (
%   concreteIndividual(ci10_1) & concreteIndividual(ci10_2) &
    concreteIndividual(ci10_3) & concreteIndividual(ci10_4) &
    concreteIndividual(ci10_5) & ~(ci10_1=ci10_2) & ~(ci10_2=ci10_3
    ) & ~(ci10_3=ci10_4) & ~(ci10_4=ci10_5) & properPart(ci10_1,
    ci10_2) & properPart(ci10_3,ci10_4) & sum(ci10_5,ci10_3,ci10_4)
% )).
```

```
%%%%%%%%%%%%%%%%%%%%%% Composition %%%%%%%%%%%%%%%%%%%%%

fof(ax_function, axiom,  (
  ![X,Y]: (functionsAs(X,Y) => (endurant(X) & type_(Y)))
)).
```

2.3.9 Composition

```
  ![X,Y]: (functionsAs(X,Y) => (endurant(X) & type_(Y)))
)).

fof(ax_genericFunctionalDependence_a55, axiom, (
  ![T1,T2,W]: (gfd(T1,T2,W) <=>
    ![E1]: ((iof(T1,E1,W) & functionsAs(T1,E1)) => ?[E2]: (~(E1=E2) & iof(T2,E2,W) & functionsAs(T2,E2))))
)).

fof(ax_individualFunctionalDependence_a56, axiom, (
  ![E1,T1,E2,T2,W]: (ifd(E1,T1,E2,T2,W) <=> (
    gfd(T1,T2,W) & iof(E1,T1,W) & iof(E2,T2,W) & (functionsAs(E1,T1) => functionsAs(E2,T2))
  ))
)).

fof(ax_componentOf_a57, axiom, (
  ![E1,T1,E2,T2,W]: (componentOf(E1,T1,E2,T2,W) <=> (properPartOf(E1,E2) & ifd(E1,T1,E2,T2,W)))
)).

% Composition in use
% (tested to rule out trivial models)

% fof(ax_compositionInUse, axiom, (
%   componentOf(e11_1,t11_1,e11_2,t11_2,w11) & ~(e11_1=e11_2) & ~(e11_1=t11_1) & ~(e11_2=t11_2) & ~(e11_1=t11_2) & ~(e11_2=t11_1) & ~(t11_1=t11_2)
% )).

%%%%%%%%%%%%%%%%%%%%% Constitution %%%%%%%%%%%%%%%%%%%%%

fof(ax_constitutedByInvolvedNatures_a58, axiom, (
  ![X,Y,W]: (constitutedBy(X,Y,W) => ((endurant(X) <=> endurant(Y)) & (perdurant(X) <=> perdurant(Y)) & world(W)))
```

2.3.10 Constitution

```

fof(ax_constitutedByDifferentKinds_a59, axiom, (
  ![E1,E2,T1,T2,W]: ((constitutedBy(E1,E2,W) & iof(E1,T1,W) & iof(E2,T2,W) & kind(T1) & kind(T2)) => (~(T1=T2)))
)).

% Ax |= "th_noSelfConstitution_t27"; conjecture commented for convenience

% fof(th_noSelfConstitution_t27, conjecture, (
```

```
%   ~?[X,W]: (endurant(X) & constitutedBy(X,X,W))
% )).

fof(ax_genericConstitutionalDependence_a60, axiom, (
  ![T1,T2]: (genericConstitutionalDependence(T1,T2) <=> (
    type_(T1) & type_(T2) & ![E1,W]: (iof(E1,T1,W) => (
      ?[E2]: (constitutedBy(E1,E2,W) & iof(E2,T2,W)
    )))
  ))
)).

fof(ax_constitution_a61, axiom, (
  ![E1,T1,E2,T2,W]: (constitution(E1,T1,E2,T2,W) <=> (
    iof(E1,T1,W) & iof(E2,T2,W) & genericConstitutionalDependence(T1,T2) & constitutedBy(E1,E2,W)
  ))
)).

fof(ax_wheneverAConstitutedPerdurantExistsTheConstitutedByRelationHolds_a62, axiom, (
  ![P1,P2,W1]: ((constitutedBy(P1,P2,W1) & perdurant(P1)) => (![W2]: (exists(P1,W2) => constitutedBy(P1,P2,W2))))
)).

fof(ax_constitutedByIsAsymmetric_a63, axiom, (
  ![E1,E2,W]: (constitutedBy(E1,E2,W) => ~constitutedBy(E2,E1,W))
)).

% Constitution in use
% (tested to rule out trivial models)

% fof(ax_constitutionInUse, axiom, (
%   object(e12_1) & object(e12_2) & objectKind(k12_1) & objectKind(k12_2) & world(w12) & ~(k12_1=k12_2) & iof(e12_1,k12_1,w12) & iof(e12_2,k12_2,w12) & constitutedBy(e12_1,e12_2,w12) & genericConstitutionalDependence(k12_1,k12_2) & constitution(e12_1,k12_1,e12_2,k12_2,w12)
% )).

%%%%%%%%%%%%%%%% Existential Dependence %%%%%%%%%%%%%%%%

fof(ax_exists_a64, axiom, (
  ![X,W]: (exists(X,W) => (thing(X) & world(W)))
```

2.3.11 Existential Dependence

```

fof(ax_existentiallyDependsOn_a65, axiom, (
  ![X,Y]: (existentiallyDependsOn(X,Y) <=> (![W]: (exists(X,W) => exists(Y,W))))
)).

fof(ax_existentiallyIndependentOf_a66, axiom, (
  ![X,Y]: (existentiallyIndependentOf(X,Y) <=> (~existentiallyDependsOn(X,Y) & ~existentiallyDependsOn(Y,X)))
)).

```

```
% Existential dependence in use
% (tested to rule out trivial models)

% fof(ax_constitutionInUse, axiom, (
%   object(e13_1) & object(e13_2) & object(e13_3) & ~(e13_1=e13_2) & ~(e13_1=e13_3) & ~(e13_2=e13_3) & existentiallyDependsOn(e13_2,e13_1) & existentiallyIndependentOf(e13_3,e13_1)
% )).

% TODO: introduce transitivity and anti-symmetry of existential dependence
% TODO: introduce continuity of existence with perdurants never ceasing to exist

%%%%%%%%%%%%%%%%%% Moments and Inherence %%%%%%%%%%%%%%%%%

%  Inherence
```

2.3.12 Moments and Inherence

```
  ![M,X]: (inheresIn(M,X) => existentiallyDependsOn(M,X))
)).

fof(ax_thingsInvolvedInInherence_a68, axiom, (
  ![M,X]: (inheresIn(M,X) => (moment(M) & (type_(X) | endurant(X))))
)).

% TODO: add definition (d5) for the "bearer" axiom

fof(ax_irreflexiveInherence, axiom, (
  ![X]: (~inheresIn(X,X))
)).

fof(ax_asymmetricInherence, axiom, (
  ![X,Y]: (inheresIn(X,Y) => ~inheresIn(Y,X))
)).

fof(ax_intransitiveInherence, axiom, (
  ![X,Y,Z]: ((inheresIn(X,Y) & inheresIn(Y,Z)) => ~inheresIn(X,Z))
)).

fof(ax_uniqueInherence_a69, axiom, (
  ![X,Y,Z]: ((inheresIn(X,Y) & inheresIn(X,Z)) => (Y=Z))
)).

% Moments

fof(ax_dMomentOf_d6, axiom, (
  ![M,X]: (momentOf(M,X) <=> (inheresIn(M,X) | (
    ?[M2]: (inheresIn(M,M2) & momentOf(M2,X))
  )))
)).

fof(ax_dUltimateBearerOf_d7, axiom, (
  ![B,M]: (ultimateBearerOf(B,M) <=> (~moment(B) & momentOf(M,B)))
)).
```

```
fof(ax_everyMomentHasUniqueAUltimateBearer_a70, axiom, (
  ![M]: (moment(M) => (?[B]: (ultimateBearerOf(B,M) & (
    ![B2]: (ultimateBearerOf(B2,M) <=> (B=B2))
  ))))
)).

fof(ax_noMomentOfCycles, axiom, (
  ~?[M]: momentOf(M,M)
)).

% Ax |= "th_irreflexiveInherence_t28"; conjecture commented for convenience

% fof(th_irreflexiveInherence_t28, conjecture, (
%   ~?[X]: (inheresIn(X,X))
% )).

% Ax |= "th_asymmetricInherence_t29"; conjecture commented for convenience

% fof(th_asymmetricInherence_t29, conjecture, (
%   ~?[X,Y]: (inheresIn(X,Y) & inheresIn(Y,X))
% )).

% Ax |= "th_antiTransitiveInherence_t30"; conjecture commented for convenience

% fof(th_antiTransitiveInherence_t30, conjecture, (
%   ![X,Y,Z]: ((inheresIn(X,Y) & inheresIn(Y,Z)) => (~inheresIn(X,Z)))
% )).

% TODO: add instances

%%%%%%%%%%%%%%%%%%%%%%%% Relators %%%%%%%%%%%%%%%%%%%%%%%%

% External Dependence and Externally Dependent Modes
```

2.3.13 Relators

```
  ~?[M,X]: (externallyDependsOn(M,X) <=> (existentiallyDependsOn(M,X) & (![Y]: (inheresIn(M,Y) => existentiallyIndependentOf(X,Y))))))
)).

fof(ax_dExternallyDependentMode_a72, axiom, (
  ![M]: (externallyDependentMode(M) <=> (mode(M) & (?[X]: (externallyDependsOn(M,X)))))
)).

% Founded by

fof(ax_foundedByInvolvedThings_a73, axiom, (
  ![M,P]: (foundedBy(M,P) <=> ((externallyDependentMode(M) | relator(M)) & perdurant(P)))
)).

fof(ax_relationalModesHaveAFoundationEvent_a74, axiom, (
```

```
  ![M]: ((externallyDependentMode(M) | relator(M)) => (?[P]: (
    foundedBy(M,P))))
)).

fof(ax_uniqueFoundationEvents_a74, axiom, (
  ![M,P1,P2]: ((foundedBy(M,P1) & foundedBy(M,P2)) => (P1=P2))
)).

% TODO: add definition (d8) for the "foundationOf" axiom

% Qua Individual

fof(ax_dQuaIndividualOf_a75, axiom, (
  ![X,Y]: (quaIndividualOf(X,Y) <=> (![Z]: (overlap(Z,X) <=> (
    externallyDependentMode(Z) & inheresIn(Z,Y) & (![P]: (foundedBy
    (X,P) => foundedBy(Z,P)))
  ))))
)).

% Ax |= "
    th_thePartsOfAQuaIndividualShareTheFoundationOfTheWhole_t31";
    conjecture commented for convenience

% fof(th_thePartsOfAQuaIndividualShareTheFoundationOfTheWhole_t31,
    conjecture, (
%   ![X,Y,Z]: ((quaIndividual(X) & partOf(Z,X)) => (![P]: (
    foundedBy(Z,P) => foundedBy(X,P))))
% )).

fof(ax_dQuaIndividual_a76, axiom, (
  ![X]: (quaIndividual(X) <=> ?[Y]: (quaIndividualOf(X,Y)))
)).

% Qua Individual is already defined as a subtype of Externally
    Dependent Mode in the taxonomy; skipping (a78)

% Skipping (a79); already defined in (a74)

fof(ax_thePartsOfARelatorShareTheFoundationOfTheWhole_a80, axiom, (
  ![X,Y,Z]: ((relator(X) & partOf(Z,X)) => (![P]: (foundedBy(Z,P)
    => foundedBy(X,P))))
)).

fof(ax_dRelator_a81, axiom, (
  ![R]: (relator(R) <=> (
    (?[X]: (properPartOf(X,R))
    & (![Y,Z]: ((properPartOf(Y,R) & properPartOf(Z,R)) => (
    quaIndividual(Y) & quaIndividual(Z) & existentiallyDependsOn(Y,
    Z) & existentiallyDependsOn(Z,Y) & (![P]: (foundedBy(Y,P) <=>
    foundedBy(Z,P))))))
    & (![Y2,Z2]: ((properPartOf(Y2,R) & quaIndividual(Z2) &
    existentiallyDependsOn(Y2,Z2) & existentiallyDependsOn(Z2,Y2) &
     (![P2]: (foundedBy(Y2,P2) <=> foundedBy(Z2,P2)))) => (
    properPartOf(Z2,R))))
  )))
)).
```

```
% Ax |= "th_relatorsImplyTheExistenceOfAtLeastTwoQuaIndividuals_t32"; conjecture commented for convenience

% fof(th_relatorsImplyTheExistenceOfAtLeastTwoQuaIndividuals_t32, conjecture, (
%   ![R]: (relator(R) => (?[Q1,X,Q2,Y]: (quaIndividualOf(Q1,X) & quaIndividualOf(Q2,Y) & ~(Q1=Q2))))
% )).

fof(ax_dMediates_a82, axiom, (
  ![R,E]: (mediates(R,E) <=> (relator(R) & endurant(E) & (?[Q]: (quaIndividualOf(Q,E) & partOf(Q,R)))))
)).

% Ax |= "th_relatorsMediateAtLeastTwoThings_t33"; conjecture commented for convenience

% fof(th_relatorsMediateAtLeastTwoThings_t33, conjecture, (
%   ![R]: (relator(R) => (?[E1,E2]: (~(E1=E2) & mediates(R,E1) & mediates(R,E2))))
% )).

% TODO: add definition (d9) for the "relator bearer" axiom

% TODO: add instances

%%%%%%%%%%%%%%%%%%%%% Characterization %%%%%%%%%%%%%%%%%%%%%

fof(ax_endurantTypeCharacterizationByMomentTypes_a83, axiom, (
  ![ET,MT]: (characterizes(MT,ET) => (
```

2.3.14 Characterization

```
    & momentType(M)
    & (![E,W]: (iof(E,ET,W) => (?[M]: (iof(M,MT,W) & inheresIn(M,E)))))
    & (![M2,W2]: (iof(M2,MT,W2) => (?[E2]: (iof(E2,ET,W2) & inheresIn(M2,E2)))))
  ))
)).

% Ax |= "th_qualitiesInheresInAUniqueEndurantConnectThroughCharacteization_a84"; conjecture commented for convenience

% fof(th_qualitiesInheresInAUniqueEndurantConnectThroughCharacteization_a84, conjecture, (
%   ![QT,ET]: ((characterizes(QT,ET) & qualityType(QT)) => (![Q,W]: (iof(Q,QT,W) => (?[E]: (iof(E,ET,W) & inheresIn(Q,E) & (![E2]: (inheresIn(Q,E2) <=> (E=E2))))))))
% )).

% TODO: add instances

%%%%%%%%%%%% Qualities and Quality Structures %%%%%%%%%%%%

% Skipping (a85); previously introduced in the taxonomy
```

```
% Skipping (a86); previously introduced in the taxonomy
```

2.3.15 Qualities and Quality Structures

```

% Quality Structures

fof(ax_dQualityStructure_d10, axiom, (
  ![QS]: (qualityStructure(QS) <=> (?[QT]: (qualityType(QT) &
    associatedWith(QS,QT))))
)).

fof(ax_dQualityStructure_d10, axiom, (
  ![QS]: (qualityStructure(QS) <=> (?[QT]: (qualityType(QT) &
    associatedWith(QS,QT))))
)).

%%%%%%%%%%%%%%%% Endurants and Perdurants %%%%%%%%%%%%%%%%

fof(ax_manifestsInvolvedThings_a104, axiom, (
  ![E,P]: (manifests(E,P) => (endurant(E) & perdurant(P)))
)).

fof(ax_lifeOfInvolvedThings_a105, axiom, (
  ![E,P]: (lifeOf(P,E) => (
```

2.3.16 Endurants and Perdurants

```
    & (![P2]: (overlap(P2,P) <=> (perdurant(P2) & manifests(E,P2)))
    )
  ))
)).

% TODO: review ax_lifeOfInvolvedThings_a105 and its translation of
    the small sigma predicate schema in (a105)

fof(ax_meetsInvolvedThings_a106, axiom, (
  ![P1,P2]: (meets(P1,P2) => (perdurant(P1) & perdurant(P2)))
)).

% TODO: add instances
```