Formal Ontology in Information Systems
M. Donnelly and G. Guizzardi (Eds.)
IOS Press, 2012

doi:10.3233/978-1-61499-084-0-191

A Method for Evaluating Ontologies

Introducing the BFO-Rigidity Decision Tree Wizard

A. Patrice Seyed [1]
Department of Computer Science and Engineering
Center for Cognitive Science
University at Buffalo, NY, USA

Abstract. In this paper we review the integration of BFO's theory of types with OntoClean's notion of Rigidity, provide our decision tree procedure for evaluating ontologies based on the integration, while also describing its implementation as a Protégé 4 plugin, the BFO-Rigidity Decision Tree Wizard. Finally we provide a practical analysis of controversial and important ontological topics surrounding the BFO-Rigidity integration work. The decision tree approach allows our wizard plugin to implicitly perform inferences on behalf of a modeler based on answers to questions. This approach is accessible because it does not require familiarity with BFO, OntoClean, or our first-order formal system, and does not require the modeler to make assertions that are not normally considered within the scope of a domain level ontology. Having chosen for our implementation a plugin environment that interoperates with a popular ontology editor, we expect that the principles underlying the integration work will become more accessible to both novice and expert domain modelers.

Keywords. BFO, OntoClean, ontology

Introduction

BFO is the designated upper ontology for the OBO Foundry. Although there is documentation listed as OBO Foundry principles on its website (http://obofoundry.org), still, there is no formal principled criteria that a candidate domain ontology must meet for ratification into the OBO Foundry. To address this problem we proposed a formal integration between OntoClean's notion of Rigidity and BFO's theory of types [1].[2] In this paper we review this integration, and provide our decision tree procedure for evaluating ontologies based on the integration. We also describe the BFO-Rigidity Decision Tree Wizard, a Protégé [2] plugin which implements the decision tree procedure. Having chosen for our implementation a plugin environment that interoperates with a popular ontology editor, we expect that the principles underlying the integration work will become more accessible to both novice and expert domain modelers (i.e., ontologists).

[1] Corresponding Author: A. Patrice Seyed, Department of Computer Science and Engineering, University at Buffalo, 201 Bell Hall, Buffalo, NY, 14260, USA; Email: apseyed@buffalo.edu.

[2] We provide revised sections of that paper.

Integration

OntoClean uses *properties* as its categorical unit, which are the intension, or meaning, of general terms. BFO uses *types*, which are defined as that in reality to which the general terms of science refer. As described in [1], to address the differing ontological perspectives committed to by BFO and OntoClean, that is, realism and a mixture of natural language semantics and epistemology, we "unify" property and types under the categorical unit *class*. In what follows, we assume a first-order, sorted logic. Although there are many theories of existence, we introduce a relation, **exists_at**(x,t), which is non-committal and means that, under a certain ontological theory, object x is within its domain and x exists at some time, t. Everything exists at some time [1].

member_of(x,A,t) means that object x satisfies the definition of class A at t. With the **member_of**(x,A,t) relation, there is no commitment about the nature of A; therefore, membership at a time *does not* presuppose that existence spans that time. This formula maintains first-order expressivity because classes are treated as first-order objects of the domain and are represented by terms, not predicates. Only particulars, not classes, are members of a class. The classification relation **subclass_of**(A,B) is defined such that if x is a member of A at t then x is a member of B at t.

A particular class might or might not satisfy the unary predicate **Instantiated**, which means there is some member of A at t that exists at t:

Definition 1. $\mathbf{Instantiated}(\mathrm{A}) =_{\mathrm{def}} \exists xt(\mathbf{member\_of}(x,A,t) \wedge \mathbf{exists\_at}(x,t))$

The class *Full_Eye_Transplant* does not satisfy **Instantiated** because no such procedure has been performed yet (therefore it has no members and is thus empty).

If a class has as members only those objects that exist at all times at which they are members, it satisfies the predicate **Members_Exist**:

Definition 2. $\mathbf{Members\_Exist}(A) =_{\mathrm{def}} \forall xt(\mathbf{member\_of}(x,A,t) \rightarrow \mathbf{exists\_at}(x,t))$

Assuming a class *Animal* is defined to have as members animals as long as they are not dead and decayed, **Members_Exist**(*Animal*) holds.[3]

Rigidity has been defined in terms of S5 modal logic [3]. As part of our integration, we provided just the underlying intuitions of those modal formalisms, prior to reformulating Rigidity in our formal system. Each object that has a Rigid property has that property at all times at which the object exists. We formalize this in terms of classes, instead of properties, by the predicate **Rigid**:

Definition 3. $\mathbf{Rigid}(A) =_{\mathrm{def}} \forall x(\exists t(\mathbf{member\_of}(x,A,t)) \rightarrow \forall t_1(\mathbf{exists\_at}(x,t_1) \rightarrow \mathbf{member\_of}(x,A,t_1)))$

Rigid(*Person*) means that all members of the class *Person* are people at all times at which they exist. As an amendment to the original formulation of Rigid, [3] proposes that Rigid properties are only instantiated by actually existing objects. We have captured

[3] Although it may be more intuitive to equate an organism's existence with living, BFO also includes the timespan that is after death and before decomposition (B. Smith, Personal Communication).

this intuition separately from Rigid, under the **Members_Exist** predicate. Also, because unexemplifiable properties are trivially Rigid, [3] constrains the theory (as suggested by [4] and [5]) to properties for which there exists some instance.[4] We have separately defined this notion, also, under the **Instantiated** predicate.

Non-Rigid is the negation of Rigid, which we applied for our class formulation under the predicate **Non-Rigid**:

Definition 4. $\textbf{Non-Rigid}(A) =_{\text{def}} \neg\textbf{Rigid}(A)$

Assuming that a person is a member of *Student* only while a registered student, **Non-Rigid**(*Student*) holds.

Anti-Rigid is true of a property, if, for every object that has that property, it is *possible* that it does not have that property at some time. An object may have an Anti-Rigid property at all times at which it exists. BFO is not concerned with what could have been, but rather what has been or currently is; therefore, Anti-Rigid is irrelevant to our theory [1].

Note that by eliminating Anti-Rigid we cannot constrain the classification relation in the same manner as was intended for OntoClean. In the modal system, it can be proven that an Anti-Rigid property can only subsume Anti-Rigid properties. In our system there is no notion of Anti-Rigid, and further, we cannot show that the **subclass_of** relation does not hold between two classes when one is Non-Rigid and the other is Rigid. For example, a class *Human* is Rigid and is the subclass of a Non-Rigid class defined by the disjunction of *Human* and *Student*,[5] and conversely, the class *Student* is Non-Rigid and is a subclass of a Rigid class *Human*. The assignments of Rigid and Non-Rigid cannot be immediately applied to inform a modeler when the **subclass_of** relation cannot hold between classes, as was the case for assignments that included Anti-Rigid in OntoClean. The Rigid/Non-Rigid distinction, however, is useful within the scope of BFO's theory of types, as we explain in what follows.

The objects of BFO's domain are partitioned into *particulars* and *types*. Particulars are entities confined to specific spatial, spatiotemporal, or temporal regions (e.g., a specific grasshopper in front of me, its life, or the time interval that its life spans, respectively). Under BFO's theory, existence of a particular is based on it being observable at some level of granularity and/or causal by some scientifically-based measure. Numbers, for example, do not exist in BFO.[6] **Type**(*A*) means that *A* is a class that meets the criteria for being a type, which we provide in what follows.

For the purposes of our evaluation method, we exclude classes that are controversial with respect to identity, such as *Embryo* and *Fetus* from our domain; hence, types satisfy **Rigid**. Types must also satisfy **Instantiated** [6]. A third criterion for every class that is a type is that every member of the class at a time exists at that time; therefore, every type satisfies **Members_Exist**. We therefore provide a set of criteria that every type satisfies:

[4] In OntoClean, "*x* instantiates a property" means that *x* has the property, which, as we describe shortly, is different quite from BFO's use of 'instantiates'.

[5] Note that it is only contingent that a class defined by the disjunction of a Rigid class and a Non-Rigid class is Non-Rigid. Our example assumes non-human students (e.g., service animals in training), but alternatively under the assumption that all students are humans, a class defined by the disjunction of *Human* and *Student* is Rigid.

[6] Note however that BFO is not a closed world artifact. Furthermore the proposals we make refer strictly to BFO 1.1, since BFO evolves only very incrementally, but the bulk of the ideas will be valid even for the later versions of BFO.

Axiom 1. $\textbf{Type}(A) \rightarrow \textbf{Rigid}(A) \wedge \textbf{Instantiated}(A) \wedge \textbf{Members\_Exist}(A)$

The right hand side does not suffice as sufficient criteria due to "false positive" classes that are not what BFO consider types, for example, *Born in North America or Asia*. With these necessary conditions in hand, we define the notion of instantiation; x is an instance of A at t means that x is a member of A at t and A is a type:

Definition 5. $\textbf{instance\_of}(x,A,t) =_{\text{def}} \textbf{member\_of}(x,A,t) \wedge \textbf{Type}(A)$

It follows that every x that instantiates a type at t exists at t.

The root type of the BFO upper ontology is *Entity*; *Continuant* and *Occurrent* are its subtypes. Continuants (e.g, bodily organs) exist fully in different time instants, but occurrent happen over time, occupy regions of time and/or space, therefore have time as a part. Strictly speaking occurrents do not exist in time, but for formal reasons we need to add time indexes to some statements about processes; thus, it becomes possible to add time indexes to statements asserting that processes exist, but these hold for all times:

Axiom 2. $\exists t(\textbf{instance\_of}(x,\mathit{Occurrent},t)) \rightarrow \forall t_1(\textbf{instance\_of}(x,\mathit{Occurrent},t_1))$

It follows that if an object instantiates *Occurrent* for some time, it exists for all time.

is_a(A,B) is the "backbone" BFO relation for scientific classification (i.e., building taxonomies) and means that if x is an instance of type A at t then x is an instance of type B at t. **is_a** is provably reflexive, transitive, and anti-symmetric. We exclude our treatment of the direct subtyping relation **direct_is_a** [1], due to space constraints.

Following Aristotle's division of objects into substances and accidents, the two subtypes of *Continuant* are *IndependentContinuant* (*IC*) and *DependentContinuant* (*DC*), respectively.[7] The shape of a specific cell instantiates *SpecificallyDependentContinuant* (*SDC*), and "depends on" a specific cell, which instantiates *IC*. **depends_on**(x,y,t) means that the specifically dependent continuant x exists at t only if the independent continuant y exists at t. It also means that x cannot migrate to another independent continuant.

Depends_On(A,B) means that for every instance of A there is some instance of B where the former instance depends on the latter:

Definition 6. $\textbf{Depends\_On}(A,B) =_{\text{def}} \forall xt(\textbf{instance\_of}(x,A,t) \rightarrow \exists y(\textbf{instance\_of}(y,B,t) \wedge \textbf{depends\_on}(x,y,t)))$

If we assume the class *Student* has as members people at times at which they have the role of student, *Student* satisfies *Non-Rigid* and is not a type. Alternatively, if the class *Student* is "re-conceived" as having as members individual student roles, which are instances of *SDC*, then **Depends_On**(*Student*,*Person*) holds and the class satisfies **Type**. At each time t at which some person x is a student, there exists some y that is a student role and is dependent on x (i.e., $\exists y(\textbf{instance\_of}(y,\mathit{Student},t) \wedge \textbf{depends\_on}(y,x,t))$). Since this type falls under *SDC* and what in BFO's theory is a *Role* type, a more fitting lexical designator for this class is *Student Role*.

[7] Due to space, we omit treatment of the *DC* subtype *GenericallyDependentContinuant* (*GDC*), and restrict our discussion to the *DC* subtype *SpecificallyDependentContinuant* (*SDC*).

BFO's theory of types is also committed to the *Disjointness Principle*,[8] that two types have no instances in common unless one is a subtype of the other. We also define a relation **disjoint_from**(*A*,*B*), which holds iff types *A* and *B* do not share any instances at any time. It follows sibling BFO upper ontology types (e.g., *Continuant* and *Occurrent*), and more generally, any types not related by **is_a**, are disjoint types.

A class that has as members instances of disjoint upper ontology types, it satisfies **Heterogeneous**:

Definition 7. $\textbf{Heterogeneous}(A) =_{\text{def}} \exists xBCt(\textbf{member\_of}(x,A,t) \wedge \textbf{member\_of}(x,B,t) \wedge \textbf{member\_of}(x,C,t) \wedge \textbf{disjoint\_from}(B,C))$

Based on this definition and the definition of **instance_of** we can show that a heterogeneous class is instantiated; therefore, the definition excludes empty classes, which is intuitive given the predicate's typical meaning.

Decision Tree and Implementation

For our evaluation method, in what follows we present a decision tree procedure based on the BFO-Rigidity integration, while at the same time introduce the BFO-Rigidity Decision Tree Wizard Plugin, developed for Protégé 4, which implements the procedure. We chose the wizard-style plugin format because it lends itself to the decision-tree's question-asking. Within a software installation wizard a user is presented with an opportunity to configure the software with certain parameters; however, the installation may not complete its execution successfully due to requirements of the software not being met (e.g., not enough disk space). Similarly, the Wizard plugin allows a user to formally model a class in many different ways; nevertheless, the Wizard Plugin will not allow a class conceived in a way not compliant with BFO to be added to an ontology.

The Wizard Plugin follows the decision-tree procedure, and assumes that a modeler is starting to build an ontology from scratch, and introduces one class at a time. In the decision tree we use natural language questions in everyday language to primarily determine from the modeler if **Instantiated**, **Members_Exist**, **Rigid** or **Non-Rigid** hold. In some screens of the Wizard plugin we present hints in the form of tips and examples, aimed to help a modeler understand the questions and ultimately BFO's theory. Further, in the event where the Wizard Plugin does not add a class to the ontology based on how the class is conceived by the modeler, the system presents an explanation as to why the class was not added to the ontology.

Figure 1 illustrates the decision tree procedure. The descriptions of the answer choices for **Question 2** correspond to more commonly modeled types under *IC*, *DC*, and *Occurrent*, namely *MaterialEntity*, *SpecificallyDependentContinuant*, and *Process*.[9] The other major types of BFO, *SpatialRegion*, *TemporalRegion*, and *SpatioTemporalRegion*, and are based on the Newtonian space-time container theory. We exclude these types

[8] Our work is based on BFO version 1.1, which we consider stable and "frozen" for our research. Recent work [7] indicates this principle only applies to the asserted **is_a** hierarchy. We address this topic in detail in a later section.

[9] This is reflected in the Gene Ontology's division of classes into *Cellular Component*, *Molecular Function*, and *Biological Process*.

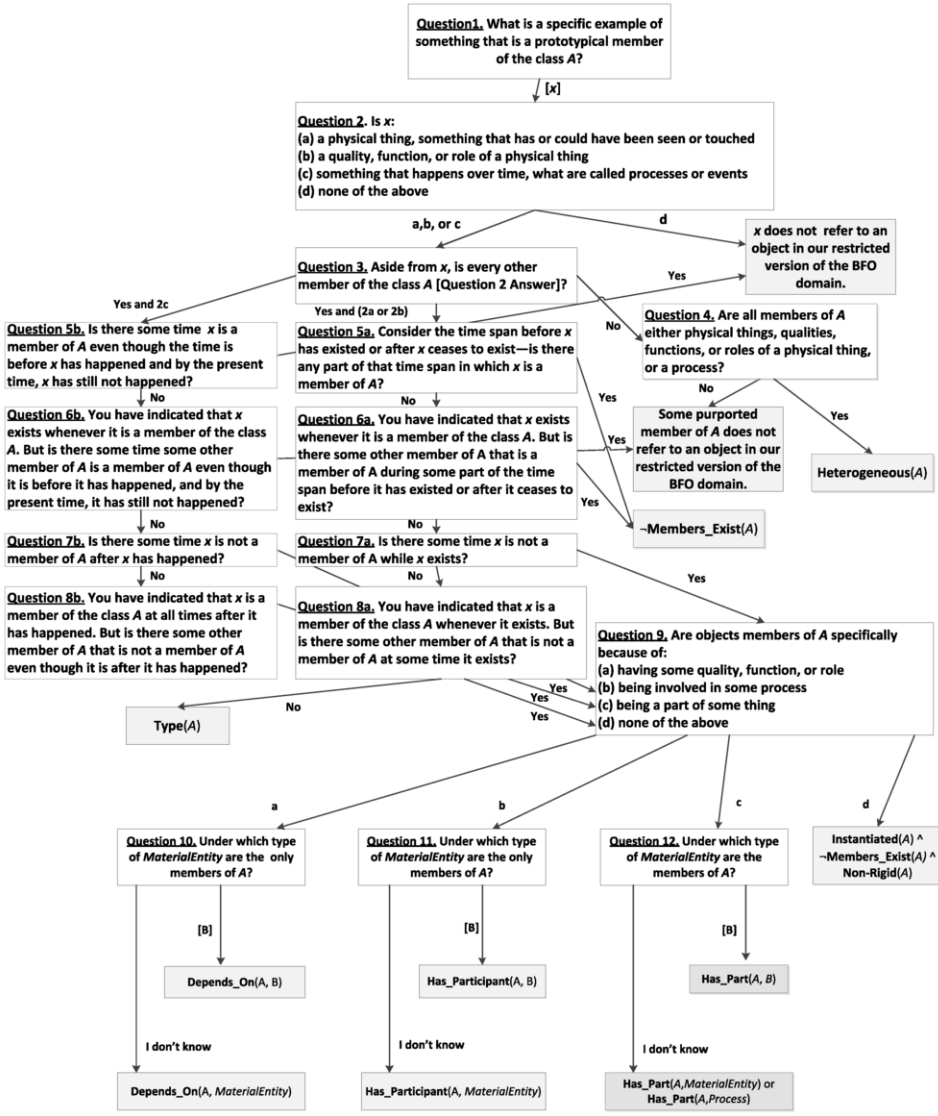

Figure 1. Decision Tree for Standardizing a Candidate Type

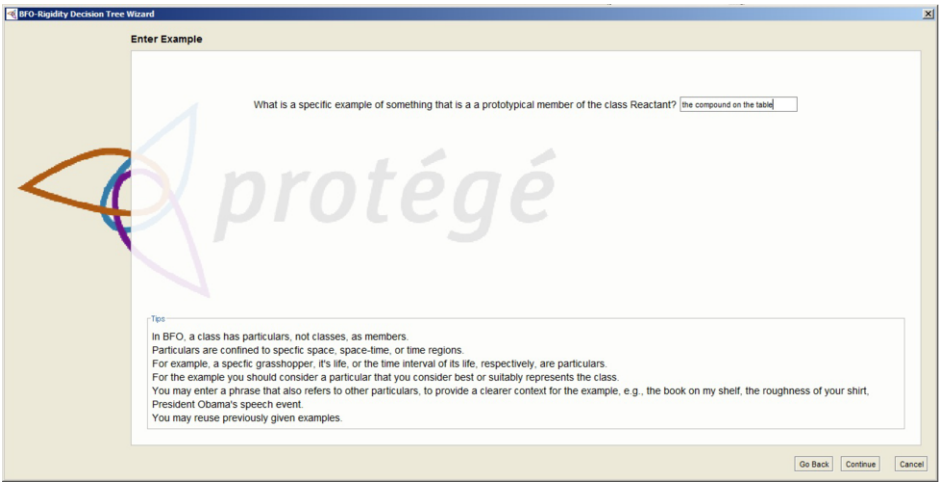

Figure 2. 'Enter Example' Screen for the class *Reactant*

from our evaluation work simply because their instances are not the sort of objects scientists reference directly in real-world settings. This is evidenced by the fact that there are no subtypes for these in the OBO Foundry's Ontology for Biomedical Investigations (see http://purl.obolibrary.org/obo/obi.owl). There are certain other types, (e.g., *GenericallyDependentContinuant*) that will appear in an expanded version of the tree, in future work.

In what follows we describe the decision tree procedures in terms of its questions and also we present formal correspondences.At the same time, we provide an example scenario where a modeler introduces a class, *Reactant*. We present Wizard plugin screenshots where interesting, which is frequently where the Wizard plugin provides additional tips.

After a class name *A* is entered, the first question (**Question 1**) asks: "What is a specific example of something that is a prototypical member of the class *A*?" In our example scenario the modeler enter "this compound on the table". In the corresponding Wizard screen (**Figure 2**), various tips are given to help the modeler better answer the question; for example, "Avoid mass nouns, they refer to cross-granular classes, which cause problems for BFO." This tip is based on our other work, an integration of BFO with OntoClean's notion of Unity, but is introduced here being it is crucial for understanding what a particular is. Based on that integration work, classes that OntoClean considers as having Anti-Unity causes an inconsistency in a BFO-compliant ontology. Another tip is: "A class has particulars, not classes, as members." This helps avoid the modeling of meta-classes, which is not compliant with our theory of classes and subsequently also not compliant with BFO.

Once **Question 1** is answered the modeler has entered both a class name *A* and an example member *a* of the class, therefore it is implicitly asserted that $\exists t(\mathbf{member\_of}(a,A,t))$. **Question 2** attempts to determine if *a* is an instance of the type (a) *MaterialEntity*, (b) *SpecificallyDependentContinuant*, or (c) *Process* by applying natural language descriptions of each. In our example scenario the modeler chooses '(a)'.

If **Question 2** is answered '(d)', then based on our restriction to material entities, specifically dependent continuants, and processes it is not the case that what 'a' represents falls within our restricted version of BFO's domain. When (d) is chosen, an error message is given that reflects this issue. The choice (d) actually corresponds, although contradictorily, to the assertion $\neg\exists t(\textbf{exists\_at}(a,t))$. As mentioned previously, if some x is in BFO's domain it exists at some time. What cannot be represented, due to the contradictory nature of the assertion, but is assumed under BFO theory, is that if something does not fall within what x ranges over it does not exist at any time. Formally the assertion leads to an inconsistent ontology because by application of the aforementioned axiom $\neg\exists t(\textbf{exists\_at}(a,t)) \wedge \exists t(\textbf{exists\_at}(a,t))$, a contradiction.

In order to revise the ontology such that it is consistent, at least one of the conjuncts must be removed. Because the latter conjunct follows from an axiom of our system, we consider if the former conjunct should be removed, $\neg\exists t(\textbf{exists\_at}(a,t))$. Unfortunately, removal of this conjunct is not the case either since the choice of (d) confirms it. Given this, the overall conjunctive assertion and a as a domain object should be removed, which follows from the fact that the purported particular a is not a particular within the restricted BFO domain. Since there may be other objects the modelers consider members of A, it is not necessarily the case that A is an empty class, but this is the case if every member follows the modeler's assumptions for a.

Also note that even if the modeler answers (a), (b), or (c) for **Question 2**, it is not necessarily the case that, according to BFO's theory, x exists at some time. The reason is that the modeler may interpret that the question is asked in such a way that it implies that x is something that exists. For example the question "Is Bigfoot a vertebrate?" may be interpreted as "*If Bigfoot were real*, is Bigfoot a vertebrate?" In cases such as this, the question is unintendedly a trick question.

Given this issue one might suggest a potential question reordering such that a question about the existence of the example come before a question about its categorization. This is also troublesome because existence means something different based on the type of particular being considered. To mitigate these problems, the current approach attempts to ground the example in notions that are more accessible to the modeler. First the modeler is asked for an example for the class she is modeling (**Question 1**), and second the modeler is asked to choose a corresponding description, each of which describes one of the BFO types *MaterialEntity*, *SDC*, and *Process* (**Question 2**). To mitigate the issue, tips are given to highlight what existence is in BFO, where negative examples are given when the categorization is requested.

Question 3 asks a question to determine if all members of A are instances of the same type selected in **Question 2**. So for instance, if (a) is selected for **Question 2**, this corresponds to the assertion $\exists t(\textbf{member\_of}(a,\textit{MaterialEntity},t))$, and if **Question 3** is answered "yes", then this corresponds to the assertion $\exists t(\textbf{member\_of}(x,A,t)) \rightarrow \exists t_1(\textbf{member\_of}(x,\textit{MaterialEntity},t))$. Note that in the latter assertion time is not bound from the antecedent to the consequent because there is nothing in the question that constrains the members to be material things at all times it is a member of A. Overall, the formal correspondence of this question is relatively weak, but the question primarily isolates classes that have as members instances of disjoint upper ontology types. For our example scenario the modeler confirms all members are material entities. As show in **Figure 1**, **Question 4** tries to help determine if the class satisfies **Heterogeneous**, and if the modeler confirms this, then an error message explains the problem.

Question 5 and **Question 6**, as given in **Figure 1**, determine whether or not the example and then all members of the class exist at all times they are members, respectively.[10] For **Question 5** the answers and corresponding assertions are:

(yes): $\exists t(\textbf{member\_of}(a,A,t) \wedge \neg\textbf{exists\_at}(a,t))$
(no): $\textbf{member\_of}(a,A,t) \rightarrow \textbf{exists\_at}(a,t)$

Based on a 'no' answer to **Question 5**, and the answer to **Question 1**, which corresponds to $\exists t(\textbf{member\_of}(a,A,t)$, it follows that $\exists t(\textbf{member\_of}(a,A,t) \wedge \textbf{exists\_at}(a,t))$. From this formula and **Definition 1** it follows that **Instantiated**(A). From the 'yes' assertion for **Question 5a** it follows that ¬**Members_Exist**(A) (based on **Definition 2**), and from that it follows that ¬**Type**(A) (based on **Axiom 1**). Note that for the version of **Question 5** for processes, **5b**, if 'yes' is the given answer, it would lead to an inconsistent ontology. This is due to the fact that processes exist for all time; therefore, the example falls outside of the domain. For our example scenario the modeler answers 'no' to **Question 5a**.

For **Question 6** the answers and corresponding assertions are:

(yes): $\exists xt(\textbf{member\_of}(x,A,t) \wedge \neg\textbf{exists\_at}(x,t))$
(no): $\textbf{member\_of}(x,A,t) \rightarrow \textbf{exists\_at}(x,t)$

From a 'yes' answer to **Question 6a** it also follows that ¬**Members_Exists**(A) (based on **Definition 2**), and from that it follows that ¬**Type**(A) (based on **Axiom 1**). As with **Question 5,** for the version of **Question 6** for processes, **6b**, if 'yes' is the answer given, it leads to an inconsistent ontology. If a 'no' answer is given for both **Question 5** and **Question 6**, then **Members_Exists**(A) holds and additional questions are posed. For our example scenario the modeler answers 'no' to **Question 6a**.

Question 7 and **Question 8** determine whether or not the example and then all members of the class are members at all time they exist, respectively. For **Question 7** the answers and corresponding assertions are:

(yes): $\exists t(\neg\textbf{member\_of}(a,A,t) \wedge \textbf{exists\_at}(a,t))$
(no): $\textbf{member\_of}(a,A,t) \rightarrow \forall t_1(\textbf{exists\_at}(a,t_1) \rightarrow \textbf{member\_of}(a,A,t_1))$

For **Question 8** the answer and corresponding asserted formulas are:

(yes): $\exists xt(\neg\textbf{member\_of}(x,A,t) \wedge \textbf{exists\_at}(x,t))$
(no): $\textbf{member\_of}(x,A,t) \rightarrow \forall t_1(\textbf{exists\_at}(x,t_1) \rightarrow \textbf{member\_of}(x,A,t_1))$

Based on 'no' answers for **Question 7** and **Question 8**, and the assertion of **Question 1** ($\exists t(\textbf{member\_of}(a,A,t))$), it follows that **Rigid**(A) (based on **Definition 3**). Based on a 'yes' answer for either **Question 7** or **Question 8**, it follows that **Non-Rigid**(A) (based on **Definition 3**). For our example scenario the modeler answers 'no' to **Question 7**. Although we don't formalize it in our system due to some exceptions, for the sake of the decision tree we assume that if A satisfies **Instantiated**, **Members_Exist**, and **Rigid**

[10] **Question** 5, 6, 7, and 8 have alternate versions: '(a)' phrases the question to address the existence of material entities or specifically dependent continuants, and '(b)' phrases the question to address the existence of processes. The version asked of the modeler is based on previous answers. For simplicity, in this chapter we refer to the overall question instead of the specific version.

it also satisfies **Type**. We hope to address the exceptions in future work and in consideration of the upcoming BFO 2.0. For the candidate *A*, the answer 'no' to **Question** 5, 6, 7 or 8 results in the inference that ¬**Type**(*A*) (based on **Axiom 1**).

The rest of the decision tree starting with **Question 9** addresses how to model the candidate based on it satisfying **Non-Rigid** (along with **Instantiated** and **Members_Exist**). **Question 9** tries to determine why *A* is Non-Rigid. The answer choices (a), (b), and (c) correspond to an assertion that objects are member of *A* because of an implicit relationship to a *SDC* particular, a *Process* particular, and being a part of something, respectively. Tips as well as examples are presented within the Wizard Plugin screen to help the modeler decide. One of the difficulties with this question is that, for example, if the class is *Student* then the answer could be (a), assuming the implicit relationship to a student role, or it could be (b), assuming the implicit relationship to the process a student participates in while being enrolled in an academic program (i.e., student life). To help address this ambiguity screen provides the tip "Select the more basic notion; if something is located in *x* because of being a part of *x*, then being part of *x* is the more basic notion." For our example scenario the modeler chooses '(a)'.

The answer for **Question 9** determines if the additional entity to represent is (a) a quality, function, or role, (b) a process (c) a part of something else under the same BFO types. **Question 10**, **11**, and **12** all ask the same question but the assertion made following it is primarily based on the answer to **Question 9**. The question is "Under which type of *MaterialEntity* are the only members of *A*?" The corresponding screen of the Wizard Plugin is slightly more verbose, and here the modeler is given the opportunity to select a superclass from the current ontology's class hierarchy. The Wizard provides the natural language parse for what a material entity is, and asks the modeler to choose the most specific class that all members of the introduced class are also members. For **Question 10** in the example scenario the modeler chooses *MaterialEntity* from the class hierarchy because a more specific class is not yet in the class hierarchy.

The next, 'Further Restriction' screen (pictured in **Figure 3** and not represented in the decision tree figure) attempts to confirm whether or not there is a class more specific than *MaterialEntity* that is in the relationship with the modeled class members. If the modeler selects an option indicating there is, then the decision tree procedure is executed again, initiated again by the modeler being asked to enter the class name for the more specific class. Only if the class satisfies the necessary criteria for being a type, confirmed by application of the decision tree procedure, is the class added into the ontology and as the restriction to the originally introduced class that was determined to be Non-Rigid.

For our example scenario, the modeler selects the option: "There is a class or group of classes more specific than Material Entity which all members of Reactant are members of and which the class hierarchy does not include yet." To complete the example scenario, the modeler enters *Compound* as the class to restrict *Reactant* to which through the decision tree is found to satisfy the necessary criteria for a type. Following *Element* is added through the next iteration after making the same choice again in the 'Further Restriction' screen. **Figure 4** displays the assertions made on behalf of the modeler for the class *Reactant* based on these modeling choices. Note here that Protege is using OWL-DL and therefore lacks expressivity for representing the restriction on a time index formalized in **Definition 6**.

Ultimately, answers to questions of the decision tree correspond to formulas that can be asserted in our formal system, and the inferences based on these formulas are implic-

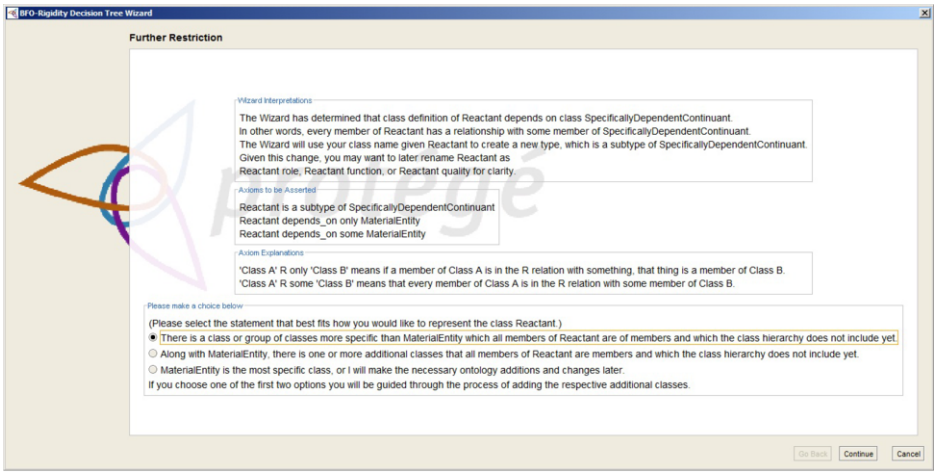

Figure 3. 'Further Restriction' Screen for class *Reactant*

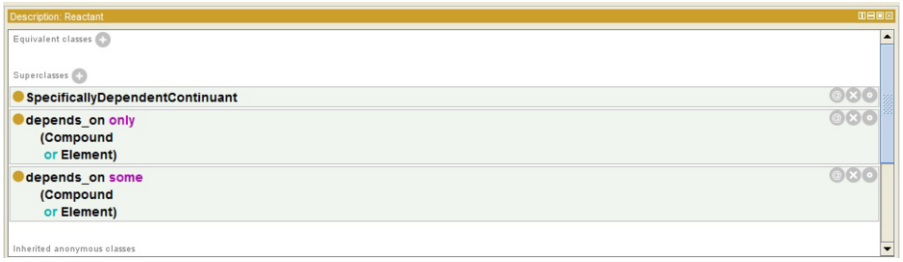

Figure 4. Description Screen for the class *Reactant*

itly made within the decision tree's structure and on behalf of the modeler. This approach avoids having the modeler assert these axioms directly within our formal system, which is useful because these sorts of assertions are not generally considered within the scope of a domain level ontology.

One benefit of forcing a modeler to identify classes that satisfy **Heterogeneous** is that it helps eliminate candidate types that conflate a classification under both *Continuant* and *Occurrent*. Another benefit of isolating classes which satisfy **Heterogeneous** is that it identifies and disallows into a domain ontology candidate types which are inherently upper level. For example, if a modeler introduces a class *Thing* which has as members all entities which BFO considers continuants and occurrents, then it is an upper level class and is redundant with *Entity*.

Practicality of the Disjointness Principle and Modeling Certain Classes that are Not Types

In recent efforts within the OBO Foundry community, it has become clear that the Disjointness Principle is a constraint that is difficult to enforce in practical modeling. In recent work, [7] advocate the principle of asserted single inheritance, citing [8] on the topic. [11]

We evaluate the principle of single inheritance in the scope of an example from OBI, that *Hybridization Oven* is asserted as a subclass of both *Incubator* and *Container*. Together, the two corresponding asserted axioms violate the aforementioned principle of asserted single inheritance. *Incubator* is an inferred subclass of *Device*, and *Container* is fully defined (i.e., the necessary and sufficient conditions are given) such that it is a subclass of *Device* and has the function of contain function. Therefore, only the condition that *Hybridization Oven* has the function of contain function would need to be given to infer that *Hybridization Oven* is a subclass of *Container*. If the definition of *Hybridization Oven* were changed in this manner, i.e., that it has the function contain function instead of being a subclass of *Container*, the violation of the principle of asserted single inheritance, recommended by Rector, would be alleviated.

However the principle of asserted single inheritance misses the point, however, because it focuses on the manner in which an ontology is specified and not on the the classification units applied to the objects of the ontology's domain. There is a way to preserve the original principle of BFO, in part by maintaining the classification units, class and type, and instead adopting another, related principle that [8] sets forth.

The principle is that in an ontology each class has no more than one primitive parent (i.e., immediate superclass). This principle and the principle of asserted single inheritance are similar, and would be one and the same under the assumption that classes can only be asserted as subclasses of those classes that are primitive. Alas, the principles are different because no such assumption is made nor is the assertion of subclasses of defined classes prevented in any ontology tool, such as Protégé.

The notion of a primitive class is connected to that of types because types are extremely difficult to define completely in a formal language [8]. Therefore types are usually primitive classes of an ontology, with the exception of being fully defined via a covering axiom.[12] Note that, however, not all primitive classes are types; a class that is typically defined may be left incompletely specified and thus primitive for any number of potential reasons.

From our example, *Hybridization Oven* is consistent with the one primitive parent principle because *Container* is a defined class, therefore *Hybridization Oven* only has one primitive superclass, *Incubator*. By this approach, then, *Container* is not a type because it is a defined class (that is not due to a covering axiom). In adopting this principle, the modeler must have a way to annotate classes that are primitive but will be defined at a

[11] The notion of asserted versus inferred hierarchy is relevant within the context of Description Logics, where an asserted class hierarchy is constructed purely based on the axioms of an ontology, and an inferred hierarchy is obtained by the application of a classifier (i.e., a classification reasoner) on the set of axioms of an ontology to obtain all subclass relationships.

[12] In the OWL version of BFO (http://www.ifomis.org/bfo/), Entity is defined by the disjunction of Continuant and Occurrent.

later time, for exclusion from the evaluation of whether an ontology is consistent with the one primitive parent principle.

Given this clarification, a larger issue is then whether or not a BFO-compliant ontology may include classes which are not types, those which are usually defined classes. If they may not, then *Container* would be excluded from the ontology, as it is fully defined. In general, there are certain classes, like *Container*, that are not types but satisfy **Instantiated** and **Members_Exist** that may be useful in a domain ontology, and there are other classes that do not satisfy either **Instantiated** or **Members_Exist**, like *Unicorn*, that are not appropriate in a BFO-compliant domain ontology. We continue to exclude both kinds of classes from our ontology evaluation method, since our method remains centered on the importance of initially developing a well-founded type hierarchy, and compliance with BFO 1.1. We do concede, however, in some cases, that the former kind may have utility and recommend that they be added later in development of the ontology, for application-specific purposes and with clear annotation that designates them as such. The type hierarchy, hence, provides a foundation for adding such classes that are fully defined.

By our approach the label applied for the class of students that is originally conceived by the modeler is applied to a different class, that which BFO considers a subtype of *Role*. With a change in policy that allows for defined classes, alternatively the class *Student* would be kept in the ontology and defined by its relationship with a class in the *Role* hierarchy (here assumed *Student-Role*) and its classification under a BFO type (here assumed *Person*): **subclass_of**(*Student*, *Person*) ∧ **Has_Dependent**(*Student*, *Student-Role*) ∧ **Depends_On**(*Student-Role*, *Person*).[13] Clearly then, such a change in policy allows the modeler to still introduce the class she originally had in mind, in the case of Non-Rigid classes like this one.

Conclusions

Violations of disjointness axioms of an ontology loaded into Protégé are made apparent only after a classifier (i.e., DL reasoner) is run on the ontology, therefore there may be inconsistencies in the ontology without a modeler's knowledge. To mitigate this problem, [9] encourage users to execute a classifier early and often. However, in typically modeling situations, the classifier may not be run until after several modeling mistakes have been made. Our plugin, based on our decision tree algorithm, enforces that an ontology remains consistent with respect to the disjointness axioms of upper ontology types, by asking certain questions, and applying the respective answers to restricting where a class is rooted. There is no currently known plugin for Protégé that accomplishes this for a modeler with respect to an upper ontology.

In performing this work we have discovered that a reformulated notion of Rigidity can play a key role in domain modeling under BFO's upper ontology. Under BFO's theory, using classes as the only categorical unit is limiting, hence the introduction of types. All types are Rigid, and a candidate type being assigned by a modeler as Non-Rigid reveals that addition work is required by the modeler for their ontology to be BFO-compliant.

[13] **Has_Dependent**(*A*,*B*) means for every instance of *x* of *A* at *t* there is some instance of *y* of *B* at *t* such that *y* depends on *x* at *t*.

One of the major challenges of integrating BFO with OntoClean is that both theories include informal and formal aspects. For instance, the document titled the BFO manual on the BFO website (http://www.ifomis.org/bfo) explicates all the types of BFO 1.1's ontology, but it does not include any axioms. There are many other papers by the originator of BFO, Barry Smith, that do include axioms, but it is not clear if these papers are considered a part of BFO or not. With a release of BFO 2.0 upcoming, it was also important not to confuse more experimental aspects of 2.0 with what is documented as BFO 1.1. These challenges were mitigated, for the most part, by personal communication with Barry Smith.

On the side of OntoClean, the formulations of Rigidity have evolved through the various papers published by Guarino and Welty. In more recent publications [10] the axioms are omitted. We surmise that this approach is taken to make OntoClean more accessible to laypeople while also minimizing time spent on various logical issues which whose resolution may not offer practical value. We offer the current work formally, with informal explication of our integrated theory, while also providing an implementation based on a decision tree that can serve as a training tool.

While testing the Wizard plugin it became more apparent that the Wizard forces a modeler to think about aspects of the class he is modeling that he may not have considered when first introducing the class. We think this is one of the values of the Wizard–a class will not be added to the ontology until the modeler undergoes the necessary thinking process needed for the class to be considered an addition to a BFO-compliant ontology. The Wizard provides the sort of evaluation that, ideally, an experienced an ontologist performs at an intuitive level when adding classes to an ontology. The Wizard makes this exercise explicit and directed.

References

[1] A.P. Seyed and S. C. Shapiro, Applying Rigidity to Standardizing OBO Foundry Candidate Ontologies, Proceedings of the International Conference on Biomedical Ontologies (ICBO) (2011), 175–181.

[2] H. Knublauch, R.W. Fergerson, N.F. Noy, and M.A. Musen, The Protege OWL Plugin: An Open Development Environment for Semantic Web Applications. In S.A. McIlraith, D. Plexousakis, Third International Semantic Web Conference, New York. Springer Verlag, (2004), 229–243.

[3] C. Welty and W. Andersen, Towards OntoClean 2.0: A framework for Rigidity. *Applied Ontology*, 1(1), IOS Press, Amsterdam (2005), 107–116.

[4] W. Andersen and C. Menzel, Modal Rigidity in the Ontoclean Methodology. *Formal Ontology in Information Systems*. Editors: A. C. Varzi and L. Vieu, IOS Press, Amsterdam, (2004), 119–127.

[5] M. Carrara. Identity and Modality in OntoClean, *Applied Ontology* 1(1), IOS Press, Amsterdam, (2004), 128–139.

[6] B. Smith, The Logic of Biological Classification and the Foundations of Biomedical Ontology. Handbook on Ontologies: Invited Papers from the 10th International Conference on Logic, Methodology and Philosophy of Science. Elsevier-North-Holland, (2003).

[7] B. Smith and W. Ceusters, Ontological realism: A methodology for coordinated evolution of scientific ontologies. *Applied Ontology* 5(3-4), (2010), 139-188.

[8] A. Rector. Modularisation of Domain Ontologies Implemented in Description Logics and related formalisms including OWL, In *Third International Conference on Knowledge Capture*, 23, (2003), 25.

[9] A. Rector, N. Drummond, M. Horridge, J. Rodgers, H. Knublauch, R. Stevens, H. Wang, and C. Wroe. OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns, Springer, (2004), 63–81.

[10] N. Guarino and C. A. Welty, An Overview of OntoClean, *Handbook on Ontologies*, (Eds.) S. Staab and R. Studer, Springer Verlag, Berlin, (2004), 151–159.