Konzeption und Entwicklung eines Systems zur Bewertung und Beurteilung von OWL Modellen im Rahmen des intelligenten tutoriellen Systems eTutor

Diplomarbeit

Zur Erlangung des akademischen Grades
„Magister der Sozial- und Wirtschaftswissenschaften“
(Mag.rer.soc.oec.)
in der Studienrichtung Witschaftsinformatik

Eingereicht an der Johannes Kepler Universität Linz
Institut für Wirtschaftsinformatik – Data & Knowledge Engineering

Betreuer: o.Univ.-Prof. Dr. Michael Schrefl
Mitbetreuer: Mag. Christian Eichinger und Dr. Michael Karlinger

Verfasst von Florian Gruber
Linz, am 25. April 2012

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe. Die vorliegende Diplomarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Kurzfassung

Das Institut für Data & Knowledge Engineering der JKU setzt in der Lehre das *Intelligente Tutorielle System* (ITS) eTutor ein. Dieses System verfügt über Expertenmodule, die von Studierenden ausgearbeitete Übungen zu bestimmten Themengebieten automatisiert korrigieren und bewerten können. Für das in der Lehre behandelte Gebiet der Ontologiemodellierung existiert noch kein solches Modul, weshalb die Übungen händisch korrigiert werden müssen. Um dieses Problem zu beheben wurde in der vorliegenden Arbeit ein derartiges Expertenmodul entwickelt.

Nach einem Vergleich bestehender Ansätze wurde ein neues Konzept entwickelt, das Methoden zum Abgleich von Ontologien mit Erfahrungen der Universität Manchester beim Lehren der *Web Ontology Language* (OWL) kombiniert. Dadurch können Probleme der Studenten mit dem in der Angabe beschriebenen Sachverhalt, der Prädikatenlogik und einige speziell auf OWL bezogene Probleme identifiziert werden.

Die in der vom Studenten als Lösung abgegebenen Ontologie enthaltenen Konzepte werden unter anderem durch Vergleich mit einer Musterlösung analysiert. Aus dabei gefundenen Merkmalen wird auf Fehlertypen geschlossen, die den obigen Problemen zugeordnet werden können. Diese Schlussfolgerungen werden den Studierenden mitgeteilt und erlauben diesen eine Verbesserung ihrer Lösung.

Auf Basis dieses neuen Ansatzes wurde ein in Java programmiertes Expertenmodul entwickelt, das sowohl im ITS eTutor als auch eigenständig, eingesetzt werden kann, beispielsweise im Ontologieeditor Protégé. Es ist modular aufgebaut und kann einfach auf alle Arten von Entities in OWL 2 erweitert werden.

Abstract

The Institute for Data & Knowledge Engineering at the JKU uses the *Intelligent Tutoring System* (ITS) eTutor to enrich their teaching. This system uses expert modules to grade assignments automatically, with each module covering only a small area of expertise. No such module exists for the field of ontology design, so the assignments handed in by the students still have to be graded manually. Solving this problem is the purpose of this thesis.

After comparing different approaches a new concept has been developed, which combines methods for aligning ontologies with findings from teaching OWL-DL at the University of Manchester. This enables the detection of mistakes made by students regarding their instructions, formal logic and OWL-specific issues.

The concepts defined in the submitted ontology are being analyzed in the presence of a model solution. Subsequently, conclusions are drawn by combining the results of this analysis. The resulting feedback allows students to improve their submissions.

Based on this concept, an expert module has been implemented in Java. This software can either be used stand-alone or within the ITS eTutor. It uses a modular approach and can easily be extended to detect mistakes in all kinds of entities available in OWL 2.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Codebeispiele

1. Einleitung

1.1. Motivation und Zielsetzung

Das Institut für Data & Knowledge Engineering der JKU setzt zur Unterstützung von Lehrveranstaltungen mit Übungscharakter das eigens dafür entwickelte *Intelligente Tutorielle System* (ITS) eTutor ein. Dieses System verfügt über Expertenmodule, die auf einzelne Themengebiete spezialisiert sind. Da für das in der Lehre behandelte Gebiet der *Ontologiemodellierung* noch kein solches Modul existiert, müssen die von den Studierenden ausgearbeiteten Übungen manuell durch als Tutoren angestellte höhersemestrige Studenten korrigiert werden. Diese Vorgehensweise ist sehr zeitaufwändig, weshalb den Tutoren wenig Möglichkeit für individuelle Betreuung bleibt, während die Studierenden erst nach etwa einer Woche eine Rückmeldung zu der von ihnen abgegeben Übung erhalten, was den Lernprozess nicht optimal unterstützt. Weiters erfolgt die Korrektur durch einen manuellen Vergleich der in der Beschreibungslogik OWL formulierten Abgabe mit einer Musterlösung, weshalb korrekte und logisch äquivalente, aber anders formulierte Lösungen schwer als solche zu erkennen sind.

Daraus ergibt sich als Ziel der vorliegenden Arbeit die Erstellung eines Expertenmoduls für die Ontologiemodellierung in OWL. Idealerweise ermöglicht das Modul eine Analyse der Abgabe auf logischer Ebene und unterstützt den Lernprozess der Studierenden durch eine aussagekräftige Rückmeldung. Da die Lehrveranstaltungen von Angehörigen verschiedener Nationalitäten besucht werden ist eine mehrsprachige Umsetzung von Vorteil.

1.2. Aufbau der Arbeit

Das Einführungskapitel (Kapitel 2) erklärt den Aufbau eines Intelligente Tutorielle System (ITS) idealtypisch und am Beispiel von eTutor, stellt die Beschreibungslogik OWL kurz vor und beschreibt das Szenario, von dem die Arbeit ausgeht. Auf Basis dieser Voraussetzungen stellt Kapitel 3 (Stand der Technik) bereits vorhandene Ansätze zur Lösung des Problems vor und zeigt, warum sie nicht ausreichend sind. Dadurch motiviert wird in Kapitel 4 ein neues Konzept zur Fehlerdiagnose entwickelt. Dazu wird erst der Einfluss einer automatisierten Auswertung auf das Ausgangsszenario genauer betrachtet und dann die Analyse schrittweise aufgebaut. Weiters wird in diesem Kapitel die „Pizza“ Ontologie vorgestellt, anhand derer alle Beispiele erklärt werden. Abschließend wird zum besseren Verständnis eine Beispielanalyse durchgeführt.

Kapitel 5 (Umsetzung) beschäftigt sich mit der Implementierung des im vorigen Kapitel aufgebauten Konzepts als Expertenmodul. In einem ersten Schritt wird die im

vorgestellten Szenario verwendete Software auf eTutor als ITS und Protégé als Ontologieeditor konkretisiert. Anschließend wird das auf OWL-Klassen bezogene Konzept zu einem Framework für die Analyse aller Arten von OWL-Entities erweitert. Nach dem Vorstellen der für die Rückmeldung verwendeten Kommunikationskanäle werden Aufbau, Konfiguration und Integration der erstellten Software detailliert besprochen. Schließlich werden die für die Verwendung nötigen Programme beziehungsweise Programmbibliotheken vorgestellt.

Davon ausgehend zeigt Kapitel 6 mögliche Folgeprojekte zur Verbesserung und Weiterentwicklung des Expertenmoduls und skizziert mögliche Anwendungen außerhalb des ITS eTutor. Am Ende der Arbeit fasst Kapitel 7 das Ergebnis der Arbeit zusammen und zeigt mögliche Weiterentwicklungen. Der Anhang enthält die mit dem Framework analysierten Testfälle und die zugehörigen Analyseergebnisse. In der gesamten Arbeit wurden, soweit verfügbar, die in den deutschen Übersetzungen der Standards [27] [10] verwendeten Begriffe übernommen, sonst wurde auf die englischen Originalbezeichnungen zurückgegriffen. Die Bezeichnungen *OWL-Klasse* und *OWL-Konzept* werden synonym verwendet.

2. Einführung

2.1. Intelligente Tutorielle Systeme (ITS)

2.1.1. Aufbau eines idealtypischen ITS

Die Komponenten eines idealtypischen intelligenten tutoriellen Systems (ITS) sind den Aufgaben eines menschlichen Lehrers nachempfunden. Aus dessen Eigenschaften ergeben sich vier separate Module (Experten-, Studenten-, Unterrichts- und Kommunikationsmodul), wobei die theoretische Trennung in der Praxis nicht immer sinnvoll ist. So werden beispielsweise Experten- und Studentenmodul oft zusammengefasst [26, 18f].

Das *Expertenmodul* repräsentiert das Fachwissen des Lehrenden. Es stellt Aufgaben und kann diese auch lösen, entweder unter Angabe des Lösungswegs (*glass box*) oder ohne Begründung (*black box*). Die gestellten Aufgaben können vom Modul selbst erzeugt oder einer vorbereiteten Aufgabensammlung entnommen werden [26, 19f].

Im *Studentenmodul* wird das für die Thematik relevante Problemlösungsverhalten des Studenten beobachtet, also sein Fachwissen analysiert. Bei dieser Fehlerdiagnose wird die Lösung des Studenten mit der des Expertenmodells verglichen, wobei zufällige und systematische Fehler getrennt werden. Diese Fehlern werden zu Merkmalen verdichtet, aus denen eine möglichst Erfolg versprechende Therapie abgeleitet wird. Die Analyse kann auf zwei Arten erfolgen: *Überlagerungsmodelle* vergleichen das Verhalten von Experte und Student und stellen den Mangel bestimmter Fähigkeiten fest. *Störungsmodelle* gehen darüber hinaus und versuchen, aus dem Verhalten des Studenten Regeln abzuleiten, anhand derer sie den richtigen Lösungsweg verändern. Eine weitere Möglichkeit besteht in der Verwendung von Zwischenergebnissen [26, 22f].

Das *Unterrichtsmodul* steuert die Inhalte der Kommunikation vom Kommunikationsmodul zum Studierenden, beispielsweise die Auswahl und Reihenfolge von Unterrichtsinhalten (das „Curriculum“). Je nach Kenntnisstand kann als *Tutor* oder als *Mentor* unterrichtet werden [26, 23f].

Im Gegensatz dazu entscheidet das *Kommunikationsmodul* über die Form der Interaktion. Wichtige Aspekte sind der natürlichsprachliche Dialog und die pädagogische Wirksamkeit der Kommunikation [26, 25f].

Alle Module hängen sehr eng zusammen, was zu einem komplexen Ablauf einer ITS-Sitzung führt. Abbildung 2.1 zeigt die Interaktion der Module anhand von Kommunikationsinhalten [26, 26f].

2.1.2. eTutor

Das webbasierte ITS eTutor wurde 2005 am Institut für Data & Knowledge Engineering entwickelt und ist aktiv in der Lehre im Einsatz. Bei der Entwicklung wurden die

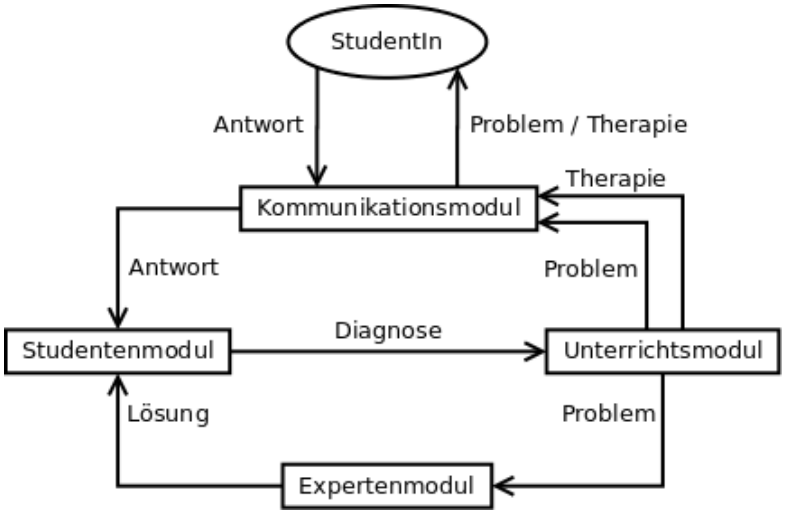

Abbildung 2.1.: Kommunikation zwischen ITS-Modulen (nach [26])

Komponenten eines idealtypischen ITS zu zwei Teilen zusammengefasst. Der eTutor-Kernel entspricht dem Kommunikationsmodul. Momentan wird weder ein Unterrichts- noch ein Studentenmodul eingesetzt [17, 89]. Für die einzelnen Bereiche der Lehre werden in gesonderten Projekten Expertenmodule entwickelt, derzeit sind unter anderem Module zu den Themengebieten Datalog, JDBC, relationale Algebra, relationaler Datenbankentwurf, SQL, konzeptueller Datenbankentwurf [24], und XQuery verfügbar. Das im Rahmen der vorliegenden Arbeit entwickelte Expertenmodul für OWL füllt eine bestehende Lücke.

Die Aufgabe jedes Expertenmoduls beschränkt sich im Vergleich zum idealtypischen ITS auf das Auswerten der Abgabe und Erzeugen der Rückmeldung, wobei mit dem Vergleich von Abgabe und Musterlösung ein ergebnisorientierten Ansatz verfolgt wird [17, 114]. Ein Generieren der Aufgabe ist nicht erforderlich, da alle Aufgabe einem Aufgabenpool entnommen werden können [17, 101].

Somit umfasst ein Expertenmodul folgende Bereiche: Bei der Fehleranalyse wird die vom Studenten abgegebene Lösung als *richtig* oder *falsch* bewertet Die Richtigkeit der Lösung wird optional durch detaillierte Punktevergabe untermauert. Weiters erhält der Student ein individuelles Feedback zu seiner Lösung, um die Qualität seiner Abgaben zu verbessern. [17, 101f].

Das ursprünglich nur aus den Komponenten Web-Browser und eTutor-Server bestehende System wurde 2010 an das an der JKU gebräuchliche Learning Management System Moodle angebunden und in dessen Oberfläche integriert [14].

2.2. Web Ontology Language (OWL)

Die *Web Ontology Language* (OWL) ist ein vom *World Wide Web Consortium* (W3C) herausgegebener Standard zur Definition von Ontologien für das *Semantic Web* [27].

Eine Ontologie ist ein konzeptuelles Modell, das einen Teil der realen Welt beschreibt [11, 359f]. Sie dient als gemeinsames Vokabular, etwa für Produkte und Dienstleistungen oder für den medizinischen Bereich. Dieses Vokabular erleichtert den Informationsaustausch zwischen Menschen oder Computerprogrammen und ermöglicht unter anderem die Analyse und Wiederverwendung von Domänenwissen [29, 1f][32, 154].

Diese Arbeit verwendet die *OWL 2 Web Ontology Language* [36], die folgende syntaktische Bausteine zur Verfügung stellt [4][27]: Klassen (*Class Expressions*), Eigenschaften (*Properties*), Individuen (*Individuals*), Datenbereiche (*Data Ranges*), Axiome (*Axioms*), Deklarationen (*Declarations*), Anmerkungen (*Annotations*) und die Einbindung anderer Ontologien (*Ontologies*).

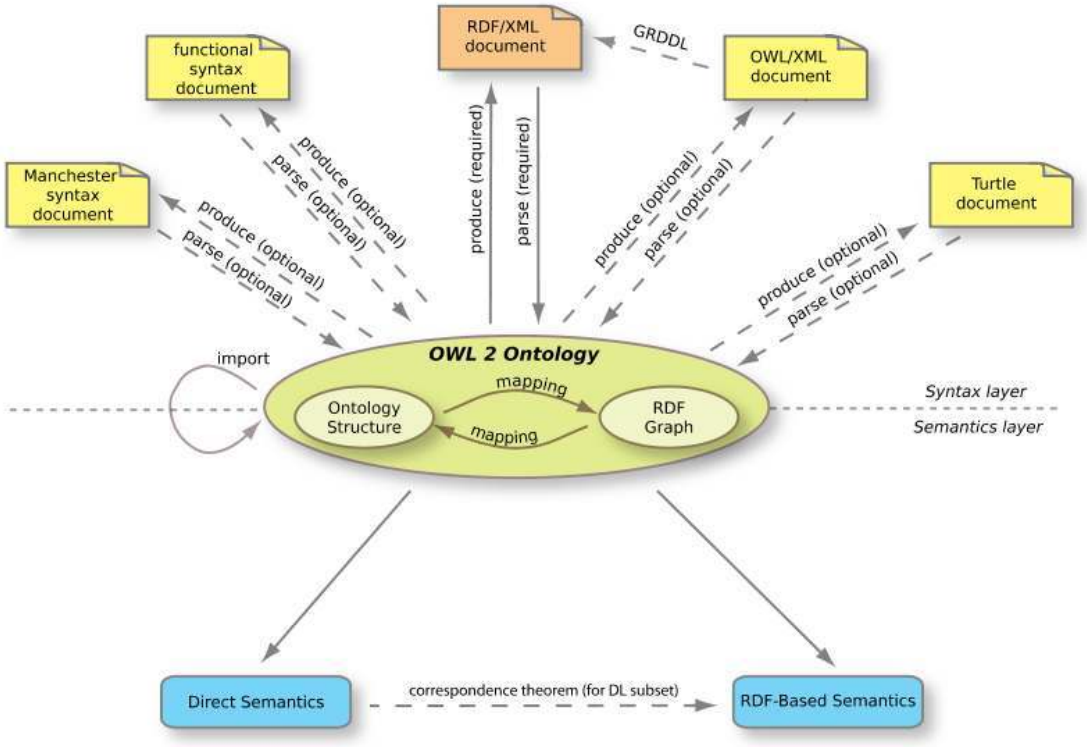

Abbildung 2.2.: Struktur von OWL 2 (nach [36])

Da OWL eine Logiksprache ist, können Computerprogramme (sogenannte *OWL Reasoner*) Schlussfolgerungen innerhalb der Ontologie ziehen, etwa über die Beziehung der Klassen zueinander und ob Widersprüche bestehen. Ein Beispiel für einen solchen Widerspruch sind zwei Klassen, deren Definitionen sich gegenseitig ausschließen. Um die Eindeutigkeit jeder Ontologie sicherzustellen wird sie mittels eines IRI (*International Resource Identifier*) definiert [2], der meist auch als Basis der Namens aller in der Ontologie beschriebenen Entities verwendet wird.

Abbildung 2.2 zeigt die vielen Arten des Zugriffs auf eine OWL-Ontologie, sowohl auf syntaktischer als auf semantischer Ebene. Für diese Arbeit besonders relevant ist, dass

die Wahl einer bestimmten Syntax keine Auswirkungen auf die semantische Ebene und damit auf die Ontologie hat. Der OWL-Standard [2] sieht dafür folgende Syntaxen vor: *Functional-Style* vor allem für Spezifikationen, *RDF/XML* für die einfache Verwendung bestehender Formate, *Turtle* als vereinfachte Schreibweise für RDF [1], die *Manchester Syntax* zur besseren Lesbarkeit für Personen ohne Logik-Vorbildung und *OWL/XML* als Syntax mit einem für OWL definierten XML-Schema. Im weiteren Verlauf der Arbeit wird die Manchester Syntax verwendet.

Die folgenden Codebeispiele stellen die Tatsache „Eine Mutter ist eine Frau, die gleichzeitig ein Elternteil ist“ (logisch: Mother $\equiv$ Woman $\sqcup$ Parent) in den verschiedenen Syntaxen dar. Dieses Beispiel wurde dem OWL 2 Primer [2] entnommen.

```
EquivalentClasses(
  :Mother
  ObjectIntersectionOf( :Woman :Parent )
)
```

Codebeispiel 2.1: Functional-Style Syntax

```
<owl:Class rdf:about="Mother">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="Woman"/>
        <owl:Class rdf:about="Parent"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

Codebeispiel 2.2: RDF/XML Syntax

```
:Mother owl:equivalentClass [
  rdf:type owl:Class ;
  owl:intersectionOf ( :Woman :Parent )
] .
```

Codebeispiel 2.3: Turtle Syntax

```
Class: Mother
  EquivalentTo: Woman and Parent
```

Codebeispiel 2.4: Manchester Syntax

```
<EquivalentClasses>
  <Class IRI="Mother"/>
  <ObjectIntersectionOf>
    <Class IRI="Woman"/>
    <Class IRI="Parent"/>
  </ObjectIntersectionOf>
</EquivalentClasses>
```

Codebeispiel 2.5: OWL/XML Syntax

2.3. Szenario: Übung Ontologiemodellierung

Als Teil des Bachelorstudiums Wirtschaftsinformatik werden im Bereich Data & Knowledge Engineering unter anderem Grundlagen von Ontologien im Semantic Web am Beispiel von OWL vermittelt [6].

Dabei werden Übungsaufgaben gestellt, bei denen Ontologien zu definieren und als Datei abzugeben sind, wobei die Ontologien oft in Textverarbeitungsprogrammen unter Verwendung von logischen Symbolen erstellt werden. Die Abgabe erfolgt durch Hochladen in das ITS eTutor, das die Abgaben zusammen mit einer Musterlösung elektronisch an vom Institut beschäftigte Tutoren zur Korrektur weiterleitet. Die Tutoren vergleichen die Lösungen händisch und versuchen die Gedankengänge beim Modellieren nachzuvollziehen, die Bewertung erfolgt nach einem von der Lehrveranstaltungsleitung definierten Punkteschlüssel. Ungefähr eine Woche nach Abgabe erhalten die Studenten die von ihnen abgegebene Datei mit den Kommentaren der Tutoren. Auftretende Fragen können sie mit einem Betreuer (Tutor oder Übungsleitung) besprechen.

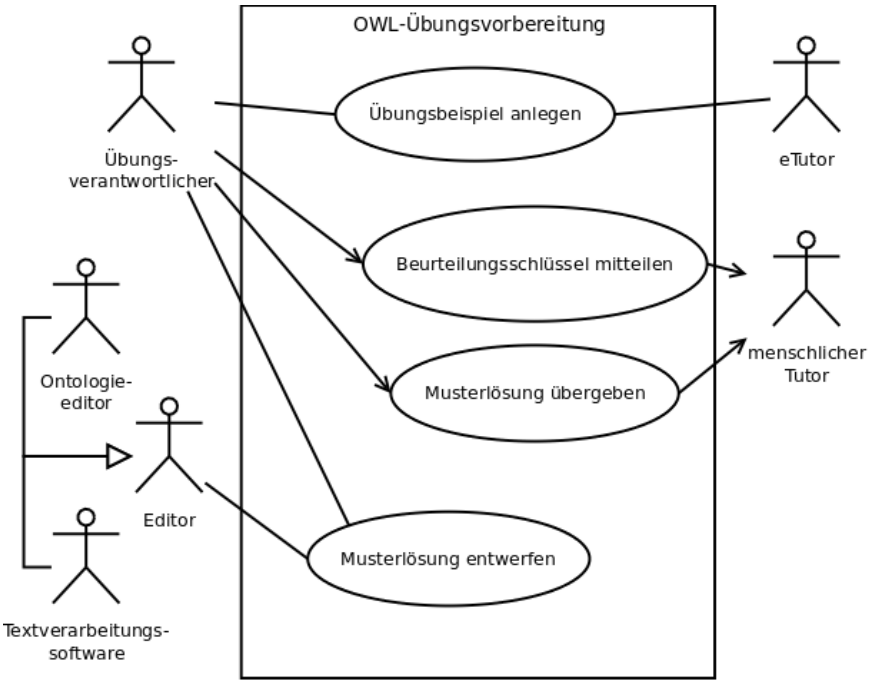

Abbildung 2.3.: Anwendungsfall *Vorbereitung der Übung*

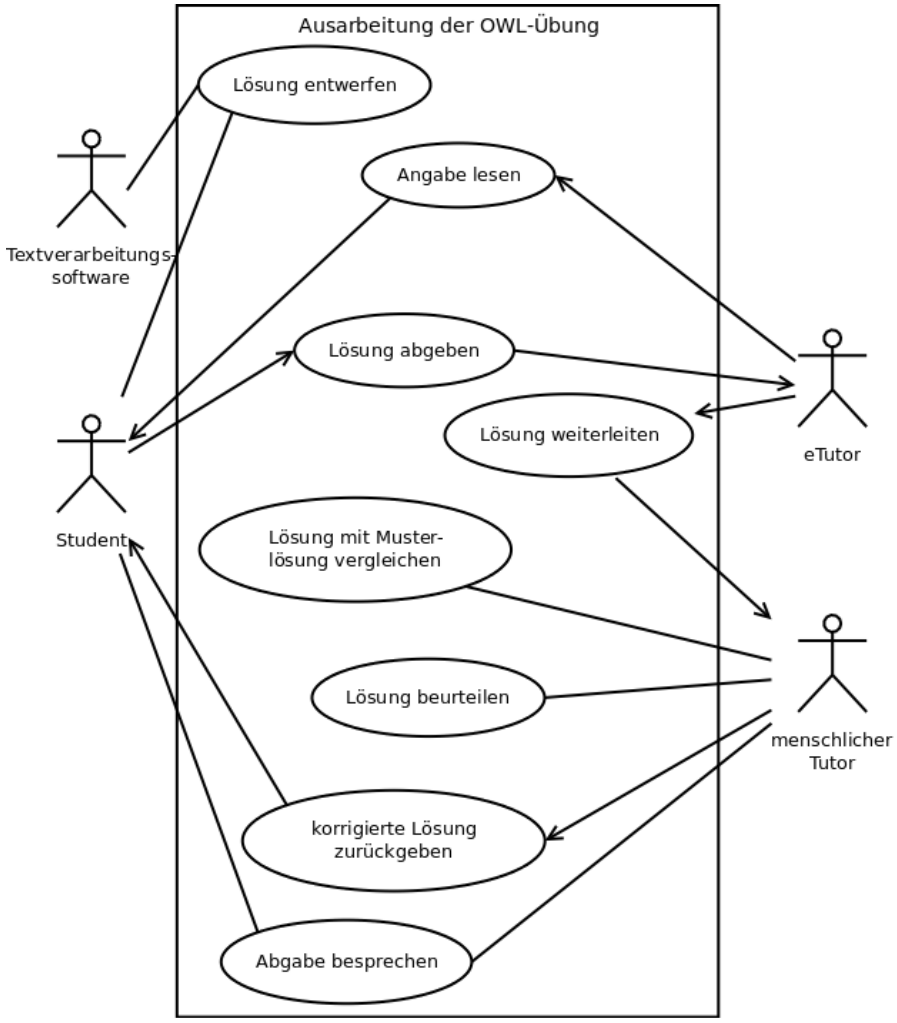

Abbildung 2.4.: Anwendungsfall *Ausarbeitung der Übung*

Die mit der Übung verbundenen Abläufe können zeitlich in folgende Anwendungsfälle geteilt werden:

Als Vorbereitung für die Übung zur Ontologiemodellierung legt ein Übungsverantwortlicher im ITS eTutor die Übungsbeispiele an, teilt den Tutoren den Beurteilungsschlüssel mit und übergibt ihnen die Musterlösung. Diese wurde vorher von ihm unter Zuhilfenahme eines Ontologieeditors oder einer Textverarbeitungssoftware erstellt. Abbildung 2.3 zeigt diese Schritte als Anwendungsfalldiagramm.

Bei der Ausarbeitung der Übung (vgl. Abbildung 2.4) liest jeder Studierende die Angabe des Übungsbeispiels, entwirft in einem Textverarbeitungsprogramm eine zur Angabe passende Lösung und gibt diese als Datei im eTutor ab. Das ITS leitet die Lösung an einen Tutor weiter, der die Abgabe mit der Musterlösung vergleicht und korrigiert. Die Abgabe wird unter Verwendung des Beurteilungsschlüssels beurteilt und dem Studenten zurück gegeben. Falls gewünscht kann der Student die Korrektur mit dem Tutor besprechen.

Der eben beschriebene Ablauf zeigt folgende *Verbesserungsmöglichkeiten*: Beim Ontologieentwurf in einem Textverarbeitungsprogramm wird die Eingabe weder syntaktischen noch logischen Prüfungen unterzogen. Weiters ist die Arbeitszeit der Tutoren begrenzt, ein verringerter Zeitaufwand bei der Korrektur bedeutet daher mehr Zeit für die Betreuung von Studenten im Tutorium.

Mit der langen Zeitspanne zwischen Abgabe und Rückmeldung verringert sich das Interesse der Studenten, sich mit der korrigierten Abgabe auseinanderzusetzen, da sie bereits mit dem Lösen anderer Aufgaben beschäftigt sind.

Die Qualität der Korrektur hängt stark von der Fähigkeit des jeweiligen Tutors ab, von der Musterlösung abweichende Lösungen zu verstehen. Syntaktisch von der Musterlösung abweichende Abgaben werden also potentiell benachteiligt, auch wenn sie eine logisch äquivalente Lösung darstellen. Außerdem ist die Bewertung möglicherweise ungerecht, da sie von verschiedenen Personen durchgeführt wird. Aus diesen Verbesserungsvorschlägen ergibt sich die Forderung nach einem Expertenmodul für OWL.

3. Stand der Technik

Das geforderte Expertenmodul soll Fehler in der abgegebenen Lösung erkennen und anschließend mittels einer geeigneten Rückmeldung erklären. Daher können folgende Aspekte der Problemlösung identifiziert werden: Erstens müssen Fehler in der Abgabe gefunden werden. Zweitens müssen fehlerhafte Teile der Abgabe den Studierenden verständlich präsentiert werden, um einen Lerneffekt zu erzielen. Schließlich müssen alle Teile automatisch ablaufen, um die Tutoren maximal zu entlasten.

Bevor die einzelnen Ansätze vorgestellt werden muss geklärt werden, welche Fehler in der abgegebenen Ontologie auftreten können. Danach werden die betrachteten Verfahren in zwei Gruppen geteilt: Solche, die zwei Ontologien vergleichen und andere, die eine einzelne Ontologie analysieren. Am Ende dieses Kapitels werden die Ansätze unter Berücksichtigung der Anforderungen verglichen.

3.1. In der Abgabe zu erwartende Fehler

Eine fehlerhafte Abgabe kann auf mehreren Ebenen fehlerhaft sein:

Syntaktische Fehler in der Abgabe passieren leicht beim händischen Erstellen von Ontologien, besonders wenn eine komplizierte Syntax wie RDF oder XML verwendet wird. Solche Fehler können durch Einsatz eines OWL-Editors erkannt und behoben werden [21, 2]. Syntaktische Fehler verhindern die Verarbeitung durch einen Reasoner.

Eine *unerfüllbare Klasse* entsteht durch widersprüchliche Definition der Klasse. Eine solcher Widerspruch kann auch beabsichtigt sein, um die Unmöglichkeit einer Klasse zu demonstrieren [30, 634]. Das Finden solcher Klassen ist eine der Funktionen, die jeder gängige Reasoner beherrscht [32, 165].

Eine *inkonsistente Ontologie* entsteht, wenn sich Axiome der Ontologie widersprechen. Inkonsistenz wird vom Reasoner möglichst früh festgestellt. Sie erschwert die Weiterverarbeitung im Reasoner, da aus einem Widerspruch jede Aussage abgeleitet werden kann [30, 634].

Unerwünschte Subsumption entsteht durch Modellieren eines falschen Sachverhalts. Diese Fehler auf semantischer Ebene können vom Reasoner nicht mehr gefunden werden [30, 635]. Mögliche Auswege bieten der Vergleich mit einer Ontologie, die den Sachverhalt korrekt modelliert oder auch, in manchen Fällen, eine Analyse der Axiome.

3.2. Vergleich von Ontologien

Zur Kategorisierung der vorgestellten Ansätze wird die Terminologie des *Ontology Mapping* [12, 34f]. verwendet. Dieses Forschungsgebiet beschäftigt sich mit der Interopera-

bilität von Ontologien, die ähnliche oder überlappende Gebiete beschreiben. Der dazu benötigte Abgleich der einzelnen Ontologien umfasst eine Zuordnung der enthaltenen Daten und eine Gewichtung ihrer Übereinstimmung. Dabei kann zwischen verschiedenen Arten von Ontology Mapping unterschieden werden:

Bei *Ontology Merging* werden mehrere verschiedene Ontologien eines Themenbereichs zu einer einzigen verschmolzen. *Ontology Alignment* verknüpft Ontologien überlappender Domänen, die Ontologien bleiben jedoch getrennt. Im Fall von *Ontology Integration* wiederum entsteht aus mehreren Ontologien unterschiedlicher Themengebiete eine einzige Ontologie. Für diese Arbeit sind nur *Ontology Merging* und *Ontology Alignment* interessant, da Abgabe und Musterlösung das selbe Themengebiet beschreiben.

Ein wesentlicher Unterschied zwischen Ontology Mapping und dem Einsatz in der Lehre ist das Ziel des Vergleichs: In der Lehre sollen gefundene Unterschiede den Studierenden erklärt werden, bei Ontology Mapping ist nur der Grad der Übereinstimmung interessant [34]. Daher ist der Einsatz von Werkzeugen des Ontology Mapping für den Vergleich von Musterlösung und Abgabe nicht sinnvoll, einzelne Aspekte des Vergleichs können jedoch übernommen werden.

3.2.1. Ontology Alignment in OWL-Lite

Dieser Ansatz beschreibt einen Algorithmus, der die Ähnlichkeit von zwei in OWL-Lite formulierten Ontologien berechnet. OWL-Lite ist eine Variante der ersten Version von OWL mit geringem Funktionsumfang, ihr Vorteil liegt in der einfacheren Berechenbarkeit gegenüber mächtigeren Varianten [27]. Die Ähnlichkeit von Ontologien basiert auf der Ähnlichkeit ihrer Bestandteile, der OWL-Entities.

Das Ähnlichkeitsmaß von Entity-Paaren wird als Summe folgender gewichteter Faktoren berechnet: *Terminologische Ähnlichkeit* betrifft die Bezeichnungen der Entities, wobei die Zeichenketten entweder direkt oder über Synonyme verglichen werden. Der *Vergleich der internen Struktur* bezieht sich auf die Axiome der Entities, beim *Vergleich der externen Struktur* hingegen wird die Position innerhalb einer Taxonomie betrachtet. Ein weiterer Vergleich analysiert die *Ausdehnung der Entities*, also die ihnen zugeordneten Klassen. Der *semantische Vergleich* betrifft die Interpretation der Entities [13, 333ff].

Das Ergebnis dieses Algorithmus ist die quantifizierte Ähnlichkeit der in den verglichenen Ontologien enthaltenen Entities. Idealerweise wird damit eine ausreichend gute Abbildung (*Mapping*) von einer Ontologie auf eine zweite erreicht.

3.2.2. OWLdiff

Die Aufgabe von *OWLdiff*[1] ist der Vergleich zweier Ontologien. Typisches Einsatzgebiet ist der Vergleich der aktuellen Version einer Ontologie mit einer Vorgängerversion, etwa in Zusammenarbeit mit einer Versionsverwaltung. OWLdiff kann alleine eingesetzt werden oder als Erweiterung im Ontologieeditor Protégé (vgl. Seite 74). Abbildung 3.1 zeigt den Einsatz dieser Erweiterung an einem einfachen Beispiel.

[1] `http://krizik.felk.cvut.cz/km/owldiff/`

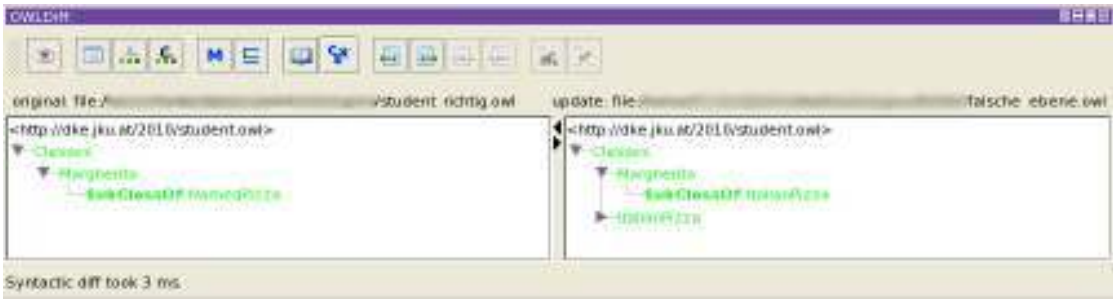

Abbildung 3.1.: *OWLdiff*-Plugin in Protégé

Der Vergleich erfolgt auf semantischer Ebene, als Reasoner wird Pellet verwendet (vgl. Seite 74). Dabei stehen zwei Algorithmen zur Auswahl: Ein einfacher Algorithmus, der nur Änderungen in der Klassenhierarchie erkennt und der komplexere Algorithmus *CEX*, der jedoch auf das OWL 2 Fragment $\mathcal{EL}$ [3] eingeschränkt ist. CEX vergleicht die in Axiomen der Ontologien verwendeten Klassen und liefert für jede Ontologie eine Liste von Klassen, die in der anderen Ontologie nicht verwendet werden [25].

3.3. Analyse einer einzigen Ontologie

In den folgenden Ansätzen wird eine einzelne Ontologie für sich analysiert, also ohne Vergleich mit einer korrekten Lösung.

3.3.1. Begründung der Klassifikation in Pellet

Der Reasoner Pellet verfügt über einen Modus, in dem die Klassifikation jedes Konzepts einer Ontologie begründet wird [30]. Diese *Erklärung* eines Konzeptes besteht aus den Axiomen, die Einfluss auf die Klassifikation des Konzepts hatten. Beispiel 3.1 zeigt einen Teil der Begründung der Musterlösung. Die einfachste Form einer Begründung ist die Wiederholung der Definition, zu sehen unter anderem in den ersten vier Zeilen des Beispiels. Im Gegensatz dazu verläuft die Begründung, warum eine Pizza Margherita vegetarisch ist, in mehreren Schritten (Zeilen 21–35) und führt dabei sowohl die Definition einer vegetarischen Pizza als auch die der verwendeten Beläge an.

```
Axiom: Pizza subClassOf Food                                    1

Explanation:                                                    3
    Pizza subClassOf Food                                       4


Axiom: PizzaTopping subClassOf Food                             7

Explanation:                                                    9
    PizzaTopping subClassOf Food                                10
```

```
Axiom: VegetarianPizza subClassOf Pizza                                    13

Explanation:                                                               15
    VegetarianPizza equivalentTo Pizza                                     16
        and not hasPizzaTopping some FishTopping                           17
        and not hasPizzaTopping some MeatTopping                           18


Axiom: Margherita subClassOf VegetarianPizza                               21

Explanation:                                                               23
    Margherita subClassOf Pizza                                            24
    Margherita subClassOf hasPizzaTopping only MozzarellaTopping           25
                                           or TomatoTopping                26
    VegetarianPizza equivalentTo Pizza                                     27
        and not hasPizzaTopping some FishTopping                           28
        and not hasPizzaTopping some MeatTopping                           29
    DisjointClasses(CheeseTopping                                          30
                    FishTopping                                            31
                    FruitTopping                                           32
                    MeatTopping                                            33
                    TomatoTopping)                                         34
    MozzarellaTopping subClassOf CheeseTopping                             35


Axiom: CheeseTopping subClassOf PizzaTopping                               38

Explanation:                                                               40
    CheeseTopping subClassOf PizzaTopping                                  41
```

Codebeispiel 3.1: Begründung der Musterlösung (Auszug)

Die Erklärung der Klassifikation soll dem Autor der Ontologie die Folgen seiner Modellierungsentscheidungen sichtbar machen und ihn zur Korrektur ungewollter Folgen animieren. Dazu kann Pellet kann auch inkonsistente Ontologien erklären.

Ein wesentlicher Nachteil dieses Ansatzes ist der Status seiner Implementierung: Das Erklären von Klassenhierarchien ist zwar im Quellcode des Reasoners Pellet enthalten, aber kein Teil dessen API. Daher müsste das Expertenmodul direkt auf interne Pellet-Klassen zugreifen, die sich in künftigen Versionen ändern können. Eine solche Verwendung wäre aufgrund des Wartungsaufwands nicht vertretbar, der Ansatz ist daher nicht geeignet.

3.3.2. Übliche Fehler beim Erlernen von OWL-DL

An der Universität Manchester wurden über viele Lehrveranstaltungen hinweg beim Erlernen von Beschreibungslogiken auftretende Schwierigkeiten beobachtet. Aufgrund

dieser Beobachtungen wurden die häufigsten Anfängerfehler beim Definieren von OWL-Klassen gesammelt und erklärt [31].

Neulinge haben beim Lernen von OWL-DL typischerweise folgende Schwierigkeiten: (1) Implizite Informationen werden nicht explizit formuliert und stehen daher dem Reasoner nicht zur Verfügung. (2) Der Allquantor wird irrtümlicherweise statt dem Existenzquantor verwendet. (3) Die *Open World Assumption* (auf Seite 33 genauer beschrieben) wird nicht verstanden. (4) Einschränkungen von Werte- und Zielbereich werden nicht beachtet. (5) Bedingungen werden versehentlich so formuliert, dass sie von einer leeren Menge erfüllt werden. (6) Der Unterschied zwischen definierten und primitiven Klassen wird nicht verstanden. (7) Logische Ausdrücke werden falsch verstanden. (8) Die mögliche Überlappung von Klassen wird nicht beachtet. (9) Schwierigkeiten beim Verständnis der Bedeutung von Subklassen.

Diese Fehler werden anhand einer Ontologie über Pizzas demonstriert, beispielsweise durch verschiedene Definitionen einer vegetarischen Pizza. Die Arbeit beschreibt dabei Algorithmen, mit denen einige der Fehler durch Analyse der Axiome gefunden werden können.

Durch Umsetzung dieser Algorithmen im Rahmen des neuen Expertenmoduls könnten viele Fehler rein durch eine Analyse der Abgabe erkannt werden, ohne dass eine Musterlösung benötigt würde.

3.4. Vergleich der beschriebenen Ansätze

Wie in Tabelle 3.1 zu erkennen, behandeln die verschiedenen Ansätze sehr verschiedene Probleme. Für den Einsatz im Expertenmodul müssen Fehler auf möglichst jeder der eingangs beschriebenen Ebenen gefunden werden. Fehler auf syntaktischer Ebene führen bei allen automatisierten Ansätzen vermutlich zum Abbruch der Verarbeitung. Diese könnte erkannt und in die Auswertung einbezogen werden.

Unerfüllbare Klassen werden durch Klassifikation mittels Reasoner erkannt. Die vergleichenden Ansätze werten sie aber nur dann als Fehler, wenn sie in nur einer der Ontologien unerfüllbar sind. Der empirische Ansatz der Universität Manchester wiederum beschreibt Denkfehler, die zu unerfüllbaren Klassen führen.

In den Arbeiten zu den vergleichenden Ansätzen konnte keine Aussage bezüglich inkonsistenter Ontologien gefunden werden. Deshalb wird angenommen, dass diese nicht erkannt werden. Im Gegensatz dazu liefert die Begründung der Klassifikation durch Pellet sehr genaue Hinweise zum Grund der Inkonsistenz. Der Ansatz der Universität Manchester erwähnt Inkonsistenz nur als Merkmal, das auf widersprüchliche Fakten hinweist.

Fehler auf semantischer Ebene wie unerwünschte Klassifikation können von Pellet nicht erklärt werden, da die Absicht der Modellierung nicht bekannt ist. Die empirisch erhobenen Anfängerfehler beinhalten sowohl textuelle Begründungen als auch Richtlinien, um semantische Fehler in den Axiomen zu entdecken. Dabei handelt es sich aber immer um Erfahrungswerte, da diese Muster auch die Folge bewusster Modellierungsentscheidungen sein können.

Je nach Aufbau der Ontologien müssen Fehler nicht unbedingt zu Änderungen der Klassenhierarchie führen, weshalb CEX dem einfachen Algorithmus in OWLdiff vorzuziehen ist.

Die Begründung der Klassifikation durch Pellet erhöht das Verständnis der Klassifikation, ist aber auf die Analyse einer einzelnen Ontologie beschränkt.

Die an der Universität Manchester gefundenen typischen Fehler beim Erlernen von OWL-DL beschreiben meist die Denkweise und können nur teilweise auf Klassen einer beliebigen Ontologie angewandt werden.

Aus der Tabelle 3.1 geht hervor, dass kein Ansatz alleine Fehler auf allen Ebenen finden kann. Für eine mögliche Kombination mehrerer Ansätze sind die Beschreibungslogiken wichtig, die jeder Ansatz unterstützt. Die vergleichenden Ansätze erlauben nur die einfachsten Varianten von OWL, während die beiden anderen aussagekräftigere Logiken unterstützen.

Für einen erfolgreichen Einsatz in der Lehre ist das Generieren einer möglichst guten Rückmeldung wichtig. Ein durch Ontology Alignment berechnetes Ähnlichkeitsmaß besteht zwar aus einzelnen Faktoren, erklärt aber die Differenz zwischen den Ontologien nicht. Der in OWLdiff verwendete Algorithmus CEX erzeugt bereits Listen mit Konzepten, die den Studenten vorgelegt werden könnten. Der Ansatz der Universität Manchester geht jedoch noch einen Schritt weiter, da er Vermutungen über das Verständnis von OWL trifft und Lerndefizite aufzeigt. Die Begründung der Klassifikation wiederum zeigt die Zusammenhänge der einzelnen Axiome und macht die Entscheidungen des Reasoners nachvollziehbar.

Ansatz (Name verkürzt)	Logik	findet	Rückmeldung	automatisiert
Ontology Alignment	OWL-Lite	S, U	Ähnlichkeitsmaß	ja
OWLdiff	OWL 2 $\mathcal{EL}$	S, U	Liste von Konzepten	ja
Begründung der Klassifikation	OWL-DL, OWL 2	S, K, I	Erklärung der Konzepte	ja
Übliche Fehler von Neulingen	OWL-DL	K, U	Verständnisprobleme	nein

S ... Syntaktische Fehler
K ... Unerfüllbare Klassen
I ... Inkonsistente Ontologie
U ... Unerwünschte Klassifikation, semantische Fehler

Tabelle 3.1.: Eigenschaften der behandelten Ansätze

4. Konzept

Keiner der im vorigen Kapitel vorgestellten Ansätze erklärt Fehler auf allen geforderten Ebenen, daher ist die Entwicklung eines neuen Konzepts erforderlich. Durch Synthese mehrerer Ansätze können Fehler aller Ebenen gefunden und die Studenten auf Verständnisprobleme hingewiesen werden.

Um eine möglichst gute Rückmeldung zu erreichen ist die Integration von Fehlerdiagnosemodellen in das Expertenmodul notwendig, da das ITS eTutor kein Studentenmodell enthält (vgl. 2.1.2).

4.1. Erkennung von Fehlern beim Erlernen der Ontologiemodellierung

Der neu entwickelte Ansatz kombiniert die empirische Untersuchung der Universität Manchester (vgl. 3.3.2) mit Methoden des Ontology Alignment (vgl. 3.2.1). Der Fokus liegt dabei auf der semantischen Ebene, da die Studierenden bereits vom Ontologieeditor auf syntaktische Fehler und inkonsistente Ontologien hingewiesen werden können. Zusätzlich werden diese beiden Fehlerebenen am Beginn der Verarbeitung im Expertenmodul überprüft.

Die restlichen Ebenen werden mit Hilfe einer Musterlösung analysiert. Unerfüllbare Klassen werden dabei nicht als Fehler gesehen, wenn sie auch in der Musterlösung unerfüllbar sind. Dadurch können Übungsbeispiele formuliert werden, bei denen sowohl erfüllbare als auch unerfüllbare Klassen definiert werden müssen.

Fehler auf semantischer Ebene werden durch Analyse sowohl der einzelnen OWL-Klassen als auch von Klassenpaaren gefunden. Diese Paare werden aus je einer Klasse der Abgabe und einer ähnlichen Klasse der Musterlösung gebildet. Die Ähnlichkeit wird in Anlehnung an Faktoren des Ontology Alignment bestimmt, und zwar hinsichtlich der Bezeichnungen und Axiome der Konzepte sowie deren Position in der klassifizierten Hierarchie.

Diese Faktoren wurden in algorithmisch leicht zu beschreibende Symptome aufgespalten, um die im Ansatz der Universität Manchester beschriebenen Probleme beim Verständnis von OWL-DL zu finden. Abbildung 4.1 zeigt den Zusammenhang des neuen Konzepts mit den beiden Ansätzen. Die in der Abbildung verwendeten Symptome und Fehlertypen sind beispielhaft und werden später genauer vorgestellt. Die meisten Fehlertypen entsprechen von der Universität Manchester gefundenen Verständnisproblemen, der Rest basiert auf Erfahrungen des Instituts für Data & Knowledge Engineering an der JKU.

Die für den Vergleich benötigte Expertenlösung (Musterlösung) für die Domäne des

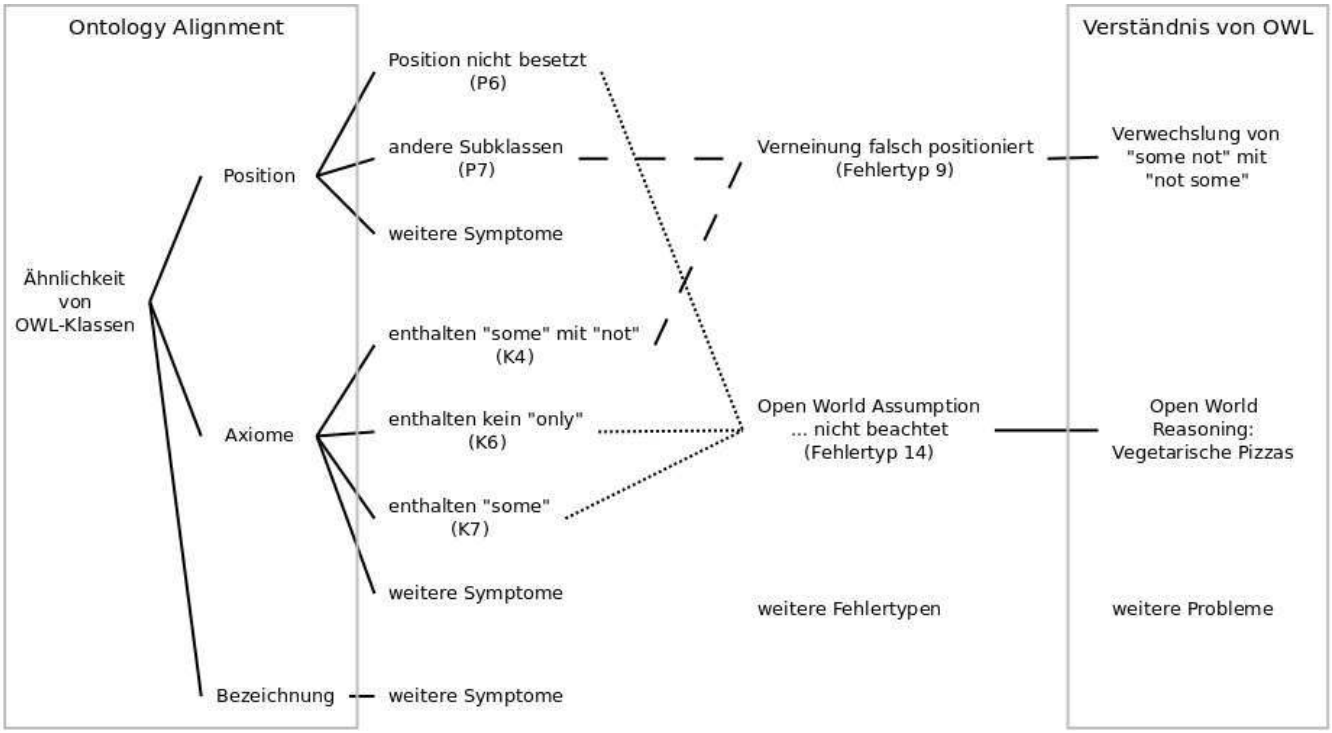

Abbildung 4.1.: Einfluss anderer Ansätze

Aufgabengebiets wird nicht vom Expertenmodul erzeugt, sondern liegt in einer Lösungssammlung vor. Es handelt sich also um eine *statische Lösung* [26, 103]. Die Verbindung zwischen Abgabe und Musterlösung wird über eine Basisontologie hergestellt, deren Vokabular von beiden Lösungen verwendet wird. Dieser Zusammenhang erlaubt es, Klassen verschiedener Ontologien in eine logische Beziehung zueinander zu setzen.

Die Musterlösung deckt nur einen Teil der beschriebenen Domäne ab. Daher kann das Expertenmodul mit den beschriebenen Vergleichen zwar Fehler beim Verständnis von OWL erkennen, aber beispielsweise keine Fehler beim Verständnis von *Pizzas*, die in der Musterlösung nicht vorkommen. Um den Unterschied zu erläutern wird im Beispiel 4.1 eine syntaktisch und logisch korrekte Pizza definiert, die aber von jedem Menschen mit rudimentärem Domänenwissen als falsch gewertet wird, da sich *Fleischbelag* und *vegetarische Pizza* widersprechen.

```
Class: VegetarianPizza
    EquivalentTo:
        Pizza
         and (hasPizzaTopping some MeatTopping)
```

Codebeispiel 4.1: Semantisch falsche vegetarische Pizza mit Fleisch

Für Vergleiche auf dieser Ebene könnte die Musterlösung zu einem Domänenmodell erweitert werden. Das Ziel des Expertenmoduls ist jedoch nicht die vollständige Beschreibung einer Domäne, sondern das Erzeugen konstruktiver Rückmeldungen zu gängigen Fehlern.

4.2. Szenario: Übung mit automatisierter Analyse

Die Studenten formulieren ihre Lösungen in einem Ontologieeditor, der sie in einer OWL-Syntax [2] speichert. Anschließend übergeben sie diese gespeicherte Ontologie dem ITS, das die Abgabe analysiert. Im Unterschied zur momentanen Vorgehensweise wird der Vergleich von Abgabe und Musterlösung von dem im Rahmen dieser Arbeit entwickelten Expertenmodul übernommen, wobei mit dem entwickelten Ansatz nur OWL-Klassen verglichen werden.

Im *Übungsmodus* wird die Abgabe um Hinweise zu erkannten Fehlern ergänzt. Diese kommentierte Ontologie wird unmittelbar nach der Analyse an den Studenten zurückgegeben, der im Ontologieeditor diese Hinweise betrachten und sofort berücksichtigen kann (siehe Anwendungsfalldiagramm 5.2 auf Seite 59). Im *Abgabemodus* wird keine Ontologie zurückgegeben, nur die erreichte Punkteanzahl. Da die wesentliche Eigenschaft des Expertenmoduls die Unterstützung der Studenten beim Erlernen der Ontologiemodellierung ist, wird einstweilen nur eine sehr einfache Beurteilung vorgenommen.

4.2.1. Grenzen der automatischen Prüfung

Der Vergleich durch das Expertenmodul berücksichtigt nur bestimmte Faktoren und ist daher *keine vollständige Prüfung*. Aus diesem Grund muss es den Studenten bei begründetem Verdacht möglich sein, die Abgabe am Expertenmodul vorbei direkt an einen menschlichen Tutor weiterzuleiten. Besonders wichtig ist das bei vermutetem Fehlverhalten des ITS, etwa wenn die Rückmeldung widersprüchliche Aussagen enthält. Daraus ergibt sich, dass das erstellte Modul kein vollständiger Ersatz der Tutoren sein kann.

4.3. Beispielontologie: Pizzas

Das durchgehende Beispiel dieser Arbeit basiert auf einer Ontologie über Pizzas[1], die an der Universität Manchester speziell zum Lehren von OWL-DL entworfen wurde [31].

Für die vorliegende Arbeit liegt der Fokus klar auf Grundlagen wie dem korrekten Verwenden der Syntax von OWL und weniger auf fortgeschrittenen Themen wie der Modularisierung von Ontologien. Ebenso wurde auf die im Original sehr betonte Verbalisierung der Axiome verzichtet.

Um Missverständnissen durch die Adaption der Ontologie vorzubeugen wurden durchgehend eigene IRIs verwendet. Weiters wurde die ursprüngliche Ontologie in Basisdaten und Musterlösung geteilt. Die Studenten erhalten die Basisontologie (Präfix *b*) gemeinsam mit der Angabe und verwenden dieses Basisvokabular bei der Modellierung ihrer Lösung. Anhand der in den Codebeispielen 4.2 und 4.3 angeführten Musterlösung respektive Basisdaten wird im weiteren Verlauf der Arbeit die Analyse erläutert.

Um den Studierenden eine möglichst präzise Rückmeldung zu geben, werden diese Mängel unterschiedlichen Fehlertypen zugeordnet. Zur besseren algorithmischen Erfassung werden diese Fehlertypen in einzelne Symptome heruntergebrochen.

[1] `http://www.co-ode.org/ontologies/pizza/2007/02/12/`

```
Ontology: <http://dke.jku.at/2010/muster.owl>
Import: <http://dke.jku.at/2010/ingredients.owl>
Prefix: b: <http://dke.jku.at/2010/ingredients.owl#>

Class: FruttiDiMare
    SubClassOf:
        b:NamedPizza,
        b:hasPizzaTopping some b:GarlicTopping,
        b:hasPizzaTopping some b:MixedSeafoodTopping,
        b:hasPizzaTopping some b:TomatoTopping,
        b:hasPizzaTopping only
            (b:GarlicTopping
             or b:MixedSeafoodTopping
             or b:TomatoTopping)
    DisjointWith:
        Margherita

Class: Margherita
    SubClassOf:
        b:NamedPizza,
        b:hasPizzaTopping some b:MozzarellaTopping,
        b:hasPizzaTopping some b:TomatoTopping,
        b:hasPizzaTopping only
            (b:MozzarellaTopping
             or b:TomatoTopping)
    DisjointWith:
        FruttiDiMare

Class: VegetarianPizza
    EquivalentTo:
        b:Pizza
         and (not (b:hasPizzaTopping some b:FishTopping))
         and (not (b:hasPizzaTopping some b:MeatTopping))

Class: CheesyPizza
    EquivalentTo:
        b:Pizza
         and (b:hasPizzaTopping some b:CheeseTopping)

Class: Icecream
    SubClassOf:
        b:Food,
        b:hasIngredient some b:FruitTopping

DisjointClasses:
    b:Food,b:Pizza,b:PizzaTopping,Icecream
```

Codebeispiel 4.2: Pizza Musterlösung

```
Ontology: <http://dke.jku.at/2010/ingredients.owl>

ObjectProperty: hasIngredient
    Characteristics:
        Transitive
    Domain:
        Food
    Range:
        Food
    InverseOf:
        isIngredientOf

ObjectProperty: isToppingOfPizza
    SubPropertyOf:
        isIngredientOf
    Characteristics:
        Functional
    Domain:
        PizzaTopping
    Range:
        Pizza
    InverseOf:
        hasPizzaTopping

ObjectProperty: hasPizzaTopping
    SubPropertyOf:
        hasIngredient
    Characteristics:
        InverseFunctional
    Domain:
        Pizza
    Range:
        PizzaTopping
    InverseOf:
        isToppingOfPizza

ObjectProperty: isIngredientOf
    Characteristics:
        Transitive
    Domain:
        Food
    Range:
        Food
    InverseOf:
        hasIngredient

Class: NamedPizza
    SubClassOf:
        Pizza
```

```
Class: CheeseTopping
    SubClassOf:
        PizzaTopping
    DisjointWith:
        FishTopping, FruitTopping, MeatTopping, VegetableTopping

Class: MixedSeafoodTopping
    SubClassOf:
        FishTopping

Class: VegetableTopping
    SubClassOf:
        PizzaTopping
    DisjointWith:
        CheeseTopping, FishTopping, FruitTopping, MeatTopping

Class: Pizza
    SubClassOf:
        Food
    DisjointWith:
        PizzaTopping

Class: Food
    SubClassOf:
        owl:Thing

Class: MeatTopping
    SubClassOf:
        PizzaTopping
    DisjointWith:
        CheeseTopping, FishTopping, FruitTopping, VegetableTopping

Class: SundriedTomatoTopping
    SubClassOf:
        TomatoTopping
    DisjointWith:
        SlicedTomatoTopping

Class: FruitTopping
    SubClassOf:
        PizzaTopping
    DisjointWith:
        CheeseTopping, FishTopping, MeatTopping, VegetableTopping

Class: GarlicTopping
    SubClassOf:
        VegetableTopping
    DisjointWith:
```

```
        TomatoTopping

Class: SlicedTomatoTopping
    SubClassOf:
        TomatoTopping
    DisjointWith:
        SundriedTomatoTopping

Class: PizzaTopping
    SubClassOf:
        Food
    DisjointWith:
        Pizza

Class: ParmaHamTopping
    SubClassOf:
        HamTopping

Class: FishTopping
    SubClassOf:
        PizzaTopping
    DisjointWith:
        CheeseTopping, FruitTopping, MeatTopping, VegetableTopping

Class: TomatoTopping
    SubClassOf:
        VegetableTopping
    DisjointWith:
        GarlicTopping

Class: HamTopping
    SubClassOf:
        MeatTopping
    DisjointWith:
        PeperoniSausageTopping

Class: MozzarellaTopping
    SubClassOf:
        CheeseTopping

Class: PeperoniSausageTopping
    SubClassOf:
        MeatTopping
    DisjointWith:
        HamTopping
```

Codebeispiel 4.3: Pizza Basisdaten

4.4. Fehlertypen

Zur systematischen Erfassung der Unterschiede zwischen Abgabe und Musterlösung werden Fehlertypen definiert. Diese werden einzelnen Themenbereichen zugeordnet, um die Rückmeldung zu erleichtern. Ob eine Klasse wirklich fehlerhaft ist, kann letztlich nur der Vergleich mit der Musterlösung entscheiden, da die Aufgabenstellung die Definition einer fehlerhaften Klasse beinhalten kann. Die nächsten Seiten enthalten die für die Analyse nötigen Definitionen, anschließend werden alle Fehlertypen vorgestellt.

4.4.1. Definitionen

Bezeichnungen

Die vom Studenten abgegebene Ontologie O_s soll mit der von der Lehrveranstaltungsleitung definierten Musterontologie O_m verglichen werden. Dies wird durch die von beiden Lösungen verwendete Basisontologie O_b erleichtert. Zur Unterscheidung der Ontologien definiert O_s eigene Konzepte mittels IRI_s während O_m dafür die IRI_m benutzt. Die Basisontologie verwendet wiederum eine eigene IRI_b.

Jeder in der Musterlösung definierten Klasse $k_m \in O_m$ entspricht idealerweise eine Klasse $k_s \in O_s$. Eine Klasse wird im logischen Kontext auch *Konzept* genannt. Die Mengen aller Klassen einer Ontologie werden mit K_s, K_m und K_b für die Ontologien O_s O_m beziehungsweise O_b bezeichnet.

Jede Klasse k_m ist durch eine analog benannte Axiomsmenge A_{k_m} definiert, die einzelne Axiome a_{m_1} bis a_{m_n} enthält. Die Definitionen für Muster- und Basisontologie erfolgen analog. Falls der Kontext klar ist wird als vereinfachte Schreibweise a_1 bis a_n verwendet. Um zwei Axiome leichter vergleichen zu können werden diese in die *Negationsnormalform* (NNF) gebracht. Die NNF ist eine logische Darstellung, in der Verneinung nur direkt vor atomaren Konzepten auftritt [8, 82f].

Eine *geforderte Klasse* $k_{mg} \in K_m$ ist eine k_m, der eine $k_s \in K_s$ „gleichen“ muss. Da solche Klassen der Namenskonvention entsprechen müssen, fallen Hilfsklassen nicht unter die geforderten Klassen.

Eine *Position* ist einer der Orte, an der ein Konzept in der klassifizierten Hierarchie einer Ontologie zu finden ist. Um Klassen unterschiedlicher Ontologien zu vergleichen werden diese vorher vereinigt und die so entstandene Ontologie klassifiziert.

Gleichheit

Die Frage nach der Gleichheit zweier Klassen kann in drei Teile zerlegt werden, und zwar der Gleichheit der Namen der Klassen, der Menge an Axiomen mittels derer die Klasse definiert wurde und schließlich dem Verhalten der Klassen in Bezug auf andere Klassen, also die Position innerhalb der Klassenhierarchie.

Zwei Konzepte k_m und k_s sind *namensgleich*, wenn sie die gleiche Zeichenkette als Name verwenden. *Positionsgleichheit* von Klassen tritt in zwei Formen auf: Falls eine der beiden Klassen über *EquivalentTo* definiert wurde müssen die Klassen als äquivalent

klassifiziert werden. Andernfalls müssen sie die selben geforderten Superklassen aufweisen.

Klassen sind *inhaltsgleich* wenn ihre in NNF betrachteten Klassenaxiome syntaktisch gleich sind. Klassen sind *gleich*, wenn sie (a) sowohl positions- als auch namensgleich oder (b) sowohl inhalts- als auch namensgleich sind. Axiome sind *gleich*, wenn ihre Negationsnormalformen äquivalent sind. Ontologien sind *gleich* wenn sie über die selbe IRI definiert sind und die darin enthaltenen Axiome gleich sind.

Die beiden Klassen im nachfolgenden Beispiel sind inhaltsgleich, aber nicht namensgleich. Da sich die Position innerhalb der Hierarchie aus den Klassenaxiomen ergibt (aber nicht umgekehrt) ist Positionsgleichheit gegeben.

```
Class: CheesyPizza
    EquivalentTo:
        Pizza
         and (hasPizzaTopping
          some CheeseTopping)
```

```
Class: PizzaWithCheese
    EquivalentTo:
        Pizza
         and (hasPizzaTopping
          some CheeseTopping)
```

4.4.2. Gruppe „Angabe oder Sachverhalt falsch verstanden“

Die in dieser Gruppe zusammengefassten Fehler entstehen, wenn Teile der Aufgabenstellung nicht beachtet oder zur Analyse benötigte Rahmenbedingungen nicht eingehalten werden. Mögliche Gründe dafür sind Unachtsamkeit seitens des Studenten oder mangelnde Kenntnis des in der Angabe beschriebenen Sachverhalts.

Fehlertyp 1 (Klasse auf falscher Ebene definiert)**.**

Merkmale. Das Konzept ist sowohl in der Musterlösung als auch in der Abgabe vorhanden, die Positionen stimmen aber nicht überein. Zwischen den beiden Klassen kann keine Super-/Subklassenbeziehung hergestellt werden, daher kann das Konzept nicht als unter- oder überspezifiziert, sonder nur als *anders definiert* erkannt werden.

Hintergrund. Die Klassen k_b und k_m sind so definiert, dass keine Super-/Subklassenbeziehung existiert.

Beispiel. Die Pizza $Margherita_s$ wurde als *ItalianPizza* statt als *NamedPizza* definiert (vgl. Codebeispiele 4.4 und 4.5). Da *ItalianPizza* auf der gleichen Ebene wie *NamedPizza* definiert wurde befindet sich $Margherita_s$ in einem anderen Teilbaum der Hierarchie als $Margherita_m$. Keine der beiden kann als Subklasse der anderen Klasse gesehen werden.

```
Class: ItalianPizza
    SubClassOf:
        Pizza

Class: Margherita
    SubClassOf:
        ItalianPizza,
        hasPizzaTopping some MozzarellaTopping,
        hasPizzaTopping some TomatoTopping,
```

```
        hasPizzaTopping only (MozzarellaTopping
        or TomatoTopping)
```

Codebeispiel 4.4: Falsch: *Margherita* direkt als *ItalianPizza* definiert

```
Class: NamedPizza
    SubClassOf:
        Pizza

Class: Margherita
    SubClassOf:
        NamedPizza,
        hasPizzaTopping some MozzarellaTopping,
        hasPizzaTopping some TomatoTopping,
        hasPizzaTopping only (MozzarellaTopping
        or TomatoTopping)
```

Codebeispiel 4.5: Richtig: *Margherita* via *NamedPizza* definiert

Fehlertyp 2 (Klasse zusätzlich definiert)**.**

Merkmale. Die Klasse ist in der Abgabe vorhanden, gleicht aber keiner geforderten Klasse.

Hintergrund. Die Klasse kann überflüssig oder als Hilfsklasse verwendet sein. Im letzteren Fall ist die Verwendung gewollt.

Beispiel. Die Hilfsklasse $ItalianPizza_s$ wurde eingeführt. In der Musterlösung ist keine derartige Klasse vorhanden.

```
Class: ItalianPizza
    SubClassOf:
        Pizza
```

Codebeispiel 4.6: Unnötige Definition von *ItalianPizza*

Fehlertyp 3 (Geforderte Klasse nicht definiert)**.**

Merkmale. Keine Klasse der Abgabe gleicht der geforderten Klasse.

Hintergrund. Der Student hat vergessen, eine in der Angabe geforderte Klasse zu definieren.

Beispiel. Es wurde keine Pizza $Margherita_s$ definiert, daher fehlt dieses Konzept in der abgegebenen Lösung.

Fehlertyp 4 (Konzept unterspezifiziert)**.**

Merkmale. Das unterspezifizierte Konzept wurde nicht so präzise wie in der Musterlösung beschrieben und ist daher zu allgemein definiert.

Hintergrund. Die Angabe wurde nicht genau genug eingehalten, möglicherweise aufgrund einer unterschiedlichen Auffassung des Sachverhalts.

Beispiel. Die Pizza $Margherita_s$ wurde mit irgendeiner Art Käse belegt. Da ein bestimmter Käse verwendet werden muss ist diese Lösung zu allgemein.

```
Class: Margherita
    SubClassOf:
        NamedPizza,
        hasPizzaTopping some CheeseTopping,
        hasPizzaTopping some TomatoTopping,
        hasPizzaTopping only (CheeseTopping
        or TomatoTopping)

Class: MozzarellaTopping
    SubClassOf:
        CheeseTopping
```

Codebeispiel 4.7: Falsch: *CheeseTopping* ist zu allgemein

```
Class: Margherita
    SubClassOf:
        NamedPizza,
        hasPizzaTopping some MozzarellaTopping,
        hasPizzaTopping some TomatoTopping,
        hasPizzaTopping only (MozzarellaTopping
        or TomatoTopping)

Class: MozzarellaTopping
    SubClassOf:
        CheeseTopping
```

Codebeispiel 4.8: Richtig: Laut Angabe muss *Mozzarella* verwendet werden

Fehlertyp 5 (Konzept überspezifiziert)**.**

Merkmale. Konzept befindet sich in Hierarchie zu weit unten (also zu weit entfernt von *owl:Thing*).

Hintergrund. Das entsprechende Konzept wurde enger definiert als in der Musterlösung vorgesehen und ist daher zu speziell. Die Angabe wurde nicht genau genug eingehalten oder der Sachverhalten anders verstanden.

Beispiel. Die Pizza $Margherita_s$ wurde mit einer speziellen Art von Tomaten belegt. In der Angabe wurden hingegen beliebige Tomaten gefordert, daher ist diese Lösung zu speziell.

```
Class: Margherita
    SubClassOf:
        NamedPizza,
        hasPizzaTopping some MozzarellaTopping,
```

```
        hasPizzaTopping some SundriedTomatoTopping,
        hasPizzaTopping only (MozzarellaTopping
        or SundriedTomatoTopping)

Class: SundriedTomatoTopping
    SubClassOf:
        TomatoTopping
```

Codebeispiel 4.9: Falsch: *SundriedTomatoTopping* ist zu speziell

```
Class: Margherita
    SubClassOf:
        NamedPizza,
        hasPizzaTopping some MozzarellaTopping,
        hasPizzaTopping some TomatoTopping,
        hasPizzaTopping only (MozzarellaTopping
        or TomatoTopping)

Class: SundriedTomatoTopping
    SubClassOf:
        TomatoTopping
```

Codebeispiel 4.10: Richtig: Laut Rezept reicht irgendein *TomatoTopping* aus

Fehlertyp 6 (Namenskonvention nicht eingehalten)**.**

Merkmale. Eine geforderte Klasse gleicht in Position oder Inhalt einer Klasse der Musterlösung, wurde aber nicht wie vereinbart benannt.

Hintergrund. Die Namenskonvention wurde nicht eingehalten. Rechtschreibfehler können ebenfalls Ursache dieses Fehlers sein.

Beispiel. Die Pizza *Margherita* wurde fälschlicherweise *Margarita* genannt. Eine Ähnlichkeit der Namen ist in diesem Fall erkennbar.

Fehlertyp 7 (Basisdaten verändert)**.**

Merkmale. Die Basisontologien von Musterlösung und Abgabe stimmen nicht überein.

Hintergrund. Der Inhalt der ausgegebenen Basisontologie O_b wurde verändert, was laut Angabe nicht erlaubt ist.

Beispiel. Eine neue Unterkategorie von Fleischbelag wurde definiert. Anschließend wurde ein in O_b vorgegebener Pizzabelag in diese Kategorie verschoben. Da beim Verschieben bestehende Basisdaten geändert werden müssen widerspricht das der Aufgabenstellung.

```
Class: SausageTopping
    SubClassOf:
        MeatTopping
    DisjointWith:
        HamTopping
```

```
Class: PeperoniSausageTopping
    SubClassOf:
        SausageTopping
```

Codebeispiel 4.11: Falsch: *PeperoniSausageTopping* verändert

```
Class: PeperoniSausageTopping
    SubClassOf:
        MeatTopping
    DisjointWith:
        HamTopping
```

Codebeispiel 4.12: Richtig: Basisdaten unverändert

4.4.3. Gruppe „Probleme mit Prädikatenlogik und logischen Symbolen“

Diese Gruppe umfasst Fehler, die durch mangelnde Kenntnis formaler Logik oder deren Symbole entstehen. Übliche Anfängerfehler sind beispielsweise das Verwechseln von Quantoren oder die falsche Positionierung der Verneinung [31, 71ff].

Fehlertyp 8 (Bedingung durch leere Menge trivial erfüllbar)**.**

Merkmale. Vom Studenten möglicherweise nicht beabsichtigte Klassifikation.

Hintergrund. Ein Widerspruch erst durch einen Wert. *Für alle* (∀, `only`) beinhaltet kein *es gibt* (∃, `some`) und kann durch die leere Menge (*owl:Nothing*) erfüllt werden [31, 72]. Jede in der Definition auf `only` folgende Klasse A muss mittels `some` abgeschlossen werden. Die auf `some` folgende Klasse muss dabei gleich A oder Unterklasse von A sein.

Beispiel. Eine leere Pizza ist nicht belegt und enthält daher auch keinen Fisch- oder Fleischbelag. Sie wird als vegetarisch klassifiziert, obwohl sie keinen vegetarischen Belag enthält.

```
Class: EmptyPizza
    SubClassOf:
        Pizza
         and (not (hasPizzaTopping some Thing))

Class: VegetarianPizza
    EquivalentTo:
        Pizza
         and (hasPizzaTopping only (not (FishTopping)))
         and (hasPizzaTopping only (not (MeatTopping)))
```

Codebeispiel 4.13: Falsch: Leere Pizza gilt als vegetarisch

```
Class: VegetarianPizza
    EquivalentTo:
        Pizza
         and (hasPizzaTopping only (not (FishTopping)))
         and (hasPizzaTopping only (not (MeatTopping)))
         and (hasPizzaTopping some Thing)
```

Codebeispiel 4.14: Bessere Alternative: Existenzquantor verbietet leere Menge

Fehlertyp 9 (Verneinung falsch positioniert)**.**

Merkmale. Die Position mancher Konzepte in der Hierarchie entspricht nicht den Erwartungen des Studenten.

Hintergrund. Die Ausdrücke „some not“ und „not some“ wurden verwechselt [31, 73f].

Beispiel. Die Definition von *VegetarianPizza*$_s$ in Beispiel 4.15 sagt aus, dass mindestens ein Belag „etwas außer Fisch“ und mindestens ein Belag „etwas außer Fleisch“ sein muss. Es wird nicht gefordert, dass die beiden Einschränkungen für den selben Belag gelten.

Eine nur mit Fleisch und Fisch belegte Pizza würde die falsche Lösung erfüllen, da die Forderung in Zeile 4 (*Belag, der kein Fisch ist*) vom Fleischbelag erfüllt wird und die in Zeile 5 (*Belag, der kein Fleisch ist*) vom Fischbelag. Somit wird die Pizza fälschlicherweise als vegetarisch erkannt.

```
Class: VegetarianPizza                                          1
    EquivalentTo:                                               2
        Pizza                                                   3
         and (hasPizzaTopping some (not (FishTopping)))         4
         and (hasPizzaTopping some (not (MeatTopping)))         5
```

Codebeispiel 4.15: Falsch: Auch *FruttiDiMare* gilt als vegetarisch

```
Class: VegetarianPizza
    EquivalentTo:
        Pizza
         and (not (hasPizzaTopping some FishTopping))
         and (not (hasPizzaTopping some MeatTopping))
```

Codebeispiel 4.16: Richtig: Nur *Margherita* gilt als vegetarisch

Fehlertyp 10 (Widerspruch innerhalb der Ontologie)**.**

Merkmale. Der Reasoner klassifiziert ein Konzept als äquivalent zu *owl:Nothing*.

Hintergrund. Dieses Konzept ist im Rahmen dieser Ontologie nicht möglich, da sich Axiome widersprechen.

Beispiel. Icecream wurde mit Früchten *belegt*. Weil *mit Pizzabelag belegen* nur bei einer *Pizza* möglich ist, müsste *Icecream* eine *Pizza* sein. Da *Icecream* und *Pizza* als überschneidungsfrei definiert wurden entsteht ein Widerspruch in der Definition von *Icecream*.

```
Class: Icecream
    SubClassOf:
        Food,
        hasPizzaTopping some FruitTopping

ObjectProperty: hasPizzaTopping
    Domain:
        Pizza
    Range:
        PizzaTopping

Class: FruitTopping
    SubClassOf:
        PizzaTopping

DisjointClasses:
    Pizza, Icecream
```

Codebeispiel 4.17: Falsch: *Icecream* verletzt den Wertebereich von *hasPizzaTopping*

```
Class: Icecream
    SubClassOf:
        Food,
        hasIngredient some FruitTopping

ObjectProperty: hasIngredient
    Domain:
        Food
    Range:
        Food

Class: FruitTopping
    SubClassOf:
        PizzaTopping

DisjointClasses:
    Pizza, Icecream
```

Codebeispiel 4.18: Richtig: *Icecream* hat Früchte als *Zutat*

Fehlertyp 11 (Implizites Wissen nicht explizit ausgedrückt)**.** Dieser Fehler erfüllt die gleichen Merkmale wie Fehlertyp 4. Da die Fehler jedoch aus unterschiedlichen Gründen entstehen, müssen beide in die Rückmeldung aufgenommen werden.

Merkmale. Aufgrund fehlender Axiome ist das Konzept zu allgemein definiert.

Hintergrund. Im Namen eines Konzepts enthaltene Information wurde nicht in logische Ausdrücke gefasst, beispielsweise wenn *Thunfisch* nicht als *Fisch* definiert wurde, weil diese Tatsache für den Modellierenden selbstverständlich war.

Da der Reasoner aus der Bezeichnung der Konzepte nicht erkennen kann, ob ein Fisch, ein Gegenstand, eine Person bestimmten Geschlechts oder etwas Anderes beschrieben wird ist die betreffende Klasse unterspezifiziert.

Beispiel. Die Pizza *FruttiDiMare* ist unter anderem mit *MixedSeafoodTopping* belegt. Wird wie im Beispiel vergessen, diesen Belag als *FishTopping* zu definieren, so gilt die Pizza fälschlicherweise als vegetarisch.

```
Class: FishTopping
    SubClassOf:
        PizzaTopping

Class: MixedSeafoodTopping
    SubClassOf:
        PizzaTopping
    DisjointWith:
        FishTopping,
        MeatTopping

Class: FruttiDiMare
    SubClassOf:
        NamedPizza,
        hasPizzaTopping some GarlicTopping,
        hasPizzaTopping some MixedSeafoodTopping,
        hasPizzaTopping some TomatoTopping,
        hasPizzaTopping only (GarlicTopping
        or MixedSeafoodTopping
        or TomatoTopping)

Class: VegetarianPizza
    EquivalentTo:
        Pizza
        and (not (hasPizzaTopping some FishTopping))
        and (not (hasPizzaTopping some MeatTopping))
```

Codebeispiel 4.19: Falsch: *MixedSeafoodTopping* direkt als *PizzaTopping* definiert

```
Class: FishTopping
    SubClassOf:
        PizzaTopping

Class: MixedSeafoodTopping
    SubClassOf:
        FishTopping

Class: FruttiDiMare      // wie oben definiert

Class: VegetarianPizza   // wie oben definiert
```

Codebeispiel 4.20: Richtig: *MixedSeafoodTopping* über *FishTopping* definiert

Fehlertyp 12 (Allquantor statt Existenzquantor verwendet)**.**

Merkmale. Einige Individuen oder Klassen werden einem Konzept entgegen der Absicht des Studenten nicht zugeordnet.

Hintergrund. Anfänger verwenden oft den Allquantor (DL[2]: ∀, Manchester: `only`) anstatt des Existenzquantors (DL: ∃, Manchester: `some`) [31, 66].

Da die Manchester Syntax auf die Verwendung spezieller Symbole verzichtet wird dieses Problem weitgehend entschärft.

Beispiel. In der abgegebenen Lösung müssen *alle* (∀) Beläge einer käsehaltigen Pizza irgendeine Art von Käse sein. Die Pizza *Margherita* ist mit Käse und Tomaten belegt und gilt daher nicht als käsige Pizza.

Laut Angabe muss jedoch nur mindestens ein (∃) Käsebelag vorhanden sein, *Margherita* soll also als käsehaltige Pizza klassifiziert werden.

```
Class: CheesyPizza
    EquivalentTo:
        Pizza
         and (hasPizzaTopping only CheeseTopping)

Class: Margherita
    SubClassOf:
        NamedPizza,
        hasPizzaTopping some MozzarellaTopping,
        hasPizzaTopping some TomatoTopping,
        hasPizzaTopping only
            (MozzarellaTopping
             or TomatoTopping)
```

Codebeispiel 4.21: Falsch: Käsepizza akzeptiert *nur* Käsebelag (und wird daher auch als vegetarisch klassifiziert)

```
Class: CheesyPizza
    EquivalentTo:
        Pizza
         and (hasPizzaTopping some CheeseTopping)

Class: Margherita    // wie oben definiert
```

Codebeispiel 4.22: Richtig: Käsepizza benötigt *mindestens einen* Käsebelag

Fehlertyp 13 (Probleme mit logischem *Oder*)**.**

Merkmale. Die mittels Konjunktion oder Disjunktion formulierten Axiome stellen den Sachverhalt der Angabe nicht korrekt dar.

[2]Die DL-Syntax verwendet logische Symbole zum Formulieren von Beschreibungslogiken

Hintergrund. Anfänger ohne Vorwissen in Aussagenlogik oder Programmieren sind mit der unterschiedlichen Verwendung von *Und* und *Oder* in Allgemeinsprache und Logik nicht vertraut [31, 72]. Während das logische *Und* bei korrekter Klammerung dem umgangssprachlichen entspricht, können bei *Oder* folgende Probleme auftreten:

Die formulierten Axiome sind unpräziser als erwartet, da umgangssprachliches *Oder* sowohl *nichtausschließend* als auch *ausschließend* eingesetzt werden kann („Bitte eine Pizza mit Salami oder mit Schinken!“). Das Fehlen eines *ausschließenden Oders* in OWL verschärft diese Fehlerquelle zusätzlich, weshalb die Manchester Syntax ein `xor`-Konstrukt als Abkürzung anbietet [19, 8]).

Weiters bereiten die ungewohnten Symbole der DL-Syntax Anfängern oft Probleme [19, 2]. Verwechslungsgefahr besteht unter anderem zwischen den Zeichen für Vereinigung (DL: $\sqcup$, Manchester: `and`) und Durchschnitt (DL: $\sqcap$, Manchester: `or`). Eine Erklärung unter Verwendung der Mengenlehre könnte das Verständnis erleichtern.

Beispiel. Umgangssprachlich essen Vegetarier „keinen Pizzabelag, der Fisch *oder* Fleisch enthält“. Die unkritische Übersetzung dieser Aussage im Beispiel 4.23 fordert aber nur mindestens eine vegetarische Zutat, daher wird *FruttiDiMare* als vegetarisch erkannt.

```
Class: VegetarianPizza
    EquivalentTo:
        Pizza
        and (not (hasPizzaTopping some FishTopping))
        or (not (hasPizzaTopping some MeatTopping))
```

Codebeispiel 4.23: Falsch: *FruttiDiMare* erfüllt *eine* der Bedingungen und wird als vegetarisch klassifiziert

```
Class: VegetarianPizza
    EquivalentTo:
        Pizza
        and (not (hasPizzaTopping some FishTopping))
        and (not (hasPizzaTopping some MeatTopping))
```

Codebeispiel 4.24: Richtig: *Beide* Bedingungen müssen erfüllt werden

4.4.4. Gruppe „OWL-bezogene Probleme“

Die in dieser Gruppe behandelten Fehler beziehen sich auf das Verständnis von OWL. Einsteiger können oft nicht einschätzen, welche Informationen dem Reasoner zur Verfügung stehen. Sie verfügen auch häufig über Erfahrung mit abgeschlossenen Systemen, daher sind ihnen die Auswirkungen der *Open World Assumption* fremd [31].

Fehlertyp 14 (*Open World Assumption* bei Klassen nicht beachtet)**.**

Merkmale. Ein Konzept wird einem allgemeineren Konzept nicht zugeordnet. In dessen Definition wurden die verwendeten Existenzquantoren nicht mit Allquantoren abgeschlossen.

Hintergrund. Das entworfene Modell steht einer Erweiterung offen und kann jederzeit um zusätzliche Fakten erweitert werden. Informationen können *wahr*, *falsch* oder *unbekannt* sein. Abgeschlossene, also vollständig bekannte Fakten müssen daher explizit als solche definiert werden.

Da praktisch alle Anfänger bisher nur mit *Closed World Assumption* zu tun hatten, in der die vorliegende Information vollständig und entweder *wahr* oder *falsch* ist, stellt dies ihre wesentlichste Hürde beim Erlernen von OWL dar [31, 68f].

Beispiel. Eine *Pizza Margherita* ist mit vegetarischen Zutaten belegt. Sie wird aber nicht als vegetarisch erkannt, da die Zutaten nicht abgeschlossen wurden. Die Musterlösung sagt aus, dass eine Pizza Margherita *nur* mit Mozzarella *und* Tomaten belegt ist und *mit nichts anderem*.

```
Class: Margherita
    SubClassOf:
        NamedPizza,
        hasPizzaTopping some MozzarellaTopping,
        hasPizzaTopping some TomatoTopping
```

Codebeispiel 4.25: Falsch: Andere Pizzabeläge sind nicht ausgeschlossen

```
Class: Margherita
    SubClassOf:
        NamedPizza,
        hasPizzaTopping some MozzarellaTopping,
        hasPizzaTopping some TomatoTopping,
        hasPizzaTopping only (MozzarellaTopping or TomatoTopping)
```

Codebeispiel 4.26: Richtig: *Nur* die angegebenen Beläge werden verwendet

Fehlertyp 15 (Einschränkungen von Werte- und Zielbereich sind auch Axiome)**.**

Merkmale. Die Angabe von Werte- oder Zielbereichen wirkt sich auf die Klassifikationshierarchie aus, obwohl das nicht beabsichtigt war. Das Konzept ist daher enger definiert als erwartet, im Extremfall ist es widersprüchlich.

Hintergrund. Entgegen dem Verständnis vieler Anfänger werden derartige Einschränkungen nicht nur überprüft, sondern auch für die Klassifikation berücksichtigt [31, 70f].

Beispiel. Wie im Beispiel zu Fehlertyp 10 auf Seite 29 ersichtlich wird durch die Verwendung von *hasPizzaTopping* dessen Wertebereich (*Pizza*) dem Typ der Klasse zugewiesen. *Icecream* wird dadurch als *Pizza* eingestuft und ist unerfüllbar.

Bei der richtigen Definition von *Icecream* mittels *hasIngredient* wird die Auswirkung dieser Eigenschaft von OWL erneut deutlich:

```
Class: Icecream
    SubClassOf:
        hasIngredient some FruitTopping

ObjectProperty: hasIngredient
    Domain:
        Food
```

Codebeispiel 4.27: Richtig: *Icecream* wird als *Food* klassifiziert

Fehlertyp 16 (*Open World Assumption* bei Properties nicht beachtet)**.**

Merkmale. Es wird nichts als zur Klasse zugehörig klassifiziert.

Hintergrund. Beim Definieren von Konzepten ist die Unterscheidung zwischen notwendig (Manchester: `SubClassOf`) und hinreichend (Manchester: `EquivalentTo`) definierten Konzepten sehr wichtig [20]. Hinreichende Klassen können rein durch Klassifikation zusätzliche Subklassen erhalten, bei notwendigen Klassen sind nur explizit definierte Subklassen möglich.

Beispiel. Es wurden nur notwendige, aber keine hinreichenden Kriterien für eine käsige Pizza definiert. Daher kann keine Pizza als *CheesyPizza* klassifiziert werden [31, 67f].

```
Class: CheesyPizza
    SubClassOf:
        Pizza
         and (hasPizzaTopping some CheeseTopping)
```

Codebeispiel 4.28: Falsch: Definition von *CheesyPizza* nicht abgeschlossen

```
Class: CheesyPizza
    EquivalentTo:
        Pizza
         and (hasPizzaTopping some CheeseTopping)
```

Codebeispiel 4.29: Richtig: *CheesyPizza* als hinreichend definiert

Fehlertyp 17 (Überlappung von Klassen nicht beachtet)**.**

Merkmale. Ein Konzept wird nicht wie gewünscht klassifiziert.

Hintergrund. Unterschiedlich benannte Konzepte sind nur dann überschneidungsfrei, wenn sie explizit als disjunkt definiert werden.

Beispiel. Ein vegetarischer Pizzabelag könnte sich mit Fisch- oder Fleischbelag überschneiden, deshalb wird *Pizza Margherita* nicht als vegetarisch erkannt. Werden die Zutaten als disjunkt definiert, so erfolgt auch die Klassifikation als vegetarisch.

```
Class: VegetableTopping
    SubClassOf:
        PizzaTopping
    DisjointWith:
        CheeseTopping,
        FruitTopping
```

Codebeispiel 4.30: Falsch: Gemüse, Fisch und Fleisch sind nicht überschneidungsfrei

```
Class: VegetableTopping
    SubClassOf:
        PizzaTopping
    DisjointWith:
        CheeseTopping,
        FishTopping,
        FruitTopping,
        MeatTopping
```

Codebeispiel 4.31: Richtig: Disjunktheit erlaubt die Klassifikation als vegetarisch

4.5. Analyse der abgegebenen Lösung

Die eben vorgestellten Fehlertypen sind zu abstrakt, um direkt erkannt zu werden und werden deshalb mit Hilfe von Symptomen erfasst. Nach der Beschreibung der einzelnen Symptome werden diese den entsprechenden Fehlertypen zugeordnet. Dieser Zusammenhang wird durch die Analyse eines einfachen Beispiels zusätzlich erläutert.

4.5.1. Sammeln von Symptomen

Jedes Symptom wird in Folge mit einem kurzen Beispiel vorgestellt. Die Einteilung der Symptome erfolgt nach der Art der analysierten Ähnlichkeit (vgl. 4.1).

Name der Klasse

Die Namen der Klassen werden rein auf syntaktischer Ebene verglichen, da sie keine logische Aussage enthalten.

Symptom N1 (ähnlich)**.** Die Namen von k_{mg} und k_s stimmen in mindestens der Hälfte der Zeichen des längeren Namens überein.

Im folgenden Beispiel ist *Margherita* das längere Wort, bestehend aus den zehn Zeichen *aaeghiMrrt*. Um das Symptom zu erfüllen müssen mindestens fünf Zeichen übereinstimmen. Da von den acht Zeichen *aaagiMrt* des kürzeren Wortes *Magarita* sieben übereinstimmen ist das Symptom positiv.

Class: **Magarita**	Class: **Margherita**

Symptom N2 (unbekannt)**.** Der Name von k_s entspricht keinem Namen einer geforderten Klasse.

Im Beispiel enthält O_s eine Klasse namens *ItalianPizza*, aber in der Musterontologie O_m existiert keine Klasse dieses Namens.

```
Class: ItalianPizza
```

```
// keine Entsprechung
```

Klassenaxiome

Symptome auf Klassenaxiome zeigen Unterschiede in der Definition von Klassen, also in der Ontologie enthaltenen Menge an Ausdrücken, die Einfluss auf die Klasse haben. Verschieden definierte Klassen können trotzdem logisch äquivalent sein und daher die selben Positionen innerhalb der Klassenhierarchie einnehmen.

Symptom K1 (inhaltsgleich)**.** Die Klassen k_{mg} und k_s wurden durch die gleichen Axiome definiert.

Im nachfolgenden Beispiel ist zwar die Deklaration a_0 der Klasse unterschiedlich, da aber die Klassenaxiome (in diesem Fall nur a_1) übereinstimmen, wurde das Symptom gefunden.

```
a0: Class: CheesyPizza
        EquivalentTo:
a1:         Pizza
             and (hasPizzaTopping
              some CheeseTopping)
```

```
a0: Class: PizzaWithCheese
        EquivalentTo:
a1:         Pizza
             and (hasPizzaTopping
              some CheeseTopping)
```

Klassenaxiome können in mehrere kleine Axiome aufgeteilt oder durch Einsatz von Junktoren zu größeren Axiomen kombiniert werden. Diese Kombination unterscheidet sich bei hinreichenden und notwendigen Axiomen und wird in den folgenden Beispielen erläutert. Das Symptom gilt jedoch nur als erfüllt, wenn die Axiome exakt gleich formuliert wurden, einschließlich eventueller Junktoren.
Das der Musterlösung entnommene Codebeispiel 4.32 definiert eine vegetarische Pizza mittels einem einzigen hinreichenden Axiom (`a1`):

```
    Class: s:VegetarianPizza
        EquivalentTo:
a1:         b:Pizza
             and (not (b:hasPizzaTopping some b:FishTopping))
             and (not (b:hasPizzaTopping some b:MeatTopping))
```

Codebeispiel 4.32: Richtige Definition mit einem hinreichenden Axiom

Werden die einzelnen, mit `and` verbundenen Teile für sich als hinreichend eingetragen, so ist die Bedeutung eine andere, da *jedes einzelne* Axiom alleine hinreichend ist, sie also durch `or` verknüpft sind (Axiom a1 oder Axiom a2 oder ...).

Die in Codebeispiel 4.33 erfolgte Definition mit mehreren Axiomen (`a1`, `a2`, `a3`) ist daher problematisch. Axiom `a1` erklärt vegetarische Pizza und Pizza als äquivalent, daher werden sehr viele andere Klassen zwangsläufig als unmöglich klassifiziert, unter anderem alle nicht vegetarischen Pizzen.

```
    Class: s:VegSeparate
        EquivalentTo:
a1:         b:Pizza,
a2:         not (b:hasPizzaTopping some b:FishTopping),
a3:         not (b:hasPizzaTopping some b:MeatTopping)
```

Codebeispiel 4.33: Falsche Definition mit mehreren hinreichenden Axiomen

Das nächsten Beispiel behandelt notwendige Axiome. Im wieder der Musterlösung entnommenen Codebeispiel 4.34 wurden alle Axiome getrennt formuliert (`a1` bis `a4`). Für jedes dieser Axiome erzeugt der Reasoner falls nötig eine anonyme Superklasse [9].

```
    Class: s:Margherita
        SubClassOf:
a1:         b:NamedPizza,
a2:         b:hasPizzaTopping some b:MozzarellaTopping,
a3:         b:hasPizzaTopping some b:TomatoTopping,
a4:         b:hasPizzaTopping only
                (b:MozzarellaTopping
                 or b:TomatoTopping)
        DisjointWith:
            s:FruttiDiMare
```

Codebeispiel 4.34: Richtige Definition mit mehreren notwendigen Axiomen

Im folgenden Codebeispiel 4.35 wurden dagegen alle Axiome zu `a1` zusammengefasst. Da einzelne notwendige Axiome implizit durch `and` verbunden sind ergibt das eine gültige Lösung. Das Ergebnis: Die Klasse erscheint an den gleiche Positionen und hat die gleichen benannten, aber andere anonyme Superklassen. Ein durch Verbinden der Axiome mit `or` entstandene Klasse würde nicht einmal mehr als Pizza klassifiziert.

```
    Class: s:MargMerged
        SubClassOf:
a1:         b:NamedPizza
             and (b:hasPizzaTopping some b:MozzarellaTopping)
             and (b:hasPizzaTopping some b:TomatoTopping)
             and (b:hasPizzaTopping only
                 (b:MozzarellaTopping
                  or b:TomatoTopping))
```

Codebeispiel 4.35: Richtige Definition mit einem notwendigen Axiom

```
    Class: s:Margherita
        SubClassOf:
a1:         b:NamedPizza
             or (b:hasPizzaTopping some b:MozzarellaTopping)
```

```
    or (b:hasPizzaTopping some b:TomatoTopping)
    or (b:hasPizzaTopping only
        (b:MozzarellaTopping
         or b:TomatoTopping))
```

Codebeispiel 4.36: Falsche Definition mit einem notwendigen Axiom

Symptom K2 (anders hinreichend)**.** Die Klasse k_{mg} oder k_s wurde nur mittels *EquivalentTo* definiert, die andere Klasse nur mittels *SubClassOf*.

Erkannt werden kann dies an der Definition der Klassen: Die Axiome einer Klasse enthalten keinen `SubClassOf`-Ausdruck, während die Axiome der anderen Klasse ohne Verwendung eines `EquivalentClasses`-Ausdruck formuliert wurden.

Im Beispiel wurde die linke Klasse im Gegensatz zur rechten mittels *EquivalentTo* definiert. Aus diesem Grund ist das Symptom erfüllt.

```
Class: CheesyPizza
    EquivalentTo:
        ...
```

```
Class: CheesyPizza
    SubClassOf:
        ...
```

Symptom K3 (*or*)**.** In mindestens einem Axiom von k_s wird *or* verwendet. Erkennbar ist dies an einem in den Klassenaxiomen enthaltenen `ObjectUnionOf`-Ausdruck.

Axiom a_1 im Beispiel enthält den `ObjectUnionOf`-Ausdruck `MozzarellaTopping or TomatoTopping`.

```
a1:     hasPizzaTopping only (MozzarellaTopping or TomatoTopping)
```

Symptom K4 (*some* mit *not*)**.** In mindestens einem Axiom von k_s werden sowohl *some* als auch *not* gemeinsam verwendet. Dies wird an in den Klassenaxiomen gleichzeitig enthaltenen `ObjectComplementOf`- und `ObjectSomeValuesFrom`-Ausdrücken erkannt.

Axiom a_1 im Beispiel enthält im `ObjectComplementOf`-Ausdruck `not (...)` den `ObjectSomeValuesFrom`-Ausdruck `hasPizzaTopping some FishTopping`.

```
a1:     (not (hasPizzaTopping some FishTopping))
```

Symptom K5 (*only*)**.** In mindestens einem in NNF betrachteten Axiom von k_s wird *only* verwendet, was am Vorkommen eines `ObjectAllValuesFrom`-Ausdrucks in den Klassenaxiomen erkannt werden kann.

Axiom a_1 im Beispiel enthält den `ObjectAllValuesFrom`-Ausdruck `hasPizzaTopping only (...)`.

```
a1:     (hasPizzaTopping only (not (MeatTopping)))
```

Symptom K6 (kein *only*)**.** In keinem der in NNF betrachteten A_{k_s} wird ein `ObjectAllValuesFrom`-Ausdruck verwendet.

Kein Axiom des folgenden Beispiels enthält einen `ObjectAllValuesFrom`-Ausdruck.

```
    Class: Margherita
        SubClassOf:
a1:         NamedPizza,
a2:         hasPizzaTopping some MozzarellaTopping,
a3:         hasPizzaTopping some TomatoTopping
```

Symptom K7 (*some*)**.** In mindestens einem der in NNF betrachteten A_{k_s} wird ein `ObjectSomeValuesFrom`-Ausdruck verwendet.

Axiom a_1 im Beispiel ist ein `ObjectSomeValuesFrom`-Ausdruck.

```
a1:         hasPizzaTopping some TomatoTopping
```

Symptom K8 (kein *some*)**.** In keinem der in NNF betrachteten A_{k_s} wird ein `ObjectSomeValuesFrom`-Ausdruck verwendet.

Keines der Axiome im Beispiel enthält einen `ObjectSomeValuesFrom`-Ausdruck.

```
    Class: CheesyPizza
        EquivalentTo:
a1:         Pizza
             and (hasPizzaTopping only CheeseTopping)
```

Symptom K9 (Werte- / Zielbereich)**.** Mindestens ein Axiom von k_s verwendet eine OWL-Property, deren Werte- oder Zielbereich eine Subklasse von *owl:Thing* ist.

Im folgenden Beispiel verwendet die betrachtete Klasse *Icecream* in a_1 die Property *hasIngredient*, deren Wertebereich in a_2 auf *Food* beschränkt ist. Da *Food* laut a_3 eine Subklasse von *owl:Thing* ist gilt das Symptom als erfüllt.

```
    Class: Icecream
        SubClassOf:
a1:         hasIngredient some FruitTopping

    ObjectProperty: hasIngredient
        Domain:
a2:         Food

    Class: Food
        SubClassOf:
a3:         owl:Thing
```

Symptom K10 (trivial erfüllbar)**.** Die Klassenaxiome von k_s werden durch eine leere Klasse erfüllt.

Ein Beispiel wurde bereits bei Fehlertyp 8 auf Seite 28 besprochen.

Position der Klasse

Symptome bezüglich der Position der Klasse geben über die Folgen ihrer Definition Auskunft, also über die vom Reasoner interpretierten Axiome. Der Inhalt der Definitionen selbst ist hier unwesentlich, da er bereits von Symptomen zu Klassenaxiomen erfasst wird.

In den Abbildungen wird die Zugehörigkeit zu einer Ontologie mittels Präfix ausgedrückt, $Margherita_s$ wird daher zu *s:Margherita*. Zur Erinnerung sei kurz auf die Präfixe hingewiesen: s steht für die vom Studenten formulierte Ontologie, m für die Musterlösung und b für die ausgegebene Basisontologie. *owl* wird für Konstanten von OWL verwendet.

Symptom P1 (*owl:Nothing*)**.** Die k_s wird als logisch äquivalent mit *owl:Nothing* klassifiziert, da sie in dieser Ontologie logisch nicht möglich ist. Abbildung 4.2 zeigt ein Beispiel.

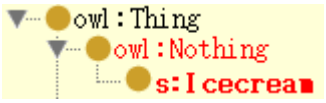

Abbildung 4.2.: *Icecream* wurde als nicht möglich klassifiziert

Symptom P2 (anderer Teilbaum)**.** k_s befindet sich in einem anderen Teilbaum der Hierarchie als k_{mg}, was an der fehlenden Super-/Subklassen-Beziehung zwischen den beiden Klassen erkannt werden kann.

Im Beispiel wurde die Klasse $Margherita_m$ als $NamedPizza$ definiert, die abgegebene $Margherita_s$ jedoch als $ItalianPizza$. Da die beiden Superklassen auf gleicher Ebene der Hierarchie liegen kann zwischen $Margherita_m$ und $Margherita_s$ keine Super-/Subklassen-Beziehung hergestellt werden. Abbildung 4.3 zeigt die klassifizierte Hierarchie.

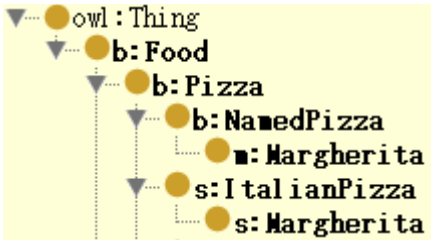

Abbildung 4.3.: $Margherita_m$ und $Margherita_s$ liegen in verschiedenen Teilbäumen

Symptom P3 (zu speziell)**.** Bei einem hinreichend definierten k_{mg} ist k_s eine Subklasse von k_{mg}. Im Fall eines nur mit notwendigen Axiomen definierten k_{mg} befinden sich beide Klassen auf gleicher Ebene, k_s ist aber mittels strengerer Axiome definiert.

Im Beispiel wurden beide Varianten von $CheesyPizza$ als hinreichend formuliert. Da die Definition von $CheesyPizza_s$ jedoch strenger ist wurde sie, wie in Abbildung 4.4 ersichtlich, als Subklasse von $CheesyPizza_m$ klassifiziert.

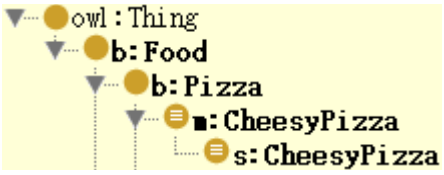

Abbildung 4.4.: $CheesyPizza_s$ wurde als Subklasse von $CheesyPizza_m$ klassifiziert

Symptom P4 (zu allgemein)**.** Bei einem hinreichenden k_s ist k_{mg} eine Subklasse von k_s, im Fall eines nur notwendig definierten k_s befindet sich k_{mg} zwar auf der gleichen Ebene, k_s ist aber mittels lockerer Axiome definiert.

Im Gegensatz zum vorigen Beispiel sind hier beide Klassen nur mit notwendigen Axiomen definiert. Wie in Abbildung 4.5 sichtbar liegen sie daher auf der gleichen Ebene. Erst eine Analyse der Klassenaxiome zeigt, dass die Axiome von $Margherita_m$ lockerer sind als die von $Margherita_s$.

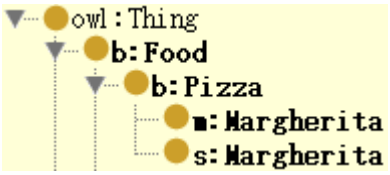

Abbildung 4.5.: $Margherita_m$ und $Margherita_s$ liegen auf gleicher Ebene

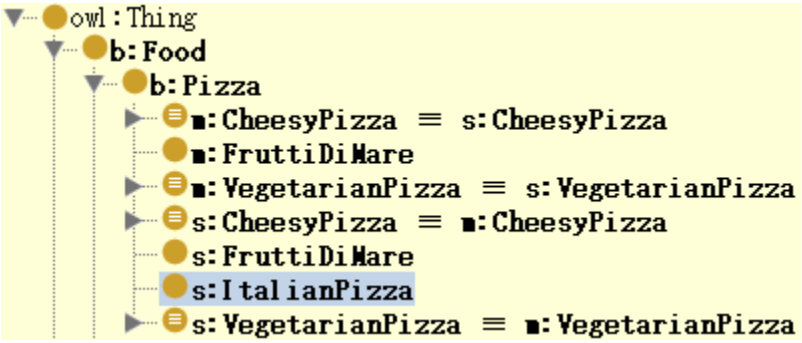

Abbildung 4.6.: An der Position von $ItalianPizza_s$ befindet sich kein $ItalianPizza_m$

Symptom P5 (wo keine k_{mg})**.** An keiner Position von k_s befindet sich eine k_{mg}.

In dem in Abbildung 4.6 gezeigten Beispiel kommt $ItalianPizza_s$ nur an einer Stelle vor. An dieser Position befindet sich keine $ItalianPizza_m$.

Symptom P6 (nicht besetzt)**.** k_s fehlt an mindestens einer der Positionen von k_{mg}.

An der in Abbildung 4.7 betrachteten Position von $Margherita_m$ befindet sich keine $Margherita_s$.

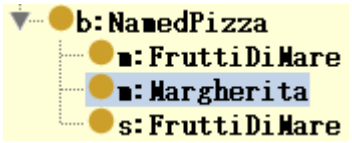

Abbildung 4.7.: An der Position von $Margherita_m$ befindet sich keine $Margherita_s$

Symptom P7 (andere Subklassen)**.** Die Subklassen von k_s unterscheiden sich von den Subklassen von k_{mg}.

Die im Beispiel in Abbildung 4.8 verglichenen vegetarischen Pizzen unterscheiden sich unter anderem in Anzahl und Typ ihrer Subklassen.

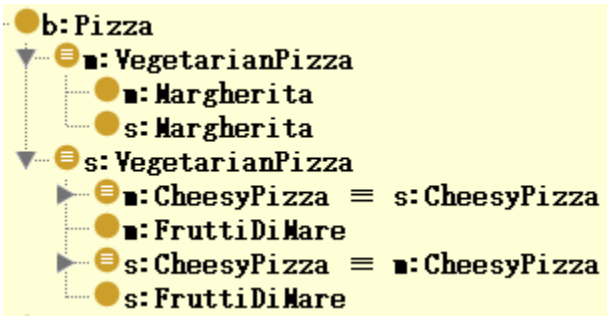

Abbildung 4.8.: Die vegetarischen Pizzen unterscheiden sich in ihren Subklassen

In gesamter Ontologie

Symptome dieser Gruppe betreffen die gesamte Ontologie.

Symptom O1 (alle Positionen gleich)**.** Jede von k_s eingenommene Position wird auch von k_{mg} besetzt und umgekehrt. Dies ist nur möglich wenn die Menge der Superklassen beider Klassen gleich ist.

Im Beispiel tritt $Margherita_s$ genau an den gleichen Positionen der Klassifikationshierarchie wie $Margherita_m$ auf. Wie in Abbildung 4.9 zu sehen haben deshalb beide Konzepte die selben Superklassen $NamedPizza$, $CheesyPizza$ und $VegetarianPizza$.

Symptom O2 (Klasse unauffindbar)**.** Es konnte keine zu k_{mg} namens-, positions-, oder inhaltsgleiche k_s gefunden werden.

Das Beispiel von Fehlertyp 3 auf Seite 25 gilt auch in diesem Fall.

Symptom O3 (Basisdaten ungleich)**.** Die von O_m und O_s referenzierten Basisontologien O_b sind unterschiedlich.

Ein ausführliches Beispiel wurde bereits bei Fehlertyp 7 auf Seite 27 diskutiert.

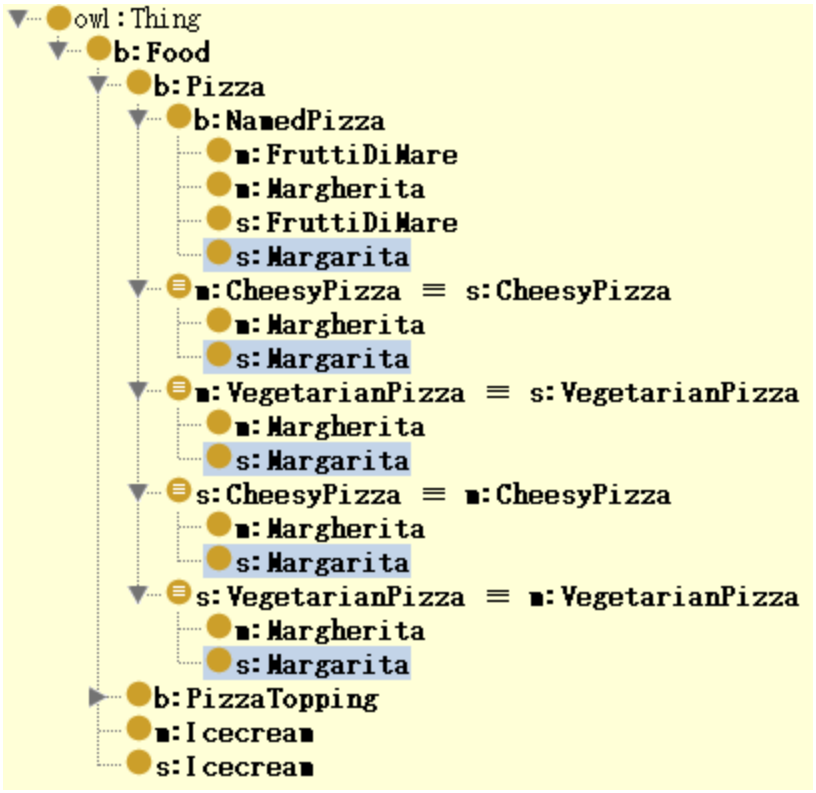

Abbildung 4.9.: $Margherita_m$ und $Margherita_s$ haben die selben Superklassen

4.5.2. Gruppieren von Symptomen zu Fehlertypen und Themenbereichen

Nach dem Feststellen der Symptome werden diese den jeweiligen Fehlertypen zugeordnet, die wiederum nach Themenbereich zusammengefasst werden können. Die aus der Sicht der Musterlösung formulierte Symptom-Fehlertyp-Matrix auf Seite 46 zeigt, welche Symptome welchen Fehlertypen zugeordnet werden können. Als Erläuterung werden in Abbildung 4.11 auf Seite 47 alle Fehlertypen aufgelistet. Die Zuordnung zu Themenbereichen erfolgt analog der Gruppen in Kapitel 4.4 und ist ebenfalls aus der Tabelle ersichtlich.

4.5.3. Ablauf der Analyse

Alle Symptome betrachten entweder Klassen der Abgabe, geforderte Klassen oder Paare aus abgegebenen und geforderten Klassen. Zusätzlich basieren manche Symptome auf den Ergebnissen anderer, weshalb die Reihenfolge ihrer Prüfung relevant ist. Der folgende Ablauf ist in Abbildung 4.10 auf Seite 45 zusammengefasst, wobei zwecks Übersichtlichkeit der Datenfluss nicht gezeigt wird.

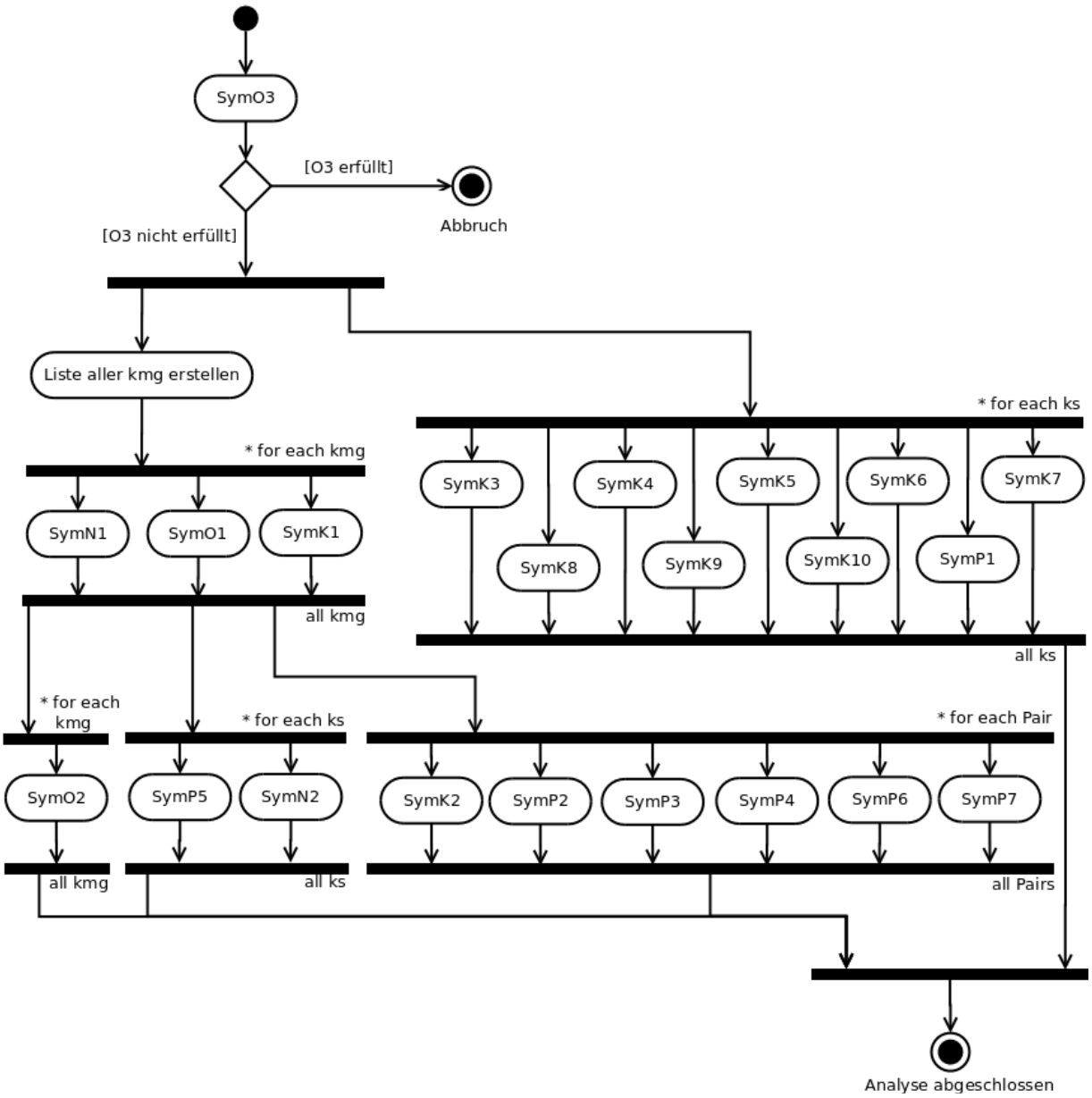

Abbildung 4.10.: Ablauf der Analyse

Symptom	Fehlertyp																
	Angabe							Logik						OWL			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Name der Klasse																	
N1 ähnlich						○											
N2 unbekannt		●				○											
Klassenaxiome																	
K1 inhaltsgleich						●											
K2 anders hinreichend																●	
K3 *or*													●				
K4 *some* mit *not*								○	●								
K5 *only*								○				●					
K6 kein *only*														●			
K7 *some*														●			
K8 kein *some*								○				●					
K9 Werte- / Zielbereich															●		
K10 trivial erfüllbar								●									
Position der Klasse																	
P1 *owl:Nothing*										●							
P2 anderer Teilbaum	●											○			○		
P3 zu speziell					●							○			○		
P4 zu allgemein				●							●	○	●				
P5 wo keine k_{mg}		●										○					
P6 nicht besetzt												○		●			●
P7 andere Subklassen									●			○					
In gesamter Ontologie																	
O1 alle Positionen gleich						●											
O2 Klasse unauffindbar			●														
O3 Basisdaten ungleich							●										
Verwechslungsgefahr																	
mit Fehlertyp	–	–	–	11	–	–	–	–	–	–	4	–	–	–	–	–	–

● ... Symptom vorhanden
○ ... Symptom an mindestens einer dieser Stellen vorhanden
Leer: Symptom für Fehlertyp nicht relevant

Tabelle 4.1.: Symptom-Fehlertyp-Matrix

1	Klasse auf falscher Ebene definiert
2	Klasse zusätzlich definiert
3	Geforderte Klasse nicht definiert
4	Konzept unterspezifiziert
5	Konzept überspezifiziert
6	Namenskonvention nicht eingehalten
7	Basisdaten verändert
8	Bedingung durch leere Menge trivial erfüllbar
9	Verneinung falsch positioniert
10	Widerspruch innerhalb der Ontologie
11	Implizites Wissen nicht explizit ausgedrückt
12	Allquantor statt Existenzquantor verwendet
13	Probleme mit logischem *Oder*
14	*Open World Assumption* bei Klassen nicht beachtet
15	Einschränkungen von Werte- und Zielbereich sind auch Axiome
16	*Open World Assumption* bei Properties nicht beachtet
17	Überlappung von Klassen nicht beachtet

Abbildung 4.11.: Fehlertypen

Vergleich der Basisontologien (O_b)

Die in O_s enthaltenen Verweise auf Klassen aus O_b werden auf Konsistenz mit den tatsächlichen Klassen geprüft. Bei Inkonsistenz wird die gesamte Analyse abgebrochen, da entweder die Basisontologie verändert oder von der Lehrveranstaltungsleitung eine falsche Ontologie ausgegeben wurde. In beiden Fällen ist eine weitere Verarbeitung nicht sinnvoll [23, 3], im letzteren Fall soll zusätzlich ehestmöglich die korrekte Basisontologie ausgegeben werden.

Nach diesem Schritt kann Symptom O3 entschieden werden.

Suche nach geforderten Klassen (k_{mg}) und Paarbildung

Die geforderte Klassen werden anhand des Kommentars *gefordert* gefunden. Anschließend wird jede k_{mg} mit jeder k_s verglichen, ein erfolgreicher Vergleich führt zu einer Paarbildung. Jede Klasse kann in mehreren Paaren vorkommen. (1) Bei der *Suche über den Namen* werden alle Symptome zum Namen der Klasse geprüft, also N1 und N2. (2) Im nächsten Schritt, der *Suche über die Position*, werden mit dem Symptom O1 alle Positionen von k_{mg} in der Klassenhierarchie durchsucht. (3) Die letzte Möglichkeit zur Paarbildung bietet die *Suche über die Klassenaxiome*, zu der das Symptom K1 verwendet wird.

Wenn bei keinem der Vergleiche eine k_s zum aktuellen k_{mg} gefunden wurde ist das Symptom O2 erfüllt, falls ein Paar gefunden wurde ist O2 nicht erfüllt.

Analyse der gefundenen Paare

In diesem Teil werden alle noch nicht geprüften paarbezogenen Symptome analysiert, also K2, P2, P3, P4, P6 und P7.[3]

Analyse aller Klassen der Abgabe

Die übrigen Symptome betreffen k_s unabhängig von Paarbildungen und werden in diesem Schritt geprüft. Konkret sind das K3, K4, K5, K6, K7, K8, K9, K10, P1 und P5.

Ergebnisse

Falls die Basisontologien übereinstimmen liefert die Analyse die Menge an Paaren k_{mg} und k_s, die irgendeine Form von Gleichheit aufweisen. Dabei sind folgende Aussagen möglich:

In Paaren, deren k_{mg} und k_s *gleich* sind wurde k_s richtig definiert. In Paaren mit nur teilweise gleichen Klassen wurde k_s möglicherweise teilweise korrekt definiert. Es kann sich aber auch nur um einen losen Zusammenhang handeln, etwa wenn beide Klassen verschiedene vegetarische Pizzen beschreiben. Schließlich werden jene k_{mg} vorgemerkt, die in keinen Paarbeziehungen vorkommen.

Zu jedem Paar werden folgende Daten aufgezeichnet: k_{mg}, k_s und Details zur Art der Gleichheit. Zu jeder k_s sind die Ergebnisse aller Symptomtests (außer O3, das rein die Basisontologie behandelt) vorhanden. Die gefundene Ergebnisse werden in weiteren Schritten zu Fehlertypen, Themenbereichen und schließlich zur Rückmeldung an den Studenten (siehe 4.6) verdichtet.

Ablauf am Beispiel

Das gewählte Beispiel zeigt die Analyse eines einzigen Fehlers bei der Definition einer OWL-Klasse der Studentenlösung. Gewählt wurden der Fehlertyp 12 (*Allquantor statt Existenzquantor verwendet*) aus dem Themenbereich „Logik“ (4.4.3) und die Klasse *CheesyPizza*. Um diesen Fehler zu finden müssen neben Symptomen bezüglich der Positionen der Klassen auch solche zu Klassenaxiomen erkannt werden. Die Symptome betreffen sowohl einzelne Klassen als auch Paare und sind teilweise Alternativen (erkennbar am Symbol ∘ in Tabelle 4.1). Um das Beispiel übersichtlicher zu gestalten werden nur Schritte besprochen, welche die Klasse *CheesyPizza* der Musterlösung oder Abgabe betreffen.

Zur Erinnerung: Der Fehlertyp 12 tritt auf, wenn eine Abgabe statt eines Existenzquantor einen Allquantor verwendet. Die Codebeispiele zeigen den wesentlichen Unterschied zwischen Studenten- und Musterlösung. Der hier händisch durchgeführte Ablauf entspricht dem mit der entwickelten Software durchgeführten „Test 12“ im Anhang.

[3] als Kurzreferenz der einzelnen Symptome bietet sich Tabelle 4.1 auf Seite 46 an

```
Class: CheesyPizza
    EquivalentTo:
        Pizza
         and (hasPizzaTopping only CheeseTopping)
```

Codebeispiel 4.37: Falsch: Käsepizza akzeptiert *nur* Käsebelag (und wird daher auch als vegetarisch klassifiziert)

```
Class: CheesyPizza
    EquivalentTo:
        Pizza
         and (hasPizzaTopping some CheeseTopping)
```

Codebeispiel 4.38: Richtig: Käsepizza benötigt *mindestens einen* Käsebelag

Die Analyse beginnt mit dem Vergleich der Basisontologien. Dieser zeigt keine Abweichungen, daher ist O3 negativ und die Analyse wird fortgesetzt. Im nächsten Schritt werden die geforderten Klassen gesucht, wobei für dieses Beispiel angenommen wird, dass nur $CheesyPizza_m$ als „gefordert“ markiert wurde. Die geforderte Klasse ist namensgleich mit $CheesyPizza_s$, was zur Bildung des ersten Paars p_1 führt. Es werden keine Klassen mit ähnlichen Namen gefunden, damit ist N1 negativ.

Vor der Verwendung positionsbezogener Symptome müssen Studenten- und Musterlösung vereinigt und klassifiziert werden. Die sich ergebende Hierarchie ist in Abbildung 4.12 sichtbar. $CheesyPizza_m$ und $VegetarianPizza_s$ weisen die gleichen Superklassen auf und bilden daher das Paar p_2, dass das Symptom O1 erfüllt. Da die Klassenaxiome keiner Kombination von k_{mg} und k_s völlig übereinstimmen ist K1 negativ und die Paarbildung abgeschlossen.

Alle Arten von Gleichheit wurden analysiert, daher kann O2 entschieden werden. Es wurde für alle geforderten Klassen eine Entsprechung in der Abgabe gefunden, somit ist dieses Symptom negativ. Wie in Abbildung 4.12 ebenfalls zu erkennen teilt $CheesyPizza_s$ nicht alle Positionen der geforderten Klasse $CheesyPizza_m$, P5 ist deshalb nicht erfüllt. Sie ist Teil von p_1 wegen einer Namensgleichheit, der Name ist daher bekannt und N2 negativ.

Die Klassenaxiome von $CheesyPizza_s$ enthalten `only`, weshalb K5 positiv und K6 negativ ist, aber kein `some`, was zu positivem K8 und negativen K7 und K4 führt. Ziel- und Wertebereich werden verwendet, daher ist K9 positiv. `or` kommt nicht vor, weshalb K3 negativ ist. Da die Klasse weder trivial erfüllbar noch unmöglich ist sind sowohl K10 als auch P1 negativ[4].

Die Klasse $VegetarianPizza_s$ hat an einer Paarbildung teilgenommen und wird hier daher ebenfalls behandelt. Die Axiome der Klasse enthalten kein `only`, weshalb K5 negativ und K6 positiv ist, dafür `some`, was zu einem negativen K8 und positiven K7 und K4 führt. Ziel- und Wertebereich werden verwendet, daher ist K9 positiv. `or` kommt nicht vor, weshalb K3 negativ ist. Die Klasse ist trivial erfüllbar, aber nicht unmöglich, daher sind K10 positiv und P1 negativ.

[4] als Kurzreferenz für Symptome bietet sich wiederum Tabelle 4.1 auf Seite 46 an, für Fehlertypen die Abbildung 4.11 auf Seite 47

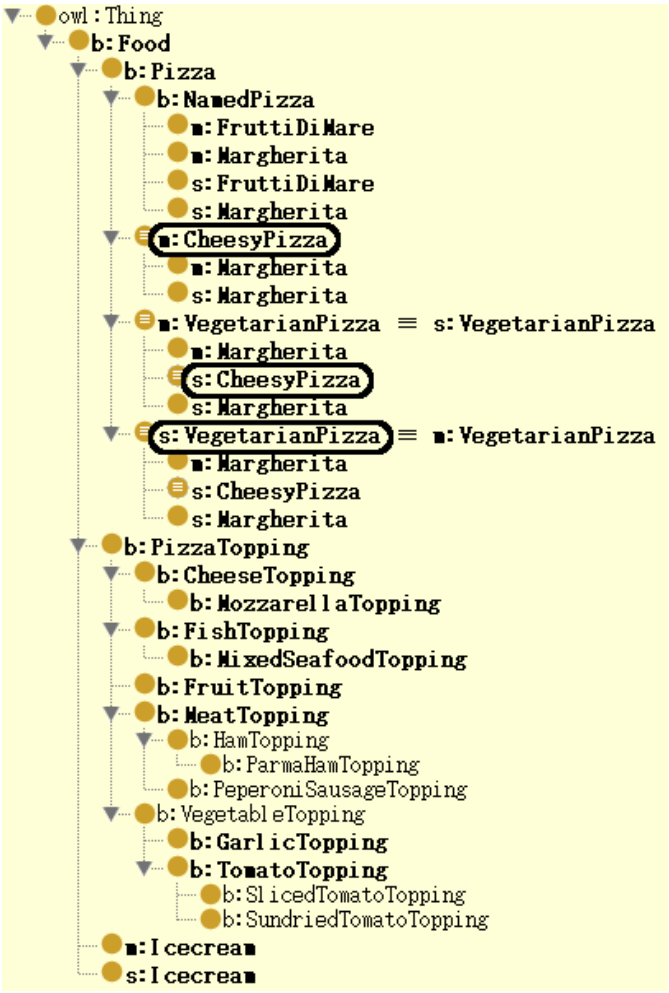

Abbildung 4.12.: Klassenhierarchie der vereinigten Studenten- und Musterlösung

Nun werden die gefundenen Paare auf weitere Symptome geprüft. Bei p_1 sind beide Klassen gleich hinreichend (K2 negativ) und befinden sich in verschiedenen Teilbäumen der Hierarchie (P2 positiv). Die Klasse k_s ist weder zu speziell noch zu allgemein definiert (P3 und P4 negativ), fehlt aber an mindestens einer Position von k_{mg} (P6 positiv) und die Subklassen der beiden Klassen unterscheiden sich (P7 positiv).

Auch im zweiten gefundenen Paar p_2 sind die Klassen gleich hinreichend (K2 negativ) und sie befinden sich in verschiedenen Teilbäumen (P2 positiv). Die Klasse k_s ist ebenfalls weder zu speziell noch zu allgemein definiert (P3 und P4 negativ), teilt in diesem Fall aber alle Positionen der Klasse k_{mg} (P6 negativ). Die Subklassen der beiden unterscheiden sich ebenfalls (P7 positiv).

Zur Auswertung werden die erfassten Symptome in Bereichen gesammelt. Dieser Schritt ist notwendig, da zwar manche Symptome nur einmal pro Ontologie, andere jedoch für jede OWL-Klasse oder jedes Paar erfasst werden. Tabelle 4.2 zeigt die Zuordnung der

Symptome zu den Bereichen. Dabei steht jeder mit π bezeichnete Bereich für eines der gefundenen Paare, Symptome im Bereich ω gelten für die gesamte Ontologie. *Nicht anwendbare Symptome* werden von anderen Bereichen abgedeckt, *positive* Symptome wurden gefunden, *negative* nicht gefunden. Das Ergebnis jedes Bereichs ergibt sich aus der Summe der Ergebnisse seiner Komponenten, wobei eine Klasse, ein Paar oder die Ontologie als Komponente auftreten kann.

Bereich	N1	N2	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	P1	P2	P3	P4	P5	P6	P7	O1	O2	O3
Bereich ω: gesamte Ontologie																						
O_s																						−
k_{mg1}																					−	
O_s OR k_{mg1}																					−	−
Bereich π_1: An p_1 beteiligte k_s																						
p_1	−		−	−										+	−	−		+	+	−		
k_{s1}		−			−	−	+	−	−	+	+	−	−				−					
p_1 OR k_{s1}	−	−	−	−	−	−	+	−	−	+	+	−	−	+	−	−	−	+	+	−		
Bereich π_2: An p_2 beteiligte k_s																						
p_2	−		−	−										+	−	−		−	+	+		
k_{s2}		+			−	+	−	+	+	−	+	+	−				−					
p_2 OR k_{s2}	−	+	−	−	−	+	−	+	+	−	+	+	−	+	−	−	−	−	+	+		

k_{s1} ... $CheesyPizza_s$
k_{s2} ... $VegetarianPizza_s$
k_{mg1} ... $CheesyPizza_m$

\+ ... Symptom positiv
− ... Symptom negativ
Leer: Symptom nicht anwendbar

Tabelle 4.2.: Gefundene Symptome pro Bereich

Die Weiterverarbeitung der Symptome zu gefundenen Fehlertypen erfolgt ebenfalls in den einzelnen Bereichen. Um die Darstellung der Auswertung konsistent mit der Symptom-Fehlertyp-Matrix des Konzepts auf Seite 46 zu halten, wurde die dortige Legende erweitert: Nicht gefundene Symptome behalten ihre Kreisform, bei gefundenen Symptomen vergrößert sich das Symbol zu einem Quadrat (vgl. Tabelle 4.5). Die Ergebnisse der einzelnen Bereiche sind auf den nächsten Seiten in den Symptom-Fehlertyp-Matrizen 4.4, 4.5 und 4.6 dargestellt. Um die vollständige Auswertung zu erhalten müssen diese Zwischenergebnisse nur noch aufsummiert werden.

Wie in Tabelle 4.3 erkennbar wurde der gesuchte Fehlertyp 12 erfolgreich gefunden. Weiters diagnostiziert wurden die Fehlertypen 1, 8, 9, 15 und 17. Typ 1 (Klasse auf falscher Ebene definiert) schlägt auf Grund eines Folgefehlers der Abgabe an und Typ 8 (Bedingung durch leere Menge trivial erfüllbar) weist korrekterweise auf ein auch in der Musterlösung vorhandenes Problem hin. Fehlertyp 9 (Verneinung falsch positioniert) ist ebenfalls ein sinnvoller Hinweis, falls der Student $\exists$ durch $\forall$ in Kombination mit Verneinung umschreiben wollte, dabei aber Fehler machte. Wie alle OWL-bezogenen Fehlertypen stellen die Typen 15 (Einschränkungen von Werte- und Zielbereich sind auch

Axiome) und 17 (Überlappung von Klassen nicht beachtet) Vermutungen hinsichtlich der Absicht beim Modellieren dar, was sich in einer größeren Unschärfe in der Auswertung niederschlägt. Deshalb ist es sinnvoll, diesen Fehlertypen eine niedrigere Priorität in der Rückmeldung zuzuweisen.

Bereich	Fehlertyp[5]																
	Angabe							Logik						OWL			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
ω			−				−										
π_1	+	−		−	−	−		−	−	−	−	+	−	−	+	−	+
π_2	+	−		−	−	−		+	+	−	−	−	−	−	+	−	−
Gesamt																	
$\omega\ OR\ \pi_1\ OR\ \pi_2$	+	−	−	−	−	−	−	+	+	−	−	+	−	−	+	−	+

+ ... Fehlertyp erkannt
− ... Fehlertyp nicht erkannt
Leer: Fehlertyp hier nicht anwendbar

Tabelle 4.3.: Im Beispiel gefundene Fehlertypen

Symptom	Fehlertyp[5]																
	Angabe							Logik						OWL			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
O2 Klasse unauffindbar			•														
O3 Basisdaten ungleich							•										
Auswertung																	
Fehlertyp erkannt?			−				−										

■ ... Symptom positiv und an dieser Stelle relevant
□ ... Symptom positiv und an mindestens einer dieser Stellen relevant
• ... Symptom negativ und an dieser Stelle relevant
○ ... Symptom negativ und an mindestens einer dieser Stellen relevant
+ ... Fehlertyp erkannt
− ... Fehlertyp nicht erkannt
Leer: Feld hier nicht relevant

Tabelle 4.4.: Symptom-Fehlertyp-Matrix für Bereich ω

[5]die Nummern der Fehlertypen werden in Abbildung 4.11 auf Seite 47 erläutert

Symptom	Fehlertyp																
	Angabe							Logik						OWL			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Name der Klasse																	
N1 ähnlich						○											
N2 unbekannt		●				○											
Klassenaxiome																	
K1 inhaltsgleich						●											
K2 anders hinreichend																●	
K3 *or*													●				
K4 *some* mit *not*								○	●								
K5 *only*								□				■					
K6 kein *only*														●			
K7 *some*														●			
K8 kein *some*								□				■					
K9 Werte- / Zielbereich															■		
K10 trivial erfüllbar								●									
Position der Klasse																	
P1 *owl:Nothing*										●							
P2 anderer Teilbaum	■											□			□		
P3 zu speziell					●							○			○		
P4 zu allgemein				●							●	○	●				
P5 wo keine k_{mg}		●										○					
P6 nicht besetzt												□		■			■
P7 andere Subklassen									■			□					
In gesamter Ontologie																	
O1 alle Positionen gleich						●											
Auswertung																	
Fehlertyp erkannt?	+	−		−	−	−		−	−	−	−	+	−	−	+	−	+

■ ... Symptom positiv und an dieser Stelle relevant
□ ... Symptom positiv und an mindestens einer dieser Stellen relevant
● ... Symptom negativ und an dieser Stelle relevant
○ ... Symptom negativ und an mindestens einer dieser Stellen relevant
\+ ... Fehlertyp erkannt
− ... Fehlertyp nicht erkannt
Leer: Feld hier nicht relevant

Tabelle 4.5.: Symptom-Fehlertyp-Matrix für Bereich π_1

Symptom	Fehlertyp																
	Angabe							Logik						OWL			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Name der Klasse																	
N1 ähnlich						○											
N2 unbekannt		■				□											
Klassenaxiome																	
K1 inhaltsgleich						•											
K2 anders hinreichend																•	
K3 *or*													•				
K4 *some* mit *not*								□	■								
K5 *only*								○				•					
K6 kein *only*														■			
K7 *some*														■			
K8 kein *some*								○				•					
K9 Werte- / Zielbereich															■		
K10 trivial erfüllbar								■									
Position der Klasse																	
P1 *owl:Nothing*										•							
P2 anderer Teilbaum	■											□			□		
P3 zu speziell					•							○			○		
P4 zu allgemein				•							•	○	•				
P5 wo keine k_{mg}		•										○					
P6 nicht besetzt												○		•			•
P7 andere Subklassen									■			□					
In gesamter Ontologie																	
O1 alle Positionen gleich						■											
Auswertung																	
Fehlertyp erkannt?	+	−		−	−	−		+	+	−	−	−	−	−	+	−	−

■ ... Symptom positiv und an dieser Stelle relevant
□ ... Symptom positiv und an mindestens einer dieser Stellen relevant
• ... Symptom negativ und an dieser Stelle relevant
○ ... Symptom negativ und an mindestens einer dieser Stellen relevant
\+ ... Fehlertyp erkannt
− ... Fehlertyp nicht erkannt
Leer: Feld hier nicht relevant

Tabelle 4.6.: Symptom-Fehlertyp-Matrix für Bereich π_2

4.6. Gestaltung der Rückmeldung

Die Gestaltung der Rückmeldung orientiert sich an den Abgabemodi des eTutors und unterscheidet deshalb zwischen Übungsmodus und Lernfortschrittsmodus [17, 11]. Die Analyse wird bei jedem Abgabemodus vollständig durchgeführt, die dabei erzeugte Ausgabe aber entsprechend der Anforderungen gefiltert.

Die Mehrsprachigkeit dieser Ausgabe [14, 63] wird durch die Verwendung von Platzhaltern erreicht, die durch Phrasen der gewünschten Sprache ersetzt werden.

4.6.1. Übungsmodus

Bei der Abgabe im Übungsmodus erfolgt keine Bewertung, es werden rein Hinweise auf der Feedback-Tiefe *Hint* [17, 54] gegeben, also Anmerkungen zu falschen Lösungsteilen, ohne dass Teile der Musterlösung gezeigt werden. Daher kann die Rückmeldung sowohl auf der Ebene der Fehlertypen als auch nach Themenbereichen gesammelt erfolgen. Bei der Rückmeldung zu einzelnen Fehlertypen wird auf die betroffene OWL-Klasse verwiesen, um eine Überarbeitung der Abgabe zu erleichtern. Die Fehlermeldung sollten dabei nach ihrer Priorität gereiht werden, um Fehlertypen mit geringerer Unschärfe attraktiver zu platzieren.

Der allgemeinere, auf Themenbereiche bezogene Teil der Rückmeldung dient weniger der unmittelbaren Verbesserung der Abgabe als dem Erkennung von Wissenslücken der Studierenden. Zu diesem Zweck kann gezielt Information zu den erkannten Bereichen angeboten werden, etwa in Form von Verweisen auf erklärende Artikel oder entsprechende Passagen im Vorlesungsskriptum. Die Texte können auch allgemeine Fragen wie die Vorgehensweisen beim Modellieren [29, 4ff] oder das Definieren von Klassen [29, 12ff] behandeln.

4.6.2. Lernfortschrittsmodus

Im Gegensatz zum Übungsmodus wird die Abgabe bewertet und zur Dokumentation der Bewertung dauerhaft gespeichert. Eine mehrfache Abgabe bietet die Möglichkeit, die erzielte Bewertung zu verbessern [17, 11].

Der Hauptzweck des neuen Expertenmoduls ist die Unterstützung der Studenten beim Erarbeiten ihrer Lösungen. Daher liegt der Fokus dieser Arbeit vor allem auf der Erstellung hochwertiger Rückmeldungen im Übungsmodus und nur wenig auf der Bewertung der Lösung. Als Folge wird die abgegebene Ontologie im Abgabemodus nur als insgesamt *mit der Musterlösung übereinstimmend* („richtig“) oder *nicht übereinstimmend* („falsch“) beurteilt.

4.7. Einordnung des neuen Ansatzes

Vergleicht man das in dieser Arbeit entwickelte Konzept mit den im Stand der Technik vorgestellten Ansätzen, so werden folgende Aspekte deutlich: Der neue Ansatz zur

Erkennung von Fehlern beim Erlernen der Ontologiemodellierung kombiniert Vergleiche zweier Ontologien mit der Analyse einer einzelnen Ontologie.

Er findet durch Verwendung von Prinzipien des Ontology Alignment die von der Bedeutung der Musterlösung abweichenden Klassen der Abgabe. Zusätzlich wurden die an der Universität Manchester formulierten Verständnisprobleme so weiterentwickelt, dass sie automatisiert erkannt werden können.

Wie in Tabelle 4.7 ersichtlich erzeugt der neue Ansatz Rückmeldungen zu unerfüllbaren Klassen und semantischen Fehlern. Sowohl eine inkonsistente Ontologie als auch syntaktische Fehler werden vom Reasoner zwar erkannt, fließen aber nicht als eigene Fehlertypen in die Rückmeldung ein. Erstere können außerdem erst durch Definition von OWL-Individuen entstehen, nicht durch Konzepte. Etwaige syntaktische Fehler werden den Studierenden bereits vom Ontologieeditor mitgeteilt.

Die zur Analyse benötigten Symptome wurden unter Verwendung von Ausdrücken der Sprache OWL-DL formuliert, in OWL 2 sind die benötigten Ausdrücke nur im ausdrucksstärksten Profil OWL 2 $\mathcal{RL}$ enthalten [3].

Ansatz (Name verkürzt)	Logik	findet	Rückmeldung	autom.
Ontology Alignment	OWL-Lite	S, U	Ähnlichkeitsmaß	ja
OWLdiff	OWL 2 $\mathcal{EL}$	S, U	Liste von Konzepten	ja
Begründung der Klassifikation	OWL-DL, OWL 2	S, K, I	Erklärung der Konzepte	ja
Übliche Fehler von Neulingen	OWL-DL	K, U	Verständnisprobleme	nein
neuer Ansatz	OWL-DL / OWL 2 $\mathcal{RL}$	K, U	abweichende Klassen, Verständnisprobleme	ja

autom. ... automatisiert
S ... Syntaktische Fehler
K ... Unerfüllbare Klassen
I ... Inkonsistente Ontologie
U ... Unerwünschte Klassifikation, semantische Fehler

Tabelle 4.7.: Vergleich der Ansätze (hervorgehobene Ansätze wurden verwendet)

5. Umsetzung des Konzepts

Ausgehend vom eben vorgestellten Konzept sind mehrere Varianten der Umsetzung möglich. Da sowohl das in Moodle integrierte ITS eTutor als auch der Ontologieeditor Protégé feststehen, wären folgende Realisierungsvarianten denkbar:

(1) Die Integration der gesamten Client-Funktionalität in die Oberfläche von Protégé. In dieser Alternative müsste die Rolle des Kommunikationskanals *Web-Browser* zusätzlich von Protégé übernommen werden. Daher wären unter anderem die Autorisierung der Studenten an Moodle, das Vorstellen der Aufgabe, die Abgabe der Lösung und das Anzeigen der Rückmeldung zu integrieren. Durch die geänderte Kommunikation würde dies auch Änderungen an den bestehenden Diensten Moodle oder eTutor erfordern.

(2) Am anderen Ende des Lösungsspektrums steht die alleinige Ausarbeitung in Moodle. In dieser Variante passiert die gesamte Kommunikation zwischen Studenten und ITS im Web-Browser, der Ontologieeditor müsste daher entweder im Browser oder in Moodle integriert werden. Auch bei dieser Alternative wären Änderungen an bestehen Diensten nötig.

(3) In der umgesetzten Lösung wird das für den jeweiligen Zweck am Besten geeignete Werkzeug gewählt: Protégé für alle auf Ontologien bezogenen Aufgaben und die Kombination Moodle/Web-Browser für die restliche Kommunikation.

Als Erweiterung zum Konzept kann das erstellte Expertenmodul jede Art von OWL-Entity vergleichen (siehe Seite 69), wobei die Prüfungen entsprechend dem Konzept momentan nur für OWL-Klassen implementiert wurden. Die Unterscheidung der Ontologien erfolgt über deren IRI (vgl. 4.4.1). So wird sichergestellt, dass Abgabe, Basisdaten und Musterlösung voneinander unterschieden werden können.

Abschließend ist zu bemerken, dass die entstandene Software zwar für den Einsatz als Expertenmodul innerhalb des ITS eTutor erstellt wurde, aber auch als eigenständiges Werkzeug verwendet werden kann. Daher kann sie in jede ITS-Umgebung eingebunden oder auch außerhalb einer solchen Umgebung eingesetzt werden. Einige mögliche Anwendungen sind als Folgeprojekte auf Seite 75 skizziert.

5.1. Szenario: Übung mit Einsatz des Expertenmoduls

Ausgehend von den in Einleitung (2.3) und Konzept (4.2) beschriebenen Szenarien ergeben sich die folgenden Abläufe:

5.1.1. Übungsvorbereitung

Für jedes neue Übungsbeispiel müssen die im Anwendungsfalldiagramm 5.1 beschriebenen Tätigkeiten durchgeführt werden: Ein Übungsverantwortlicher entwirft im Ontolo-

gieeditor Protégé die Basisontologie und Musterlösung und legt im eTutor das Übungsbeispiel mitsamt der Angabe an. Dem Expertenmodul wurde der Beurteilungsschlüssel bereits im Rahmen der Implementierung „mitgeteilt“, weshalb nur noch die mögliche Punktezahl angegeben werden muss. Daher wurde dieser Anwendungsfall mit Klammern versehen.

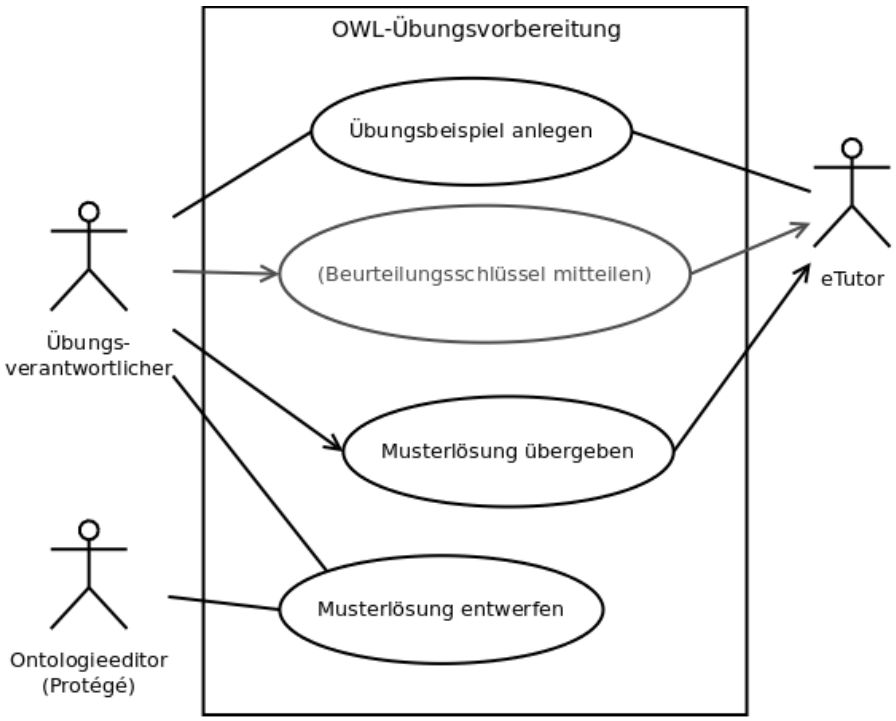

Abbildung 5.1.: Geänderter Anwendungsfall *Vorbereitung der Übung*

Im Gegensatz zum in der Einführung skizzierten Ablauf (Seite 7) übernimmt der eTutor die Aufgaben des menschlichen Tutors. Weiters modelliert der Übungsverantwortliche jetzt immer im Ontologieeditor und lädt die Musterlösung in das ITS. Bis auf diese Details sind seine Abläufe jedoch unverändert.

5.1.2. Ausarbeitung der Übung

Wie im Diagramm 5.2 erkennbar arbeiten die Studenten nach dem Lesen der vom eTutor erhaltenen Angabe das Beispiel in Protégé aus. Bei der Abgabe der Lösung kann jeder Student zwischen Übungs- und Abgabemodus wählen.

Im Übungsmodus werden die durch den Vergleich der abgegebenen mit der Musterlösung gefundenen Erkenntnisse zur Abgabe kommentiert und der Student dadurch detailliert auf die erkannten Fehler hingewiesen. Wurde hingegen der Abgabemodus gewählt, so führt der Vergleich der Lösungen zu einer Beurteilung, die das ITS dem Studenten mitteilt.

Auf Wunsch kann das Beispiel mit einem Betreuer besprochen werden, entweder im Tutorium mit einem Tutor oder in einer Übungseinheit mit der Übungsleitung.

Zwei Hauptunterschied sind im Vergleich mit dem ursprünglichen Ablauf (Seite 8) erkennbar: Erstens geht die Beurteilung vom Betreuer an den eTutor über, zweitens gewinnen die Studenten durch den Übungsmodus die Möglichkeit zur prompten Rückmeldung, anhand derer sie in ihre Lösung verbessern können. Für detaillierte Fragen stehen die durch die Änderungen entlasteten Betreuer zur Verfügung.

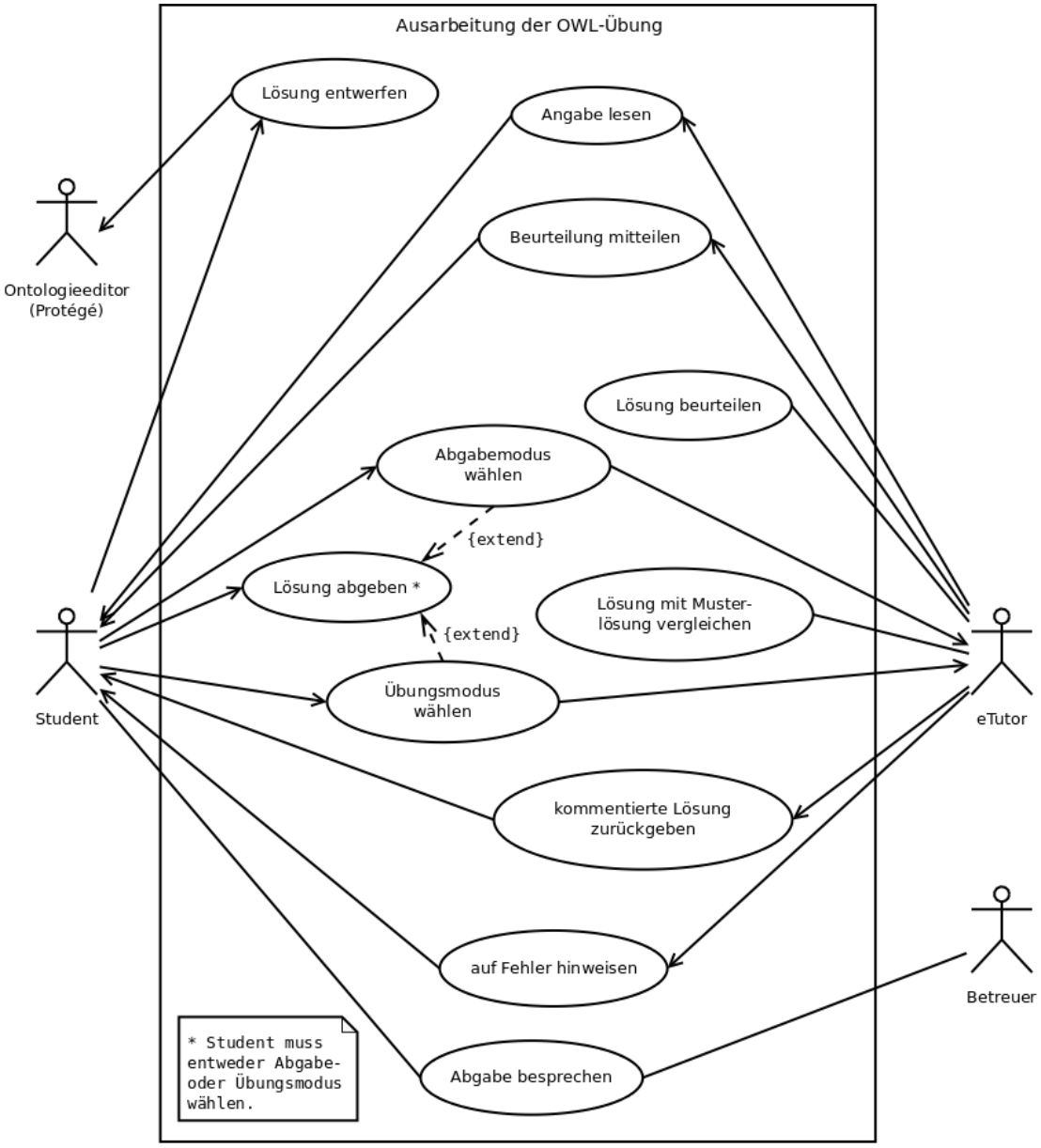

Abbildung 5.2.: Geänderter Anwendungsfall *Ausarbeitung der Übung*

5.2. Problemdiagnose

Das Expertenmodul sammelt Hinweise, die zu Schlussfolgerungen in verschiedenen Themenbereichen verdichtet werden. Eine *Schlussfolgerung* entspricht einem Fehlertyp im Konzept, muss aber auf keine negative Tatsache („Fehler“) bezogen sein. Abbildung 5.3 bietet einen Überblick über den Ablauf der Diagnose. Die im Folgenden beschriebenen Schritte sind idealtypisch und müssen nicht genau mit der Konfiguration übereinstimmen, Details sind in den Kommentaren der Konfigurationsdateien am beiliegenden Datenträger zu finden.

Abbildung 5.3.: Ablauf der Problemdiagnose

In der *Vorbereitung* werden die Basisontologie und beide Ontologien geladen, der Reasoner für die Ontologien erzeugt und die Definitionen der Schlussfolgerungen geladen.

Darauf folgt die *Datenaufbereitung*: Falls die Ontologien Rückmeldungen aus früheren Interaktionen mit dem Modul enthalten werden diese entfernt. Weiters erzeugt dieser Schritt Referenzen auf die in den Ontologien gefundenen OWL-Entities, wobei momentan nur OWL-Klassen berücksichtigt werden (siehe 5.4.2).

Auf diese vorbereitenden Schritte setzt die *Analyse* auf, die alle Symptomprüfungen auf den OWL-Klassen in der in Diagramm 5.4 skizzierten Stufen durchführt. Die so gewonnenen Daten werden in der *Auswertung* interpretiert, indem sie mit den Definitionen der Schlussfolgerungen verknüpft werden. Ergebnisse dieses Schrittes sind Schlussfolgerungen auf den Ebenen Entity und Ontologie, die anhand der Definitionen den Themenbereichen zugeordnet sind.

Abschließend werden diese Schlussfolgerungen im Schritt *Erklärung* den passenden Datenstrukturen der abgegebenen Ontologie als Kommentare hinzugefügt. Diese sollen dem Studenten ein rasches Erfassen der Qualität seiner Abgabe ermöglichen und ihn bei der Fehlerkorrektur unterstützen.

5.3. Rückmeldung

Als Kanäle für die Rückmeldung stehen das Learning Management System Moodle und der Ontologieeditor Protégé zur Verfügung (vgl. 2.1.2 bzw. 5.4.5), beide werden bereits vom Institut in der Lehre eingesetzt. Damit sind Ausgaben in Hypertext[1] und OWL-Ontologien[2] möglich.

[1] HTML und XHTML

[2] in allen von der OWL API unterstützten Serialisierungsformaten (vgl. 5.4.5)

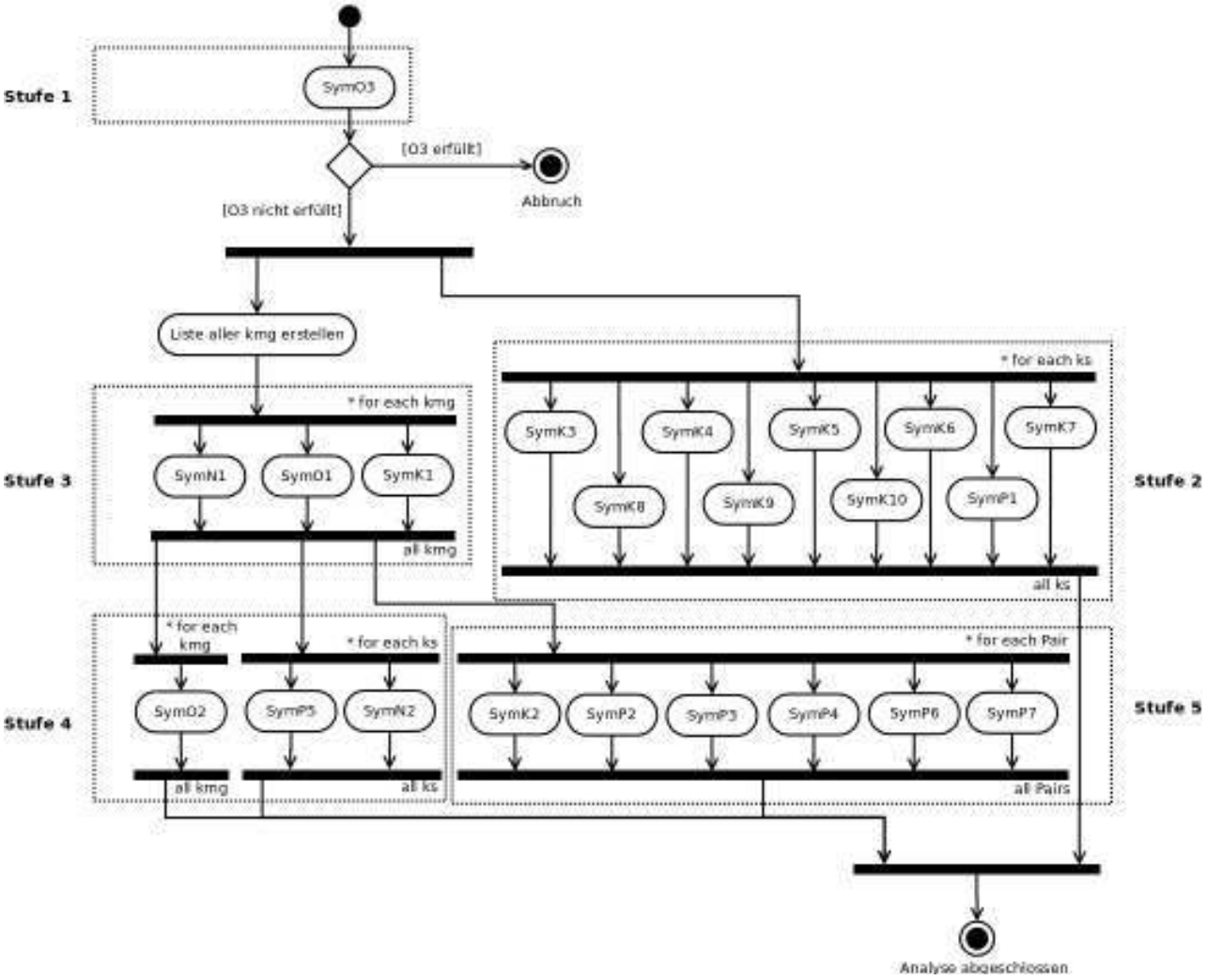

Abbildung 5.4.: Stufen im Ablauf der Analyse

Im Übungsmodus wird die kommentierte Ontologie im Format der Abgabe zurückgegeben. Damit ist sichergestellt, dass der Student die Ontologie öffnen kann, selbst wenn er einen anderen Ontologieeditor verwendet. Die Schlussfolgerungen werden zusätzlich als HTML exportiert und in Moodle angezeigt. Damit bestehen für die Studenten zwei Möglichkeiten, die Antwort des ITS zu betrachten:

(1) Die mittels Web-Browser angezeigte HTML-Antwort liefert eine schnelle Zusammenfassung, ist aber statisch und bietet keine Möglichkeit der Interaktion mit der Rückmeldung. Dafür muss bei dieser Alternative am Client keine weitere Software gestartet werden.

(2) Völlig anders gestaltet sich die Interaktion nach dem Öffnen der kommentierten Abgabe im Ontologieeditor Protégé. Durch die hier vorhandene Möglichkeit, die Ontolo-

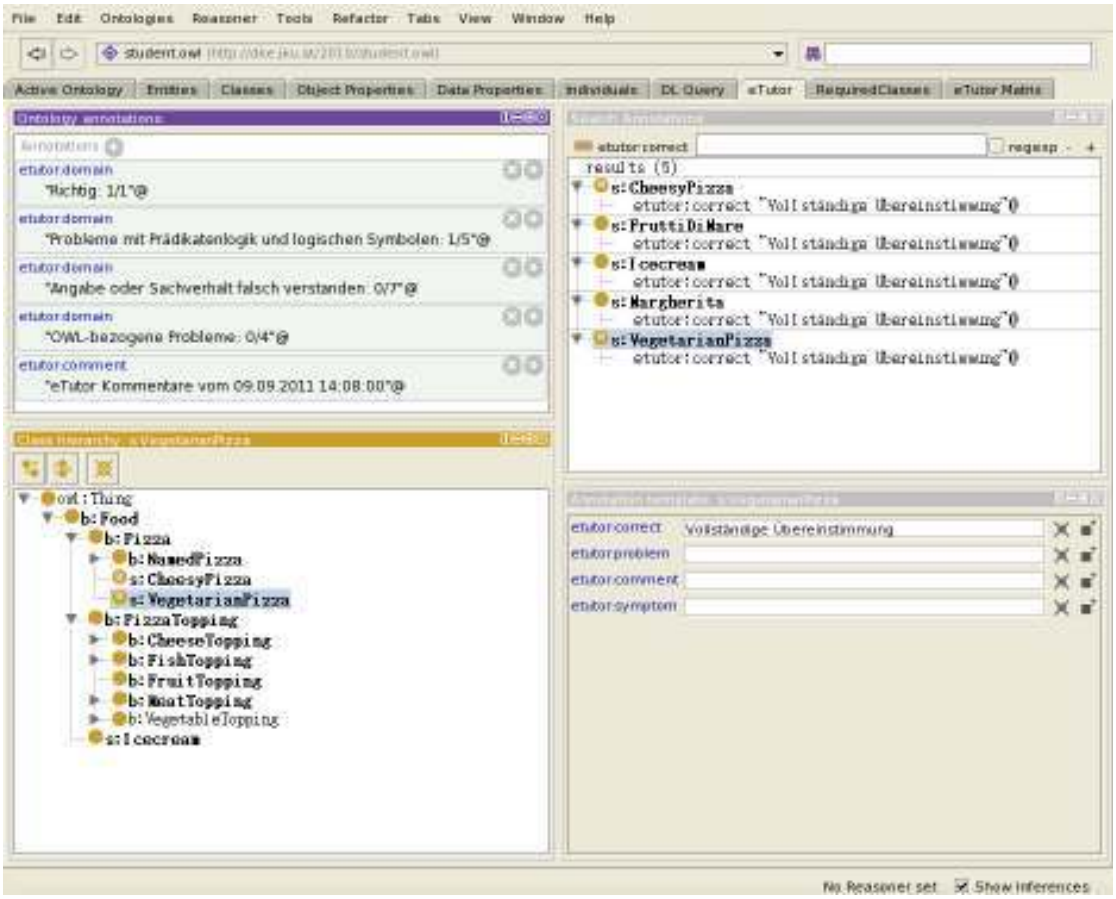

Abbildung 5.5.: eTutor-Reiter in Protégé

gie nach den Kommentaren des eTutors zu filtern, kann gezielt nach vom ITS gefundenen Schlussfolgerungen gesucht werden. Weiters können die dem Studenten vom Modellieren bereits bekannten Entities mit den zusätzlichen Kommentaren betrachtet und verändert werden, um erkannte Fehler zu korrigieren. Um dies zu erleichtern wurde ein eigener eTutor-Reiter erstellt, der unter Verwendung einiger Protégé-Erweiterungen die vom Expertenmodul hinzugefügten Kommentare kompakt darstellt (siehe Abbildung 5.5).

Entsprechend dem Konzept auf Seite 55 wird im Lernfortschrittsmodus keine kommentierte Ontologie zurückgegeben, wobei die Lehrveranstaltungsleitung den Grad der Rückmeldung frei wählen kann. Eine weitere Möglichkeit der Dokumentation ist die Symptom-Schlussfolgerungs-Matrix im ASCII-Format, die auch für die Testfälle im Anhang verwendet wurde.

5.4. Implementierung

Die erstellte Software ist *ein Framework zum Vergleich einer OWL-Ontologie mit einer Musterlösung (ebenfalls in Form einer OWL-Ontologie), das Schlussfolgerungen aufgrund von Prüfungen der OWL-Entities zieht.* Gemäß den Anforderungen des eTutor [17, 15] wurde sie in der Programmiersprache Java [16] implementiert.

Das Konzept wurde bis auf folgende Änderungen wie beschrieben umgesetzt: Der Fehlertyp 8 („Klasse durch leere Menge trivial erfüllbar“) wurde nicht realisiert, weil das dafür benötigte Symptom (K10) von der Lehrveranstaltungsleitung die Definition je einer *trivialen OWL-Klasse* für jede als hinreichend definierte Klasse der Abgabe erfordern würde, was nicht zweckmäßig erscheint.

Weiters konnte die für den Fehlertyp 9 („Verneinung falsch positioniert“) nötige Erwartung des Studenten bezüglich der Position einer Klasse nicht als Symptom formuliert werden. Es basiert daher rein auf der Analyse von Klassenaxiomen und wird deshalb nicht als *Fehler*, sondern als *Hinweis* analog Hinweistyp 2 umgesetzt.

Zur Verbesserung der Genauigkeit der Analyse wurde direkt vor der Auswertung ein Filter eingeführt, der überflüssige Paare entfernt (vgl. `RemoveExcessPairs` auf Seite 66). Dadurch war eine Reihung der Schlussfolgerungen nach Priorität nicht mehr notwendig.

5.4.1. Prinzipieller Aufbau

Das Expertenmodul ist sehr modular und hochgradig konfigurierbar aufgebaut, daher hängen Java-Programm und Konfiguration sehr eng zusammen und werden gemeinsam besprochen. Einstellungen erfolgen mittels `java.util.Properties`, die thematisch in mehrere Dateien aufgeteilt sind und bei Bedarf einfach geändert oder ergänzt werden können. Alle Konfigurationsdateien werden bei jedem Aufruf durch den Studenten neu eingelesen, daher ist ein Unterbrechen des Betriebs des Expertenmoduls nur bei Änderungen am Framework erforderlich, alle anderen Änderungen werden beim nächsten Aufruf aktiv.

Die Datei `general.properties` enthält allgemeine Einstellungen wie Zeichen für die Verwendung in anderen Konfigurationsdateien, den zu verwendenden OWL-Reasoner und die Orte der Ontologien mit den IRIs, über die sie definiert wurden. Eine wesentliche Einstellung ist die die Reihenfolge der Schritte, die abgearbeitet werden sollen. Abbildung 5.6 zeigt den Zusammenhang der Konfigurationsschritte mit denen der Problemdiagnose.

Diese Schritte werden in der Datei `actions.properties` den gewünschten Aktionen zugeordnet, wobei die Aufteilung der Prüfungen in die Schritte *Checks1* bis *Checks5* sicherstellt, dass alle Aktionen eines Schrittes nur von vorigen Schritten abhängen, voneinander aber unabhängig und daher parallelisierbar sind (siehe Ablaufdiagramm auf Seite 61). Aktionen können auch mehrmals eingetragen werden falls ein mehrmaliger Aufruf gewünscht wird, etwa bei der Fehlersuche.

Die Datei `conclusions.properties` definiert Schlussfolgerungen anhand gefundener Symptome (in Form der Klassennamen ihrer Prüfungen). Der Kern jeder Schlussfolgerung sind die aus der Symptom-Fehlertyp-Matrix bekannten sicheren und möglichen Symptome. Zusätzlich werden der Themenbereich, die Signifikanz und Art der Schluss-

folgerung festgelegt, wobei die Arten *korrekte Lösung*, *Problem* und *fatales Problem* zur Auswahl stehen. Die momentan implementierten Aktionen überspringen bei Existenz fataler Probleme den Export anderer Schlussfolgerungen.

Die Einträge der Datei `MessagesBundle.properties` legen die Texte für die deutschsprachige Ausgabe fest.

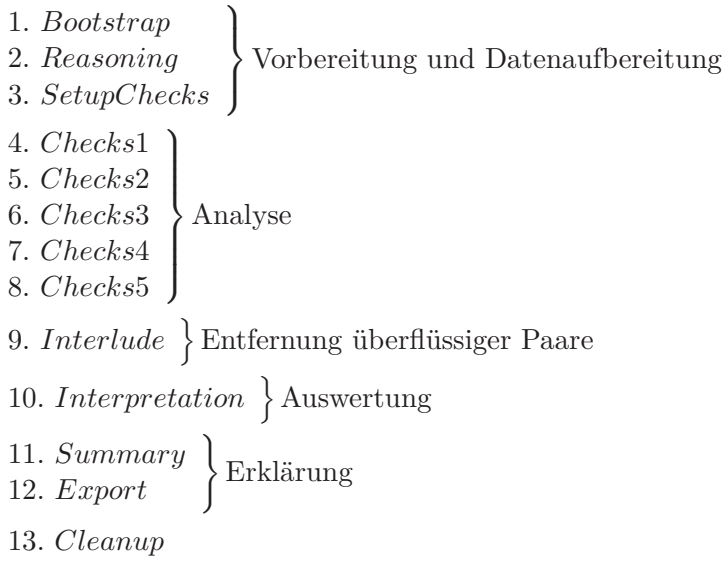

Abbildung 5.6.: Konfiguration der Schritte der Problemdiagnose

5.4.2. Paketübersicht

Die Paketnamen ergeben sich aus der Internetdomäne der *Johannes Kepler Universität Linz* (`jku.at`), dem *Institut für Wirtschaftsinformatik - Data & Knowledge Engineering* (`dke`), dem dort im Einsatz befindlichen ITS *eTutor* (`etutor`) und dem im Rahmen der vorliegenden Arbeit entwickelten *Expertenmodul für Description Logics* (`dl`) und entsprechen daher der in Java üblichen Namenskonvention [16, 170].

Der Zusammenhang zwischen dem Expertenmodul und der OWL API (siehe Seite 74) ist sehr eng. Die Abhängigkeiten der implementierten Pakete von jenen der OWL API wird durch das Diagramm 5.7 verdeutlicht, wobei erst die Verwendung der Klassen und Schnittstellen durch Methodenaufrufe oder Vererbung als Interaktion gewertet werden, reine Weitergabe von Daten jedoch nicht.

Die folgenden Seiten und das Klassendiagramm auf Seite 67 bieten nur einen groben Überblick über die implementierten Pakete, nähere Information ist in der im Javadoc-Format vorliegenden API-Dokumentation zu finden.

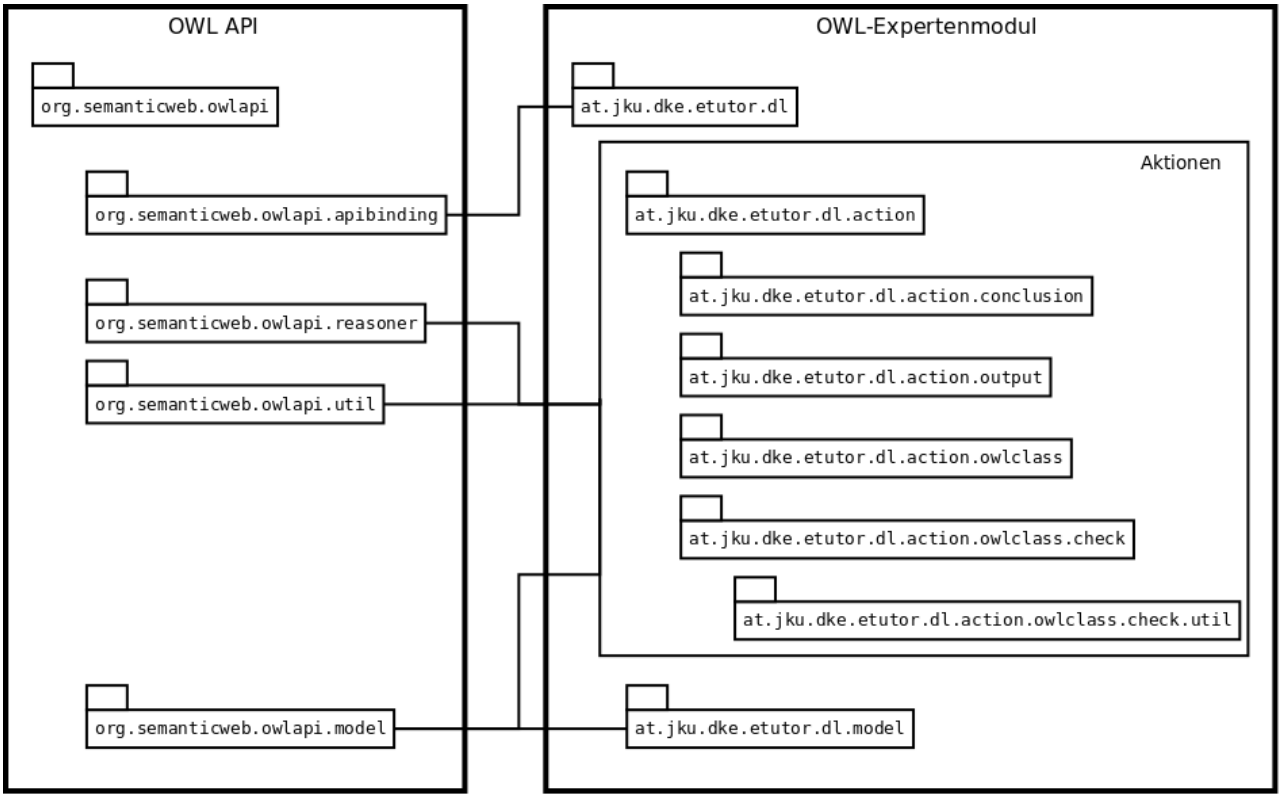

Abbildung 5.7.: Interaktion zwischen OWL API und Expertenmodul auf Paketebene

at.jku.dke.etutor.dl

Dieses Paket ist der Einstiegspunkt in das Expertenmodul. Es enthält zwei Möglichkeiten zum Starten der Problemdiagnose: Der einfache `Starter` führt die zu einem Schritt zusammengefassten Aktionen nacheinander in der angegebenen Reihenfolge aus. Im Gegensatz dazu startet der `QuickStarter` alle Aktionen eines Schritts als parallel laufende Threads und kann daher die Analyse schneller durchführen. Aufgrund mangelnder Thread-Sicherheit von OWL API und Pellet[3] wird jedoch der serielle `Starter` für den Produktiveinsatz empfohlen.

Der Mehrfachstarter `BucketOTests` analysiert alle Studentenontologien in einem Verzeichnis und wurde bei der Entwicklung zur Stapelverarbeitung der im Anhang angeführten Testfällen verwendet. Die ebenfalls hier zu findende `Util`-Klasse stellt die Verbindung zwischen Konfiguration und Aktionen her.

[3] laut Diskussion in der Pellet-users mailing list unter
`http://lists.owldl.com/pipermail/pellet-users/2009-May/003578.html`

at.jku.dke.etutor.dl.action

Die Schnittstelle Aktion (`Action`) bildet die kleinste Verarbeitungseinheit im Expertenmodul. Durch Konfiguration werden Aktionen so zu Schritten zusammengefasst, dass keine Abhängigkeiten innerhalb eines Schrittes bestehen.

Weiters enthält dieses Paket allgemeine Aktionen zum Laden der Ontologien (`LoadOntologies`), des Reasoners (`AddReasoner`) und zum Entfernen von Kommentaren früherer Durchläufe (`PurgeAnnotations`). Bezüglich unterstützter Ontologieformate und Reasoner ist nur auf die Kompatibilität mit der verwendeten Version der OWL API (siehe Seite 74) zu achten.

Außerdem enthalten ist eine Aktion, die überflüssige Paare aus der Analyse entfernt (`RemoveExcessPairs`). Ein ungleiches Paar ist dann überflüssig, bereits wenn ein anderes Paar komplette Übereinstimmung aufweist. Wie im kommentierten Beispiel auf Seite 48 zu erkennen, wird die Verarbeitung durch solche Paare unnötig verkompliziert.

Die Aktionen dieses Paketes können alle Arten von Entities (Abbildung 5.8) verarbeiten, beispielsweise OWL-Klassen oder Properties.

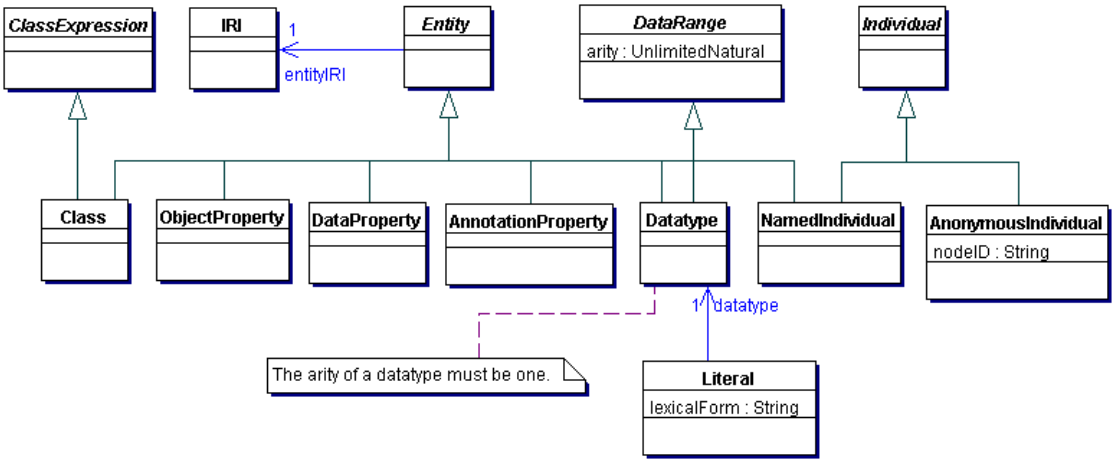

Abbildung 5.8.: Entities in OWL 2 (nach [5])

at.jku.dke.etutor.dl.action.conclusion

In diesem Paket befinden sich alle Aktionen, die mit Schlussfolgerungen (`Conclusions`) zu tun haben, etwa indem sie diese aus deren Konfiguration erzeugen (`AddConclusions`) oder mittels bei Entities gefundenen Symptomen auf Gültigkeit prüfen (`AddEvidenceToConclusions`). Auch in diesem Paket können alle Aktionen jede Art von Entities verarbeiten.

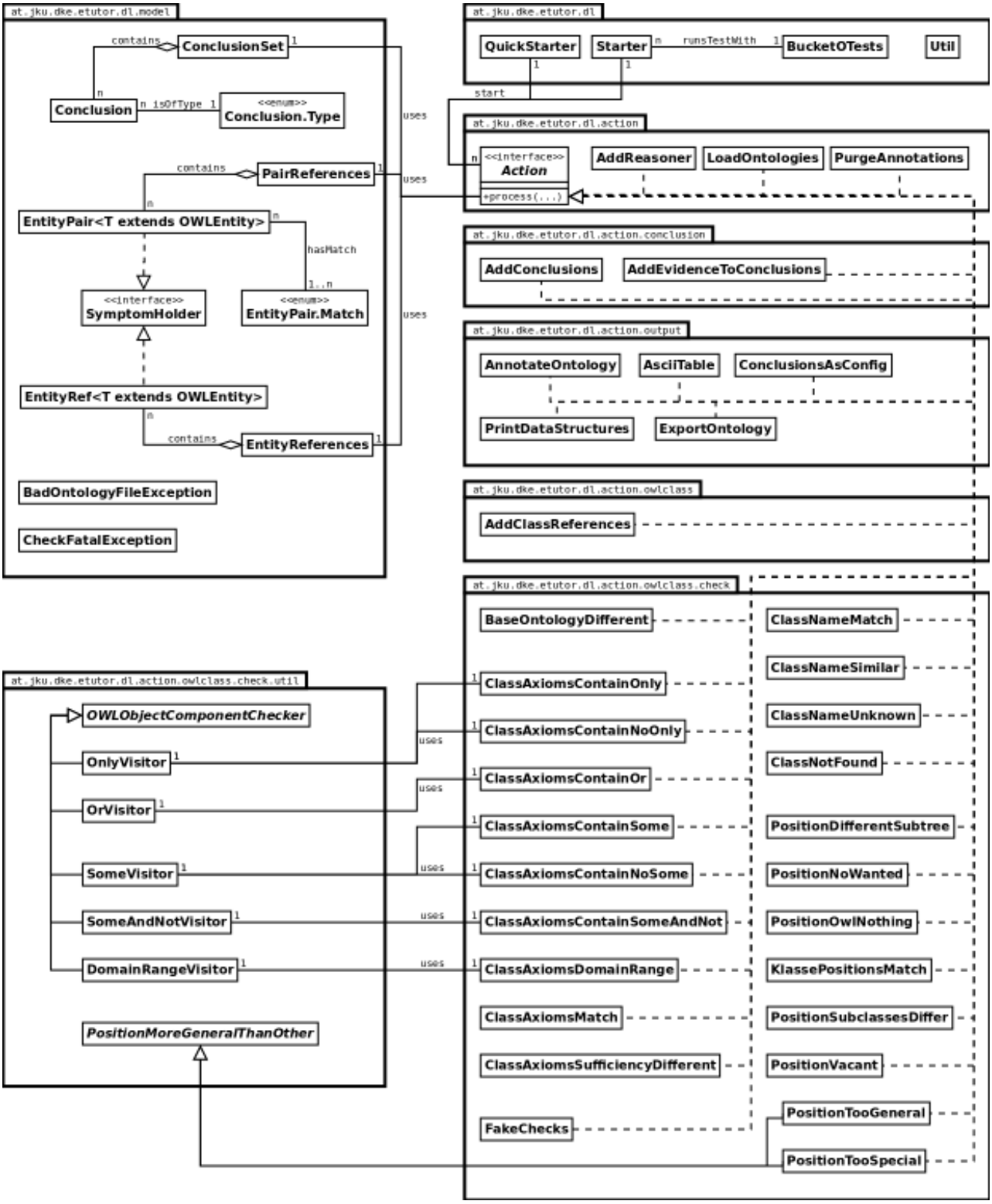

Abbildung 5.9.: Klassendiagramm des Frameworks

at.jku.dke.etutor.dl.action.output

Hier finden sich neben Ausgaben zur Fehlersuche (`AsciiTable` und `PrintDataStructures`) alle für den Erklärungsschritt (siehe 5.2 auf Seite 60) verantwortlichen Aktionen: `AnnotateOntology` fügt allen behandelten Entities der Studentenlösung Anmerkungen mit Ergebnissen von Analyse und Auswertung hinzu, die erzeugten Texte wurden wie in Java üblich internationalisiert[4]. Die Aktion `ExportOntology` gibt die so kommentierte Studentenlösung im gleichen Format aus, in dem sie abgegeben wurde. Wie die vorigen Pakete behandeln die hier enthaltenen Aktionen alle Arten von Entities.

at.jku.dke.etutor.dl.action.owlclass

Alle Pakete mit `owlclass` im Namen behandeln nur OWL-Klassen (Konzepte), andere Arten von Entities werden ignoriert. Die einzige Aktion in diesem Paket verarbeitet OWL-Klassen, führt aber kein Symptomprüfungen durch: `AddClassReferences` erzeugt Verweise auf geforderte Klassen der Musterlösung und alle Klassen der Studentenlösung, damit später Symptome darauf bezogen werden können.

at.jku.dke.etutor.dl.action.owlclass.check

Dieses Paket enthält alle Symptomprüfungen für OWL-Klassen. Die Namen der Java-Klassen folgen der Einteilung von Symptomen in Abschnitt 4.5.1: Prüfungen auf den *Namen* der OWL-Klasse (wie `ClassNameSimilar`) beginnen mit `ClassName`, solche auf *Klassenaxiome* (beispielsweise `ClassAxiomsContainOr`) mit `ClassAxioms` und jene auf die *Position* der Klasse (etwa `PositionTooGeneral`) mit `Position`. Die Zuordnung zwischen den im Konzept definierten Symptomen und den Aktionen, die auf diese Symptome prüfen, ist aus den Kurzbezeichnungen in der Konfiguration dieser Aktionen ersichtlich.

Die noch nicht besprochene Klasse `FakeChecks` erzeugt immer Symptome und wurde zur Fehlersuche bei der Definition neuer Symptome entwickelt. Weitere Information zu ihrer Verwendung ist in der API-Dokumentation der Klasse enthalten.

at.jku.dke.etutor.dl.action.owlclass.check.util

In diesem Paket befinden sich Hilfsklassen für die Symptomprüfungen der OWL-Klassen, und zwar sowohl allgemeine Versionen von Prüfungen, die von mehreren Aktionen direkt oder nur leicht verändert verwendet werden (`PositionMoreGeneralThanOther`) als auch Java-Klassen zum Durchlaufen der Axiome einer OWL-Klasse mittels *Visitor* Entwurfsmuster [15, 628f]: `OWLObjectComponentChecker` implementiert das Traversieren der Klassenaxiome entweder in der Form in der sie definiert wurden oder in NNF (siehe Seite 23). Auf `Visitor` endende Klassen überschreiben Methoden für von ihnen gesuchte OWL-Operatoren.

[4] `http://download.oracle.com/javase/tutorial/i18n/index.html`

at.jku.dke.etutor.dl.model

Da jede Aktion im Wesentlichen mit den vom Starter als Parameter übergebenen Daten auskommen muss, gleichzeitig aber verschiedene Arten von OWL-Entities verarbeiten kann, umfasst die Implementierung neben einfachen Klassen auch Datenstrukturen zu deren Verwaltung.

Die einfachen Datentypen wurden unter Einsatz von *Generics* [16, 178ff] erstellt und können daher für alle Arten von `OWLEntity`, also alle in der OWL2 Spezifikation angeführten Typen von Entities (*classes*, *datatypes*, *object properties*, *data properties*, *annotation properties* und *named individuals*) [5] verwendet werden (siehe Abbildung 5.8). Im Detail:

Jede Instanz von `EntityRef` ist ein Verweis auf eine bestimmte Entity einer Ontologie und enthält die zu dieser Entity gefundenen Symptome. Analog dazu werden Paare aus je einer Entity der Musterlösung und einer der Abgabe als `EntityPair` verwaltet, das neben Symptomen auch Information über die Art der Paarbildung (`EntityPair.Match`) enthält. `EntityPair` und `EntityRef` sind beide `SymptomHolder` und können daher von der Auswertung gleich abgefragt werden.

Bei der Auswertung fügt die bereits besprochene Aktion `AddEvidenceToConclusions` jeder Schlussfolgerung (`Conclusion`) die passenden `SymptomHolder` hinzu. Es gibt verschiedene Arten (`Conclusion.Type`) von Schlussfolgerungen, die zu unterschiedlichen Anmerkungen in der abgegebenen Ontologien führen.

Alle Referenzen auf einzelne Entities werden gesammelt als `EntityReferences` vom Starter an jede Aktion übergeben, alle Referenzen auf Paare als `PairReferences`. Ebenfalls zur Verfügung gestellt wird die Menge aller Schlussfolgerungen (`ConclusionSet`).

Zwei Ausnahmebedingungen wurden definiert: Eine `BadOntologyFileException` weist auf Probleme mit der abgegebenen Datei hin und wird an den Studenten weitergegeben, die `CheckFatalException` hingegen wird nur intern beim Annotieren der Ontologie verwendet, um beim Auftreten bestimmter Schlussfolgerungen nur diese zu kommentieren.

5.4.3. Verwendung des Frameworks

Grundsätzliche Überlegungen

Trotz der hohen Flexibilität sind einige Prinzipien des erstellten Framework vorgegeben: Als Daten stehen je eine Muster-, Studenten- und Basisontologie mit der jeweiligen IRI zur Verfügung. Die Starter sowie die Pakete `at.jku.dke.etutor.dl.model` und `at.jku.dke.etutor.dl.action` bilden die Basis des Frameworks und sind daher nicht für Änderungen vorgesehen.

Die Schnittstelle für Aktionen[5] wurde unter Berücksichtigung möglicher Erweiterungen entworfen und verwendet entsprechende Datenstrukturen. Ein Teil davon sind Paare von OWL-Entities, deren Art der Übereinstimmung momentan auf *Name*, *Position* und *Klassenaxiome* beschränkt ist. Bei zukünftigen Bedürfnissen sollte die Enumeration `at.jku.dke.etutor.dl.model.EntityPair.Match<T>` um neue Gemeinsamkeiten

[5] Interface `Action`, nähere Informationen als Javadoc auf beiliegendem Datenträger

erweitert werden.

Ein anderer Fixpunkt ist die Einteilung des Ablaufs in einen konfigurierbaren *Durchlauf* des Starters mit aufeinanderfolgenden *Schritten*, die aus parallelisierbaren *Aktionen* in Form einzelner Java-Klassen bestehen. Daher wird die Beibehaltung folgender Reihenfolge der Schritte empfohlen: Ontologien einlesen, Reasoning, diverse Aktionen ausführen, Ontologie ausgeben.

Wichtige Aspekte im Umgang mit dem Framework

Das Einbinden des Frameworks erfolgt am Einfachsten durch Instanzieren des `Starter`s im Kontext der gewünschten Anwendung analog der Verwendung in `BucketOTests`.

Ein Austausch des OWL-Reasoners beschränkt sich theoretisch auf das Einspielen der JAR-Dateien eines mit der OWL API kompatiblen Reasoners und dem Eintrag der *Factory*-Klasse in die allgemeine Konfiguration. Eine Änderung der Aktionen ist nicht nötig. Bei einem testweisen Austausch des Reasoners Pellet durch Hermit [28, 167] war allerdings eine Änderung der entsprechenden Aktion (`AddReasoner`) erforderlich.

Hinzufügen, Ändern und Entfernen einer Schlussfolgerung

Die Existenz einer Schlussfolgerung ergibt sich rein aus dem entsprechenden Konfigurationseintrag (`conclusions.properties`). Daher ist eine Änderung der Schlussfolgerungen sehr einfach und wird automatisch beim nächsten Aufruf berücksichtigt.

Deaktivieren einer fehlerhaften Prüfung

Um eine unerwünschte Prüfung zu deaktivieren reicht es, deren Eintrag in der Konfiguration der Aktionen auszukommentieren. Beim nächsten Aufruf des Starters wird die Konfigurationsdatei neu gelesen und die Zeile ignoriert. Sind Schlussfolgerungen über die deaktivierte Aktion definiert, so bewirkt die fehlende Prüfung, dass aus Sicht der Schlussfolgerungen das Symptom bei keiner Entity auftritt. Falls dieses Verhalten nicht gewünscht ist sollten die betroffen Schlussfolgerungen ebenfalls deaktiviert werden.

Ersetzen einer fehlerhaften Aktion

Um eine fehlerhafte Aktion durch eine neue Version zu ersetzen reicht es aus, die entsprechende Java-Klasse zu ersetzen.

Hinzufügen einer neuen Aktion mit einem eigenem Schritt

Um eine neue Aktion in den Ablauf einzubinden muss sie nur im Classpath vorhanden und in der Konfiguration eingetragen sein (`actions.properties`), wobei der vollständige Name der Klasse, der Schritt in dem die Aktion ablaufen soll und optional eine Kurzbezeichnung anzugeben sind.

Falls die Aktion einem neuen Schritt (*stage*) zugeordnet werden soll muss dieser in der allgemeinen Konfiguration (`general.properties`) an der gewünschten Stelle erwähnt

werden. Durch Eintragen entsprechender Schritten in die Konfiguration ist das Aktivieren und Deaktivieren einer Fehlersuche im laufenden Betrieb problemlos möglich. Schritte können auch mehrfach eingetragen werden.

Anpassung der Parallelisierung

Bei Verwendung der parallelisierten Implementierung (`QuickStarter`) laufen alle zu einem Schritt gehörende Aktionen gleichzeitig als Threads. Hauptgrund der Teilung in Schritte ist zwar die Ablaufsteuerung, diese Aufteilung kann aber zusätzlich zur Optimierung der Systemlastgenutzt werden: Durch die weitere Teilung einzelner Schritte kann die Anzahl der parallelen Schritte reduziert und dadurch Prozessorzeit anderen Anwendungen überlassen werden werden. Das Minimum der Schritte ist durch die nötige Trennung aufeinander folgender Aktionen beschränkt. Da die Einteilung der Schritte rein in Konfigurationsdateien passiert werden die Änderungen beim Verarbeiten der nächsten Studentenontologie wirksam und können so unter Last optimiert werden.

5.4.4. Integration in eTutor

Die Integration in eTutor als OWL-Expertenmoduls erfolgte anhand des aktuellen Administrationshandbuchs für eTutor [14, 112ff]. Dieser Anleitung entsprechend wurde das bestehende SQL-Modul als Vorlage verwendet, um die Verarbeitung von Basisontologien ergänzt und an das erstellte OWL-Framework angepasst. Im Folgenden wird nur auf die wesentlichsten Aspekte des neuen Expertenmoduls eingegangen, die technische Feinheiten finden sich in den Handbüchern auf dem beiliegenden Datenträger.

Verbindung zu eTutor

Das Paket `etutor.modules.owl` stellt den Großteil der Verbindung zum eTutor her. Dazu implementiert die Klasse `OWLEvaluator` die im eTutor vorgesehenen Schnittstellen zur automatischen Korrektur der Übungsaufgabe, der Punktevergabe und der Erstellung der Rückmeldung an die Studierenden [14, 113]. Als Datenstruktur für den Transport von Basisontologie, Musterlösung und Abgabe wird das eigens dazu erstellte `OWLExerciseBean` verwendet. Die Klasse `OWLExerciseManager` implementiert das Laden und Speichern von Musterlösungen und Basisontologien in Verbindung mit einer Oracle SQL-Datenbank.

Analyse der Abgabe

Aufgabe des Pakets `etutor.modules.owl.analysis` ist die Analyse der abgegebenen Lösung. Dabei ruft die Klasse `OWLAnalyzer` das im Rahmen dieser Arbeit erstellte Framework zum Vergleich zweier OWL-Ontologien auf und schreibt die Analyseergebnisse in ein `OWLAnalysis`-Objekt. Eventuelle bei der Analyse auftretende Fehler werden als `AnalysisException` gespeichert. Zu diesem Zeitpunkt sind die Analyseergebnisse bereits internationalisiert, werden also in der vom Benutzer in Moodle gewählten Sprache verfasst.

Punktevergabe

Auf Wunsch des Studenten werden, ausgehend von den Analyseergebnissen, die bei der Aufgabe erreichten Punkte berechnet. Die im Paket `etutor.modules.owl.grading` implementierte Beurteilung unterscheidet dabei nur zwischen richtiger und falscher Lösung. Eine Abgabe wird dann als richtig gewertet und erhält die volle Punktezahl, wenn alle in der Musterlösung geforderten Klassen gefunden wurden. Wird auch nur eine geforderte Klasse nicht gefunden, so bewertet die Klasse `OWLGrader` diese Aufgabe mit null Punkten. Andere während der Analyse gezogene Schlussfolgerungen haben keine Auswirkung auf die Punktevergabe.

Erstellung der Rückmeldung

Im Paket `etutor.modules.owl.report` werden die Ergebnisse der Analyse für die Ausgabe vorbereitet. Dazu erzeugt die Klasse `OWLReporter` einen `Report`, dessen Detailliertheit den vom Übungsverantwortlichen definierten Grad der Rückmeldung berücksichtigt. Bei einem Grad größer Null werden sowohl kommentierte Abgabe als auch Kurzzusammenfassung ausgegeben, sonst keine der beiden.

Benutzeroberfläche

Das ITS eTutor interagiert mit allen Benutzern über das webbasierte Learning Management System Moodle. Daher erfolgt die gesamte Ein- und Ausgabe des Expertenmoduls mittels in Moodle integrierter HTML-Formulare. Das Erzeugen der Formulare und Verarbeiten der übertragenen Daten ist die Aufgabe des Pakets `etutor.modules.owl.ui` in Kombination mit der *Velocity Template Engine*[6]. Die Ausgabe erfolgt durch die Java-Klassen `OWLExerciseSettingView`, `OWLShowEditorView` und `OWLPrintReportView`, die dem jeweiligen Velocity Template alle benötigten Daten zur Verfügung stellen. Diese Aufteilung ermöglicht eine klare Trennung von Programmlogik und Webdesign.

Alle Benutzereingaben werden von der Klasse `OWLEditor` verarbeitet, sowohl bei der Erstellung und Bearbeitung von Übungsbeispielen als auch bei der Abgabe der Übungen. Um die Mehrsprachigkeit des Expertenmoduls zu gewährleisten wurden alle Velocity Templates internationalisiert, gemeinsam mit den Ausgaben des OWL-Frameworks ins Englische übersetzt und in die Ausgabetexte des eTutors integriert.

Die Abbildung 5.10 zeigt einen Ausschnitt der Benutzeroberfläche, nachdem eine Abgabe zur Bewertung hochgeladen wurde. Das Expertenmodul hat die Lösung als nicht richtig erkannt und gibt detaillierte Rückmeldung zu den gefundenen Fehlern. Die Basisontologie, gespeicherte und kommentierte Abgabe können über Hyperlinks heruntergeladen werden. Der Student hat nun die Möglichkeit, seine Lösung zu verbessern und erneut abzugeben.

[6] `http://velocity.apache.org`

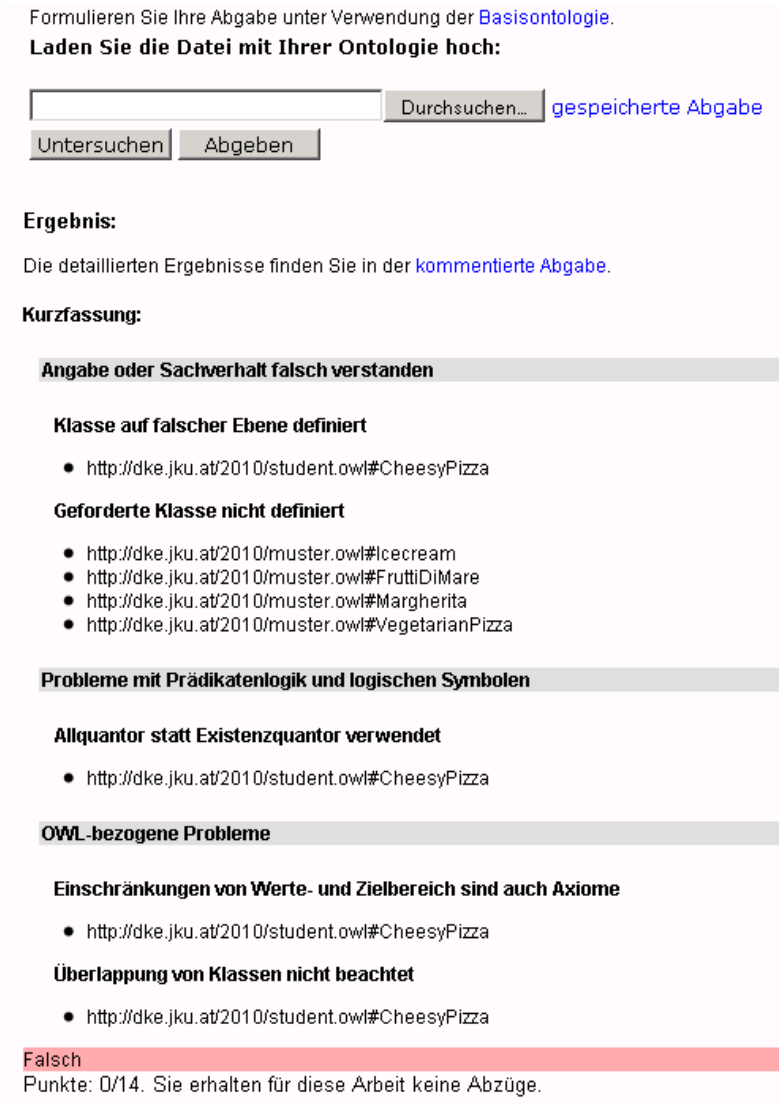

Abbildung 5.10.: Ausschnitt der Benutzeroberfläche des Expertenmoduls

5.4.5. Für die Funktion wesentliche Programme und -bibliotheken

OWL API

Die OWL API [18] ist eine freie Referenzimplementierung des OWL-Standards in Java. Der für diese Arbeit wesentlicheste Teil ist die vollständige Implementierung aller Strukturelemente von OWL2 [5]. Neben der Verwendung der Schnittstellen für diverse Reasoner ist die Unterstützung zahlreicher Serialisierungsformate [18, 4] von Bedeutung, durch die jeder Student das von ihm bevorzugtes Format verwenden kann ohne dass das Expertenmodul betroffen ist. Die verwendete Version 3 wird an der Universität von Manchester entwickelt.

Pellet

Die OWL API definiert nur Schnittstellen für OWL-Reasoner, enthält aber keine derartigen Implementierungen [18, 9f]. Die Wahl des Reasoners fiel auf Pellet [35], da er OWL-DL und OWL 2 unterstützt, in Java implementiert und unter einer freien Lizenz verfügbar ist. Pellet bietet interessante Zusatzfunktionen (*non-standard reasoning services*) wie das Begründen der Klassifikation einer Ontologie (vgl. Seite 12). Da eine Verwendung derartiger Funktionen die Austauschbarkeit des Reasoners im Expertenmodul erschweren würde, werden jedoch nur in jedem OWL-Reasoner vorhandene Funktionen (*standard reasoning services*) [32, 163f] verwendet. Diese können direkt über die OWL API angesprochen werden.

Protégé-OWL

Protégé-OWL ist eine Erweiterung des Ontologie-Editors Protégé für OWL [22]. In der aktuellen Version 4 verwendet es unter anderem die OWL API [18, 7f], ist unter einer freien Lizenz erhältlich und kann mittels Erweiterungen sehr einfach angepasst werden. So kann etwa durch Installation eines OWL-Reasoners die definierte Ontologie mittels Klassifikation überprüft und verwendet werden. Die Oberfläche von Protégéwurde so gestaltet, das viele Fehler möglichst vermieden oder frühzeitig erkannt werden [31, 78].

6. Mögliche Folgeprojekte

6.1. Erweiterung der Prüfung auf andere Entity-Typen

Exemplarisch für jede weitere Art von OWL-Entities (vgl. Abbildung 5.8) wird das Vorgehen zur Erweiterung um Individuen beschrieben. Es besteht aus mehreren Teilen: Zuerst werden Aktionen zur Behandlung von Individuen (`OWLNamedIndividual`) programmiert und als neue Pakete `at.jku.dke.etutor.dl.action.owlnamedindividual` und `at.jku.dke.etutor.dl.action.owlnamedindividual.check` hinzugefügt. Dabei können die Aktionen für OWL-Klassen in `at.jku.dke.etutor.dl.action.owlclass` und `at.jku.dke.etutor.dl.action.owlclass.check` als Vorlage dienen.

Im zweiten Teil werden die neuen Aktionen Schritten zugeteilt, wobei die bestehenden Schritte `Checks1` bis `Checks5` verwendet werden. Zu Testzwecken oder bei Bedarf können neue Schritte definiert werden.

Im dritten und letzten Teil werden die Auswirkungen der geprüften Symptome bestimmt, also deren Verwendung in Schlussfolgerungen. Dabei können entweder bestehende Folgerungen erweitert werden oder neue Schlussfolgerungen nur für Individuen definiert werden.

6.2. Stärkere Integration von Protégé in eTutor

Die momentane Präsentation der vom Expertenmodul erstellten Kommentare in Protégé ist sehr spartanisch. Durch die Verwendung von Erweiterungen wie *EditorPanePlain*[1] könnten die aktuell sehr kurzen, unformatierten Texte durch die Verwendung von HTML ansprechend formatiert, mit Hyperlinks versehen und in Protégé angezeigt werden. Außerdem könnte eine Übersicht eingebunden werden, die Links auf fehlerhafte Entities enthält. Dadurch würden die Vorteile der momentan in Web-Browser und Ontologieeditor getrennten Teile vereint werden.

Ebenfalls möglich wäre die Entwicklung eines eigenen Protégé-PlugIns zur Kommunikation mit eTutor, um eine Ontologie direkt vom Editor aus abgeben zu können.

Eine weitere Alternative zur stärkeren Integration von eTutor und Protégé ist die Verlagerung des Ontologieeditors vom Computer des Studenten auf einen Webserver. Dadurch könnte den Studenten ein sofort einsatzfähiger Editor mit allen benötigten PlugIns zur Verfügung gestellt und die Möglichkeit clientseitiger Softwareprobleme reduziert werden. Konkret wäre der Einsatz des webbasierten Ontologieeditors *WebProtégé*[2] möglich,

[1] `http://protegewiki.stanford.edu/wiki/EditorPanePlain`
[2] `http://protegewiki.stanford.edu/wiki/WebProtege`

der momentan an der Universität Stanford entwickelt wird. Zusätzlich könnte eine eventuelle Einbindung in die Lernumgebung Moodle das lokale Abspeichern und Hochladen ersetzen.

6.3. Differenzierte Beurteilung für das Expertenmodul

Durch den Fokus der vorliegenden Arbeit auf den Übungsmodus enthält der Lernfortschrittsmodus nur eine rudimentäre Beurteilung, die durch ein Folgeprojekt erweitert werden könnte. Dazu wären verschiedene Möglichkeiten der Punktevergabe vorstellbar, etwa ein *Abzugsmodell*, das mit voller Punktezahl beginnt und bei erkannten Fehlern je nach Schwere entsprechend Punkte subtrahiert, oder ein *additives Modell*, das von null Punkten ausgeht und richtige Teilergebnisse belohnt.

Die für die Beurteilung relevanten Symptomprüfungen müssen nicht unbedingt mit denen zur Erklärung von Fehlern übereinstimmen, können aber leicht als eigene Aktionen in das erstellte Framework eingebunden werden. Die Gewichtung einzelner Teile der Abgabe könnte durch Annotationen bei Entities der Musterlösung ähnlich der geforderten OWL-Klassen realisiert werden. Ein weiterer wichtiger Punkt wäre eine Methode zum Erkennen von Folgefehlern und deren Berücksichtigung in der Bewertung.

6.4. Protégé-Plugin zur Unterstützung menschlicher Tutoren

Ausgehend vom in der Einführung auf Seite 7 beschriebenen Szenario ist auch eine sehr einfache dezentrale Lösung ohne Einsatz eines ITS denkbar. Darin erhalten die Tutoren alle Ontologien als OWL-Dateien, und zwar Basisontologie und Musterlösung von der Lehrveranstaltungsleitung und die Abgabe direkt von den Studenten. Im Unterschied zum Ausgangsszenario müssen die Tutoren den Vergleich nicht manuell durchführen, sondern können die Abgabe lokal in Protégé auswerten lassen. Die Bewertung erfolgt anhand der vom PlugIn (dem modifizierten Framework) zurückgegebenen Kommentare.

Der Mehrwert gegenüber existierenden Erweiterungen wie OWLdiff (vgl. 3.2.2) besteht in der Ausrichtung auf die didaktische Verwendung und dem Zusatzwissen über OWL-Fehlermodelle.

6.5. Einführen zusätzlicher Fehlertypen und Symptome

Ein weiterer interessanter Fehlertyp wäre, ob eine *Klassifikation vorweggenommen* wurde. Ein solcher Fehler entsteht, wenn in den Axiomen einer OWL-Klasse eine Superklasse explizit angeführt wurde, der Zusammenhang aber erst durch Klassifikation entstehen soll. Der Fehler weist auf ein mangelndes Verständnis einer wesentlichen Eigenschaft von OWL hin, dem logischen Schließen aus gegebenen Fakten. Das dafür nötige Symptom könnte die klassifizierten Superklassen mit den in den Klassenaxiomen erwähnten vergleichen. Ein Beispiel wäre die direkte Definition von $Margherita$ als Subklasse von $VegetarianPizza$.

Ein zweiter wichtiger Punkt wäre die Entwicklung eines alternativen Ansatzes zum Finden trivial erfüllbarer OWL-Klassen (Fehlertyp 8). Wie auf Seite 63 erwähnt ist der Ansatz, eigens erstellte triviale Klassen als Subklassen zu detektieren nicht sinnvoll. Ein Vorgehen über die Klassenaxiome ist möglicherweise vielversprechender.

7. Zusammenfassung und Ausblick

Vorrangiges Ziel dieser Diplomarbeit war die Erweiterung des ITS eTutor um ein Expertenmodul, das die Studierenden beim Erlernen der Beschreibungslogik OWL unterstützt. Dazu wurde nach Analyse möglicher Alternativen ein neues, auf dem Erkennen empirischer Fehlermodelle basierendes Konzept entwickelt. Den Kern dieses Konzepts bilden häufige Probleme beim Modellieren von OWL-Klassen, die durch gut überprüfbare Symptome erkannt werden. Auf diesem theoretischen Fundament wurde ein Expertenmodul in der Programmiersprache Java entwickelt, das Abgabe und Musterlösung sowohl auf semantischer als auch auf logischer Ebene vergleicht. Weiters wurde ein Ablauf für den Lehrbetrieb definiert, in dem Studierende ihre Lösungen im Ontologieeditor Protégé ausarbeiten, via Web-Browser im Learning Management System Moodle abgeben und unmittelbar ihre korrigierte Abgabe erhalten, die sie wiederum überarbeiten können. Die Texte der Rückmeldung wurden internationalisiert und können daher ohne Änderung der Software in beliebige Sprachen übersetzt werden.

Über die Basisanforderungen hinaus wurde großer Wert auf die Erweiterbarkeit des Expertenmoduls gelegt. Das geschaffene Framework kann nicht nur OWL-Klassen, sondern alle im OWL-Standard erwähnten Arten von OWL-Entities untersuchen. Um den Anforderungen im praktischen Einsatz gerecht zu werden, wird der Ablauf der Diagnose rein durch Konfiguration festgelegt und kann ohne Unterbrechung des Produktivbetriebs geändert werden.

Einige Folgeprojekte wurden bereits im vorigen Kapitel skizziert, besonders die Erweiterung auf weitere OWL-Entities und eine detailliertere Beurteilung wären eine logische Folge dieser Arbeit mit direktem Vorteil für die Lehre. Andere Punkte der Weiterentwicklung des Expertenmoduls wären ein Anreichern der Rückmeldung um Verweise auf Lehrmittel oder die Entwicklung und Integration weiterer Fehlermodelle. Des Weiteren wäre eine empirische Überprüfung der Auswirkungen des Expertenmoduls auf den Lernfortschritt der Studierenden interessant. Da längeres Auseinandersetzen mit dem Problem zu verbesserten Leistungen führt [33, 172f] sollte gegenüber der momentanen Situation eine Verbesserung der erreichten Noten zu erkennen sein, denn sowohl das Konstruieren der Ontologie im Editor als auch das wiederholte Durchlaufen des Übungsmodus im ITS erhöhen die Zeit, in der sich Studenten mit dem Thema Ontologiemodellierung befassen.

Anhang

A. Testfälle

Um das erstellte Framework zu testen wurden Ontologien erstellt, die möglichst zu genau einer Schlussfolgerung führen. Anschließend wurden die Ontologien durch die Software mit der Musterlösung verglichen (vgl. Seite 19). Aufgrund von Abhängigkeit der Schlussfolgerungen, bedingt durch gemeinsame Symptome, wurden dabei zwangsläufig auch andere Schlussfolgerungen gefunden.

Alle Analysen wurden mit der Java-Klasse `BucketOTests` durchgeführt (vgl. Seite 65), die gemeinsam mit den Testdaten auf dem beiliegenden Datenträger enthalten ist.

Die folgenden Seiten enthalten jeweils den Testfall als Titel, die abgegebene Ontologie und die tabellarischen Analyseergebnisse. Die Spalten der Tabelle listen die Schlussfolgerungen auf, die Zeilen Symptome. Das Ergebnis der Analyse befindet sich jeweils in der letzten Zeile („`f?`“ steht dabei für „found?“).

Anzumerken ist, dass jede Tabelle die gesammelten Schlussfolgerungen aller analysierter Klassen enthält und daher eine stark aggregierte Darstellung bietet. Beispielsweise bedeutet ein gefundenes `OK` nur, dass mindestens eine geforderte Klasse korrekt formuliert wurde. Die vollständigen Analyseergebnisse finden sich in der jeweiligen kommentierten Ontologie auf dem Datenträger. Die Symbole der Tabellen wurden denen im Musterbeispiel auf Seite 53 nachempfunden:

`*X` ... Symptom positiv und an dieser Stelle relevant
`oX` ... Symptom positiv und an mindestens einer dieser Stellen relevant
`*` ... Symptom negativ und an dieser Stelle relevant
`o` ... Symptom negativ und an mindestens einer dieser Stellen relevant
`X` ... Schlussfolgerung erkannt (in der Zeile `f?`):
Leer ... Feld hier nicht relevant bzw. Schlussfolgerung nicht erkannt

A.0. Korrekte Lösung

Analysierte Datei: `00.owl`

A.0.1. Ontologie

```
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: s: <http://dke.jku.at/2010/student.owl#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: skos: <http://www.w3.org/2004/02/skos/core#>

Ontology: <http://dke.jku.at/2010/student.owl>

Import: <http://dke.jku.at/2010/ingredients.owl>

AnnotationProperty: rdfs:label

ObjectProperty: b:hasPizzaTopping

ObjectProperty: b:hasIngredient

Class: b:Pizza

Class: s:Margherita

    Annotations:
        rdfs:label "Margherita"@

    SubClassOf:
        b:NamedPizza,
        b:hasPizzaTopping some b:MozzarellaTopping,
        b:hasPizzaTopping some b:TomatoTopping,
        b:hasPizzaTopping only
            (b:MozzarellaTopping
             or b:TomatoTopping)

    DisjointWith:
        s:FruttiDiMare

Class: b:TomatoTopping

Class: b:FishTopping

Class: s:CheesyPizza
```

```
    Annotations:
        rdfs:label "CheesyPizza"@

    EquivalentTo:
        b:Pizza
         and (b:hasPizzaTopping some b:CheeseTopping)

Class: b:NamedPizza

Class: b:GarlicTopping

Class: s:FruttiDiMare

    Annotations:
        rdfs:label "FruttiDiMare"@

    SubClassOf:
        b:NamedPizza,
        b:hasPizzaTopping some b:GarlicTopping,
        b:hasPizzaTopping some b:MixedSeafoodTopping,
        b:hasPizzaTopping some b:TomatoTopping,
        b:hasPizzaTopping only
            (b:GarlicTopping
             or b:MixedSeafoodTopping
             or b:TomatoTopping)

    DisjointWith:
        s:Margherita

Class: b:FruitTopping

Class: b:MeatTopping

Class: b:MozzarellaTopping

Class: b:PizzaTopping

Class: b:CheeseTopping

Class: b:MixedSeafoodTopping

Class: b:Food

Class: s:VegetarianPizza

    Annotations:
        rdfs:label "VegetarianPizza"@

    EquivalentTo:
```

```
        b:Pizza
         and (not (b:hasPizzaTopping some b:FishTopping))
         and (not (b:hasPizzaTopping some b:MeatTopping))

Class: s:Icecream

    Annotations:
        rdfs:label "Icecream"@

    SubClassOf:
        b:Food,
        b:hasIngredient some b:FruitTopping

DisjointClasses:
    b:Pizza,b:PizzaTopping,s:Icecream
```

A.0.2. Ergebnis der Analyse

	01	02	03	04	05	06	07	09	10	11	12	13	14	15	16	17	OK
K0		o															
K1						*											oX
K2															*		
K3												*					
K4								*X									
K5											*						
K6													*				
K7								*X					*				
K8											*						
K9														*			
N0	*																*X
N1						o											
N2		*				o											
O1						*											oX
O2			*														
O3							*										
P1									*								
P2	*										o			o			
P3					*						o			o			
P4				*						*	o	*					
P5		o									o						
P6											o		*			*	
P7											o						
f?								X									X

Die gesuchte Schlussfolgerung `OK` wurde gefunden. Aufgrund der verwendeten Quantoren wird auch der Hinweis `09` erzeugt.

A.1. Klasse auf falscher Ebene definiert

Analysierte Datei: `01.owl`

A.1.1. Ontologie

```
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: s: <http://dke.jku.at/2010/student.owl#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: skos: <http://www.w3.org/2004/02/skos/core#>

Ontology: <http://dke.jku.at/2010/student.owl>

Import: <http://dke.jku.at/2010/ingredients.owl>

AnnotationProperty: rdfs:label

ObjectProperty: b:hasPizzaTopping

ObjectProperty: b:hasIngredient

Class: b:Pizza

Class: s:Margherita

    Annotations:
        rdfs:label "Margherita"@

    SubClassOf:
        s:ItalianPizza,
        b:hasPizzaTopping some b:MozzarellaTopping,
        b:hasPizzaTopping some b:TomatoTopping,
        b:hasPizzaTopping only
            (b:MozzarellaTopping
             or b:TomatoTopping)

Class: b:TomatoTopping

Class: b:FishTopping

Class: b:NamedPizza
```

Class: b:GarlicTopping

Class: b:FruitTopping

Class: b:MeatTopping

Class: b:MozzarellaTopping

Class: b:PizzaTopping

Class: b:CheeseTopping

Class: s:ItalianPizza

SubClassOf:
b:Pizza

Class: b:MixedSeafoodTopping

Class: b:Food

A.1.2. Ergebnis der Analyse

	01	02	03	04	05	06	07	09	10	11	12	13	14	15	16	17	OK
K0		oX															
K1					*												o
K2															*		
K3												*					
K4							*										
K5											*						
K6													*				
K7								*					*				
K8											*						
K9														*X			
N0	*X																*
N1					o												
N2		*X			o												
O1					*												o
O2			*X														
O3						*											
P1									*								
P2	*X										o			oX			
P3				*							o			o			
P4			*							*	o	*					
P5		oX									o						

```
P6 |  |  |  |  |  |  |  |  |  |o |  |* |  |  |*X|  |
P7 |  |  |  |  |  |  |  |  |  |o |  |  |  |  |  |  |
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
f? | X| X| X|  |  |  |  |  |  |  |  |  |  | X|  | X|  |
```

Die gesuchte Schlussfolgerung `01` wurde gefunden. Der Aufbau des Testfalls führt zusätzlich zu anderen Schlussfolgerungen.

A.2. Klasse zusätzlich definiert

Analysierte Datei: `02.owl`

A.2.1. Ontologie

```
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: s: <http://dke.jku.at/2010/student.owl#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: skos: <http://www.w3.org/2004/02/skos/core#>

Ontology: <http://dke.jku.at/2010/student.owl>

Import: <http://dke.jku.at/2010/ingredients.owl>

AnnotationProperty: rdfs:label

ObjectProperty: b:hasPizzaTopping

ObjectProperty: b:hasIngredient

Class: b:Pizza

Class: b:TomatoTopping

Class: b:FishTopping

Class: b:NamedPizza

Class: b:GarlicTopping

Class: b:FruitTopping

Class: b:MeatTopping
```

Class: b:MozzarellaTopping

Class: b:PizzaTopping

Class: b:CheeseTopping

Class: s:ItalianPizza

SubClassOf:
b:Pizza

Class: b:MixedSeafoodTopping

Class: b:Food

A.2.2. Ergebnis der Analyse

	01	02	03	04	05	06	07	09	10	11	12	13	14	15	16	17	OK
K0		oX															
K1					*												o
K2															*		
K3												*					
K4							*										
K5											*						
K6													*				
K7							*						*				
K8											*						
K9														*			
N0	*																*
N1					o												
N2		*X			o												
O1					*												o
O2			*X														
O3						*											
P1								*									
P2	*									o			o				
P3				*						o			o				
P4			*						*	o	*						
P5		oX								o							
P6										o		*			*		
P7										o							
f?		X	X														

Die gesuchte Schlussfolgerung O2 wurde gefunden. Da andere geforderte Klassen fehlen werden weitere Schlussfolgerungen gezogen.

A.3. Geforderte Klasse nicht definiert

Analysierte Datei: `03.owl`

A.3.1. Ontologie

```
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: s: <http://dke.jku.at/2010/student.owl#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: skos: <http://www.w3.org/2004/02/skos/core#>

Ontology: <http://dke.jku.at/2010/student.owl>

Import: <http://dke.jku.at/2010/ingredients.owl>

AnnotationProperty: rdfs:label

ObjectProperty: b:hasPizzaTopping

ObjectProperty: b:hasIngredient

Class: b:Pizza

Class: b:FishTopping

Class: b:TomatoTopping

Class: b:GarlicTopping

Class: b:NamedPizza

Class: b:FruitTopping

Class: b:MeatTopping

Class: b:MozzarellaTopping

Class: b:CheeseTopping

Class: b:PizzaTopping

Class: b:Food

Class: b:MixedSeafoodTopping
```

A.3.2. Ergebnis der Analyse

```
   |01|02|03|04|05|06|07|09|10|11|12|13|14|15|16|17|OK|
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
K0 |  |o |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
K1 |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |o |
K2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |* |  |  |
K3 |  |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |
K4 |  |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |
K5 |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |
K6 |  |  |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |
K7 |  |  |  |  |  |  |  |* |  |  |  |  |* |  |  |  |  |
K8 |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |
K9 |  |  |  |  |  |  |  |  |  |  |  |  |  |* |  |  |  |
N0 |* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |* |
N1 |  |  |  |  |  |o |  |  |  |  |  |  |  |  |  |  |  |
N2 |  |* |  |  |  |o |  |  |  |  |  |  |  |  |  |  |  |
O1 |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |o |
O2 |  |  |*X|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
O3 |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |
P1 |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |
P2 |* |  |  |  |  |  |  |  |  |  |o |  |  |o |  |  |  |
P3 |  |  |  |  |* |  |  |  |  |  |o |  |  |o |  |  |  |
P4 |  |  |  |* |  |  |  |  |  |* |o |* |  |  |  |  |  |
P5 |  |o |  |  |  |  |  |  |  |  |o |  |  |  |  |  |  |
P6 |  |  |  |  |  |  |  |  |  |  |o |  |* |  |  |* |  |
P7 |  |  |  |  |  |  |  |  |  |  |o |  |  |  |  |  |  |
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
f? |  |  | X|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
```

Die gesuchte Schlussfolgerung `03` wurde gefunden.

A.4. Konzept unterspezifiziert

Analysierte Datei: `04.owl`

A.4.1. Ontologie

```
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: s: <http://dke.jku.at/2010/student.owl#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

```
Prefix: skos: <http://www.w3.org/2004/02/skos/core#>

Ontology: <http://dke.jku.at/2010/student.owl>

Import: <http://dke.jku.at/2010/ingredients.owl>

AnnotationProperty: rdfs:label

ObjectProperty: b:hasPizzaTopping

ObjectProperty: b:hasIngredient

Class: b:Pizza

Class: s:Margherita

    Annotations:
        rdfs:label "Margherita"@

    SubClassOf:
        b:NamedPizza,
        b:hasPizzaTopping some b:TomatoTopping,
        b:hasPizzaTopping only
            (b:MozzarellaTopping
             or b:TomatoTopping)

Class: b:TomatoTopping

Class: b:FishTopping

Class: b:NamedPizza

Class: b:GarlicTopping

Class: b:FruitTopping

Class: b:MeatTopping

Class: b:MozzarellaTopping

Class: b:PizzaTopping

Class: b:CheeseTopping

Class: b:MixedSeafoodTopping

Class: b:Food
```

A.4.2. Ergebnis der Analyse

	01	02	03	04	05	06	07	09	10	11	12	13	14	15	16	17	OK
K0		o															
K1					*												o
K2															*		
K3												*X					
K4							*										
K5											*						
K6													*				
K7								*					*				
K8											*						
K9														*			
N0	*																*
N1						o											
N2		*				o											
O1						*											o
O2			*X														
O3							*										
P1									*								
P2	*										o			o			
P3					*						o			o			
P4				*X						*X	o	*X					
P5		o									o						
P6											o		*			*X	
P7											o						
f?			X	X						X		X				X	

Die gesuchte Schlussfolgerung 04 wurde gefunden. Der Aufbau des Testfalls führt zusätzlich zu anderen Schlussfolgerungen.

A.5. Konzept überspezifiziert

Analysierte Datei: 05.owl

A.5.1. Ontologie

```
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: s: <http://dke.jku.at/2010/student.owl#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: skos: <http://www.w3.org/2004/02/skos/core#>

Ontology: <http://dke.jku.at/2010/student.owl>

Import: <http://dke.jku.at/2010/ingredients.owl>

AnnotationProperty: rdfs:label

ObjectProperty: b:hasPizzaTopping

ObjectProperty: b:hasIngredient

Class: b:Pizza

Class: s:SunnyPizza

    EquivalentTo:
        b:Pizza
         and (b:hasPizzaTopping some b:SundriedTomatoTopping)

Class: s:Margherita

    Annotations:
        rdfs:label "Margherita"@

    SubClassOf:
        b:NamedPizza,
        b:hasPizzaTopping some b:MozzarellaTopping,
        b:hasPizzaTopping some b:SundriedTomatoTopping,
        b:hasPizzaTopping only
            (b:MozzarellaTopping
             or b:SundriedTomatoTopping)

Class: b:TomatoTopping

Class: b:FishTopping

Class: b:NamedPizza

Class: b:GarlicTopping

Class: b:FruitTopping

Class: b:MeatTopping

Class: b:MozzarellaTopping

Class: b:PizzaTopping
```

Class: b:CheeseTopping

Class: b:MixedSeafoodTopping

Class: b:Food

Class: b:SundriedTomatoTopping

A.5.2. Ergebnis der Analyse

```
   |01|02|03|04|05|06|07|09|10|11|12|13|14|15|16|17|OK|
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
K0 |  |oX|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
K1 |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |o |
K2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |* |  |  |
K3 |  |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |
K4 |  |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |
K5 |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |
K6 |  |  |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |
K7 |  |  |  |  |  |  |  |* |  |  |  |  |* |  |  |  |  |
K8 |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |
K9 |  |  |  |  |  |  |  |  |  |  |  |  |  |*X|  |  |  |
N0 |* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |* |
N1 |  |  |  |  |  |o |  |  |  |  |  |  |  |  |  |  |  |
N2 |  |*X|  |  |  |o |  |  |  |  |  |  |  |  |  |  |  |
O1 |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |o |
O2 |  |  |*X|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
O3 |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |
P1 |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |
P2 |* |  |  |  |  |  |  |  |  |  |o |  |  |o |  |  |  |
P3 |  |  |  |  |*X|  |  |  |  |  |o |  |  |oX|  |  |  |
P4 |  |  |  |* |  |  |  |  |  |* |o |* |  |  |  |  |  |
P5 |  |oX|  |  |  |  |  |  |  |  |o |  |  |  |  |  |  |
P6 |  |  |  |  |  |  |  |  |  |  |o |  |* |  |  |*X|  |
P7 |  |  |  |  |  |  |  |  |  |  |o |  |  |  |  |  |  |
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
f? |  | X| X|  | X|  |  |  |  |  |  |  |  | X|  | X|  |
```

Die gesuchte Schlussfolgerung `05` wurde gefunden. Der Aufbau des Testfalls führt zusätzlich zu anderen Schlussfolgerungen.

A.6. Namenskonvention nicht eingehalten

Analysierte Datei: `06.owl`

A.6.1. Ontologie

```
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: s: <http://dke.jku.at/2010/student.owl#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: skos: <http://www.w3.org/2004/02/skos/core#>

Ontology: <http://dke.jku.at/2010/student.owl>

Import: <http://dke.jku.at/2010/ingredients.owl>

AnnotationProperty: rdfs:label

ObjectProperty: b:hasPizzaTopping

ObjectProperty: b:hasIngredient

Class: b:Pizza

Class: s:Margarita

    Annotations:
        rdfs:label "Margherita"@

    SubClassOf:
        b:NamedPizza,
        b:hasPizzaTopping some b:MozzarellaTopping,
        b:hasPizzaTopping some b:TomatoTopping,
        b:hasPizzaTopping only
            (b:MozzarellaTopping
             or b:TomatoTopping)

Class: b:TomatoTopping

Class: b:FishTopping

Class: b:NamedPizza

Class: b:GarlicTopping

Class: b:FruitTopping

Class: b:MeatTopping

Class: b:MozzarellaTopping
```

Class: b:PizzaTopping

Class: b:CheeseTopping

Class: b:MixedSeafoodTopping

Class: b:Food

A.6.2. Ergebnis der Analyse

```
   |01|02|03|04|05|06|07|09|10|11|12|13|14|15|16|17|OK|
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
K0 |  |o |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
K1 |  |  |  |  |  |*X|  |  |  |  |  |  |  |  |  |  |o |
K2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |* |  |  |
K3 |  |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |
K4 |  |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |
K5 |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |
K6 |  |  |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |
K7 |  |  |  |  |  |  |  |* |  |  |  |  |* |  |  |  |  |
K8 |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |
K9 |  |  |  |  |  |  |  |  |  |  |  |  |  |* |  |  |  |
N0 |* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |* |
N1 |  |  |  |  |  |oX|  |  |  |  |  |  |  |  |  |  |  |
N2 |  |* |  |  |  |o |  |  |  |  |  |  |  |  |  |  |  |
O1 |  |  |  |  |  |*X|  |  |  |  |  |  |  |  |  |  |o |
O2 |  |  |*X|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
O3 |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |
P1 |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |
P2 |* |  |  |  |  |  |  |  |  |  |o |  |  |o |  |  |  |
P3 |  |  |  |  |* |  |  |  |  |  |o |  |  |o |  |  |  |
P4 |  |  |  |* |  |  |  |  |  |* |o |* |  |  |  |  |  |
P5 |  |o |  |  |  |  |  |  |  |  |o |  |  |  |  |  |  |
P6 |  |  |  |  |  |  |  |  |  |  |o |  |* |  |  |* |  |
P7 |  |  |  |  |  |  |  |  |  |  |o |  |  |  |  |  |  |
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
f? |  |  | X|  |  | X|  |  |  |  |  |  |  |  |  |  |  |
```

Die gesuchte Schlussfolgerung `06` wurde gefunden. Da andere geforderte Klassen fehlen wird auch die Schlussfolgerung `03` gezogen.

A.7. Basisontologie verändert

Analysierte Datei: `07.owl`

A.7.1. Ontologie

```
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: s: <http://dke.jku.at/2010/student.owl#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: skos: <http://www.w3.org/2004/02/skos/core#>

Ontology: <http://dke.jku.at/2010/student.owl>

Import: <http://dke.jku.at/2010/ingredients.owl>

Annotations:
    rdfs:comment "contains merged base ontology for FT7"@,
    owl:versionInfo "basiert auf ..."@de

AnnotationProperty: owl:versionInfo

AnnotationProperty: rdfs:label

AnnotationProperty: rdfs:comment

ObjectProperty: b:hasPizzaTopping

    Annotations:
        rdfs:comment "Note that ..."@en

    SubPropertyOf:
        b:hasIngredient

    Characteristics:
        InverseFunctional

    Domain:
        b:Pizza

    Range:
        b:PizzaTopping

    InverseOf:
        b:isToppingOfPizza

ObjectProperty: b:isIngredientOf

    Annotations:
        rdfs:comment "The inverse ..."@en
```

```
    Characteristics:
        Transitive

    Domain:
        b:Food

    Range:
        b:Food

    InverseOf:
        b:hasIngredient

ObjectProperty: b:isToppingOfPizza

    Annotations:
        rdfs:comment "Any given instance ..."@en

    SubPropertyOf:
        b:isIngredientOf

    Characteristics:
        Functional

    Domain:
        b:PizzaTopping

    Range:
        b:Pizza

    InverseOf:
        b:hasPizzaTopping

ObjectProperty: b:hasIngredient

    Annotations:
        rdfs:comment "NB Transitive ..."@en

    Characteristics:
        Transitive

    Domain:
        b:Food

    Range:
        b:Food

    InverseOf:
        b:isIngredientOf
```

```
Class: owl:Thing

Class: b:Pizza

    Annotations:
        rdfs:label "Pizza"@en

    SubClassOf:
        b:Food

    DisjointWith:
        b:PizzaTopping

Class: b:FishTopping

    Annotations:
        rdfs:label "CoberturaDePeixe"@pt

    SubClassOf:
        b:PizzaTopping

    DisjointWith:
        b:CheeseTopping, b:FruitTopping, b:MeatTopping, b:VegetableTopping

Class: b:TomatoTopping

    Annotations:
        rdfs:label "CoberturaDeTomate"@pt

    SubClassOf:
        b:VegetableTopping

    DisjointWith:
        b:GarlicTopping

Class: b:PeperoniSausageTopping

    Annotations:
        rdfs:label "CoberturaDeCalabreza"@pt

    SubClassOf:
        s:SausageTopping

    DisjointWith:
        b:HamTopping

Class: b:SlicedTomatoTopping
```

```
    Annotations:
        rdfs:label "CoberturaDeTomateFatiado"@pt

    SubClassOf:
        b:TomatoTopping

    DisjointWith:
        b:SundriedTomatoTopping

Class: b:NamedPizza

    Annotations:
        rdfs:comment "A pizza that can be found on a pizza menu"@en,
        rdfs:label "PizzaComUmNome"@pt

    SubClassOf:
        b:Pizza

Class: b:GarlicTopping

    Annotations:
        rdfs:label "CoberturaDeAlho"@pt

    SubClassOf:
        b:VegetableTopping

    DisjointWith:
        b:TomatoTopping

Class: b:VegetableTopping

    Annotations:
        rdfs:label "CoberturaDeVegetais"@pt

    SubClassOf:
        b:PizzaTopping

    DisjointWith:
        b:CheeseTopping, b:FishTopping, b:FruitTopping, b:MeatTopping

Class: b:FruitTopping

    Annotations:
        rdfs:label "CoberturaDeFrutas"@pt

    SubClassOf:
        b:PizzaTopping

    DisjointWith:
```

```
        b:CheeseTopping, b:FishTopping, b:MeatTopping, b:VegetableTopping

Class: b:MeatTopping

    Annotations:
        rdfs:label "CoberturaDeCarne"@pt

    SubClassOf:
        b:PizzaTopping

    DisjointWith:
        b:CheeseTopping, b:FishTopping, b:FruitTopping, b:VegetableTopping

Class: b:MozzarellaTopping

    Annotations:
        rdfs:label "CoberturaDeMozzarella"@pt

    SubClassOf:
        b:CheeseTopping

Class: b:CheeseTopping

    Annotations:
        rdfs:label "CoberturaDeQueijo"@pt

    SubClassOf:
        b:PizzaTopping

    DisjointWith:
        b:FishTopping, b:FruitTopping, b:MeatTopping, b:VegetableTopping

Class: b:PizzaTopping

    Annotations:
        rdfs:label "CoberturaDaPizza"@pt

    SubClassOf:
        b:Food

    DisjointWith:
        b:Pizza

Class: s:SausageTopping

    SubClassOf:
        b:MeatTopping

Class: b:MixedSeafoodTopping
```

```
    Annotations:
        rdfs:label "CoberturaDeFrutosDoMarMistos"@pt

    SubClassOf:
        b:FishTopping

Class: b:Food

    SubClassOf:
        owl:Thing

Class: b:SundriedTomatoTopping

    Annotations:
        rdfs:label "CoberturaDeTomateRessecadoAoSol"@pt

    SubClassOf:
        b:TomatoTopping

    DisjointWith:
        b:SlicedTomatoTopping

Class: b:ParmaHamTopping

    Annotations:
        rdfs:label "CoberturaDePrezuntoParma"@pt

    SubClassOf:
        b:HamTopping

Class: b:HamTopping

    Annotations:
        rdfs:label "CoberturaDePresunto"@pt

    SubClassOf:
        b:MeatTopping

    DisjointWith:
        b:PeperoniSausageTopping
```

A.7.2. Ergebnis der Analyse

```
   |01|02|03|04|05|06|07|09|10|11|12|13|14|15|16|17|OK|
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
K0 |  |oX|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
K1 |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |o |
K2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |* |  |  |
```

```
K3 |  |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |
K4 |  |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |
K5 |  |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |
K6 |  |  |  |  |  |  |  |  |  |  |  |  |  |* |  |  |  |
K7 |  |  |  |  |  |  |  |* |  |  |  |  |  |* |  |  |  |
K8 |  |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |
K9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |* |  |  |
N0 |* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |* |
N1 |  |  |  |  |  |  |o |  |  |  |  |  |  |  |  |  |  |  |
N2 |  |  |*X|  |  |  |o |  |  |  |  |  |  |  |  |  |  |  |
O1 |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |o |
O2 |  |  |  |*X|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
O3 |  |  |  |  |  |  |  |*X|  |  |  |  |  |  |  |  |  |  |
P1 |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |
P2 |* |  |  |  |  |  |  |  |  |  |  |o |  |  |o |  |  |  |
P3 |  |  |  |  |  |* |  |  |  |  |  |o |  |  |o |  |  |  |
P4 |  |  |  |  |* |  |  |  |  |  |* |o |* |  |  |  |  |  |
P5 |  |  |oX|  |  |  |  |  |  |  |  |o |  |  |  |  |  |  |
P6 |  |  |  |  |  |  |  |  |  |  |  |o |  |* |  |  |  |* |
P7 |  |  |  |  |  |  |  |  |  |  |  |o |  |  |  |  |  |  |
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
f? |  |  | X|  | X|  |  |  | X|  |  |  |  |  |  |  |  |  |
```

Die gesuchte Schlussfolgerung `07` wurde gefunden. Der Aufbau des Testfalls führt zusätzlich zu anderen Schlussfolgerungen.

A.8. Bedingung durch leere Menge trivial erfüllbar

Wie im Teil *Umsetzung* beschrieben wurde dieser Fehlertyp nicht implementiert.

A.9. Verneinung falsch positioniert

Analysierte Datei: `09.owl`

A.9.1. Ontologie

```
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: s: <http://dke.jku.at/2010/student.owl#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: skos: <http://www.w3.org/2004/02/skos/core#>
```

```
Ontology: <http://dke.jku.at/2010/student.owl>

Import: <http://dke.jku.at/2010/ingredients.owl>

AnnotationProperty: rdfs:label

ObjectProperty: b:hasPizzaTopping

ObjectProperty: b:hasIngredient

Class: b:Pizza

Class: s:Margherita

    Annotations:
        rdfs:label "Margherita"@

    SubClassOf:
        b:NamedPizza,
        b:hasPizzaTopping some b:MozzarellaTopping,
        b:hasPizzaTopping some b:TomatoTopping,
        b:hasPizzaTopping only
            (b:MozzarellaTopping
             or b:TomatoTopping)

    DisjointWith:
        s:FruttiDiMare

Class: b:TomatoTopping

Class: b:FishTopping

Class: s:CheesyPizza

    Annotations:
        rdfs:label "CheesyPizza"@

    EquivalentTo:
        b:Pizza
         and (b:hasPizzaTopping some b:CheeseTopping)

Class: b:NamedPizza

Class: b:GarlicTopping

Class: s:FruttiDiMare

    Annotations:
```

```
        rdfs:label "FruttiDiMare"@

    SubClassOf:
        b:NamedPizza,
        b:hasPizzaTopping some b:GarlicTopping,
        b:hasPizzaTopping some b:MixedSeafoodTopping,
        b:hasPizzaTopping some b:TomatoTopping,
        b:hasPizzaTopping only
            (b:GarlicTopping
             or b:MixedSeafoodTopping
             or b:TomatoTopping)

    DisjointWith:
        s:Margherita

Class: b:FruitTopping

Class: b:MeatTopping

Class: b:MozzarellaTopping

Class: b:PizzaTopping

Class: b:CheeseTopping

Class: b:MixedSeafoodTopping

Class: b:Food

Class: s:VegetarianPizza

    Annotations:
        rdfs:label "VegetarianPizza"@

    EquivalentTo:
        b:Pizza
         and (b:hasPizzaTopping some (not (b:FishTopping)))
         and (b:hasPizzaTopping some (not (b:MeatTopping)))

Class: s:Icecream

    Annotations:
        rdfs:label "Icecream"@

    SubClassOf:
        b:Food,
        b:hasIngredient some b:FruitTopping

DisjointClasses:
```

b : Pizza , b : PizzaTopping , s : Icecream

A.9.2. Ergebnis der Analyse

```
   |01|02|03|04|05|06|07|09|10|11|12|13|14|15|16|17|OK|
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
K0 |  |o |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
K1 |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |oX|
K2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |* |  |  |
K3 |  |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |
K4 |  |  |  |  |  |  |  |*X|  |  |  |  |  |  |  |  |  |
K5 |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |
K6 |  |  |  |  |  |  |  |  |  |  |  |  |*X|  |  |  |  |
K7 |  |  |  |  |  |  |  |*X|  |  |  |  |*X|  |  |  |  |
K8 |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |
K9 |  |  |  |  |  |  |  |  |  |  |  |  |  |*X|  |  |  |
N0 |*X|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |*X|
N1 |  |  |  |  |  |o |  |  |  |  |  |  |  |  |  |  |  |
N2 |  |* |  |  |  |o |  |  |  |  |  |  |  |  |  |  |  |
O1 |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |oX|
O2 |  |  |* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
O3 |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |
P1 |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |
P2 |*X|  |  |  |  |  |  |  |  |  |o |  |  |oX|  |  |  |
P3 |  |  |  |  |* |  |  |  |  |  |o |  |  |o |  |  |  |
P4 |  |  |  |* |  |  |  |  |  |* |o |* |  |  |  |  |  |
P5 |  |o |  |  |  |  |  |  |  |  |o |  |  |  |  |  |  |
P6 |  |  |  |  |  |  |  |  |  |  |o |  |*X|  |  |*X|  |
P7 |  |  |  |  |  |  |  |  |  |  |o |  |  |  |  |  |  |
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
f? | X|  |  |  |  |  |  | X|  |  |  |  | X| X|  | X| X|
```

Die gesuchte Schlussfolgerung `09` wurde gefunden. Die Schlussfolgerung `OK` stammt von den korrekt definierten geforderten Klassen, weitere Schlussfolgerungen vom restlichen Aufbau des Testfalls.

A.10. Widerspruch innerhalb der Ontologie

Analysierte Datei: `10.owl`

A.10.1. Ontologie

```
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: : <http://www.w3.org/2002/07/owl#>
```

```
Prefix: s: <http://dke.jku.at/2010/student.owl#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: skos: <http://www.w3.org/2004/02/skos/core#>

Ontology: <http://dke.jku.at/2010/student.owl>

Import: <http://dke.jku.at/2010/ingredients.owl>

AnnotationProperty: rdfs:label

ObjectProperty: b:hasPizzaTopping

ObjectProperty: b:hasIngredient

Class: b:Pizza

Class: b:TomatoTopping

Class: b:FishTopping

Class: b:NamedPizza

Class: b:GarlicTopping

Class: b:FruitTopping

Class: b:MeatTopping

Class: b:MozzarellaTopping

Class: b:PizzaTopping

Class: b:CheeseTopping

Class: b:MixedSeafoodTopping

Class: b:Food

Class: s:Icecream

    Annotations:
        rdfs:label "Icecream"@

    SubClassOf:
        b:Food,
        b:hasPizzaTopping some b:FruitTopping
```

```
DisjointClasses:
    b:Pizza, b:PizzaTopping, s:Icecream
```

A.10.2. Ergebnis der Analyse

```
   |01|02|03|04|05|06|07|09|10|11|12|13|14|15|16|17|OK|
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
K0 |  |o |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
K1 |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |o |
K2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |* |  |  |
K3 |  |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |
K4 |  |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |
K5 |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |
K6 |  |  |  |  |  |  |  |  |  |  |  |  |*X|  |  |  |  |
K7 |  |  |  |  |  |  |  |* |  |  |  |  |*X|  |  |  |  |
K8 |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |
K9 |  |  |  |  |  |  |  |  |  |  |  |  |  |*X|  |  |  |
N0 |* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |* |
N1 |  |  |  |  |  |o |  |  |  |  |  |  |  |  |  |  |  |
N2 |  |* |  |  |  |o |  |  |  |  |  |  |  |  |  |  |  |
O1 |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |o |
O2 |  |  |*X|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
O3 |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |
P1 |  |  |  |  |  |  |  |  |*X|  |  |  |  |  |  |  |  |
P2 |* |  |  |  |  |  |  |  |  |  |o |  |  |o |  |  |  |
P3 |  |  |  |  |*X|  |  |  |  |  |o |  |  |oX|  |  |  |
P4 |  |  |  |* |  |  |  |  |  |* |o |* |  |  |  |  |  |
P5 |  |o |  |  |  |  |  |  |  |  |o |  |  |  |  |  |  |
P6 |  |  |  |  |  |  |  |  |  |  |o |  |*X|  |  |*X|  |
P7 |  |  |  |  |  |  |  |  |  |  |o |  |  |  |  |  |  |
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
f? |  |  | X|  | X|  |  |  | X|  |  |  | X| X|  | X|  |
```

Die gesuchte Schlussfolgerung `10` wurde gefunden. Der Aufbau des Testfalls führt zusätzlich zu anderen Schlussfolgerungen.

A.11. Implizites Wissen nicht explizit ausgedrückt

Analysierte Datei: `11.owl`

A.11.1. Ontologie

```
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: : <http://www.w3.org/2002/07/owl#>
```

```
Prefix: s: <http://dke.jku.at/2010/student.owl#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: skos: <http://www.w3.org/2004/02/skos/core#>

Ontology: <http://dke.jku.at/2010/student.owl>

Import: <http://dke.jku.at/2010/ingredients.owl>

AnnotationProperty: rdfs:label

ObjectProperty: b:hasPizzaTopping

ObjectProperty: b:hasIngredient

Class: b:Pizza

Class: s:Margherita

    Annotations:
        rdfs:label "Margherita"@

    SubClassOf:
        b:NamedPizza,
        b:hasPizzaTopping some b:TomatoTopping,
        b:hasPizzaTopping only
            (b:MozzarellaTopping
             or b:TomatoTopping)

Class: b:TomatoTopping

Class: b:FishTopping

Class: b:NamedPizza

Class: b:GarlicTopping

Class: b:FruitTopping

Class: b:MeatTopping

Class: b:MozzarellaTopping

Class: b:PizzaTopping

Class: b:CheeseTopping

Class: b:MixedSeafoodTopping
```

Class: b:Food

A.11.2. Ergebnis der Analyse

	01	02	03	04	05	06	07	09	10	11	12	13	14	15	16	17	OK
K0		o															
K1					*												o
K2															*		
K3												*X					
K4							*										
K5											*						
K6													*				
K7								*					*				
K8											*						
K9														*			
N0	*																*
N1						o											
N2		*				o											
O1						*											o
O2			*X														
O3							*										
P1									*								
P2	*										o			o			
P3					*						o			o			
P4				*X						*X	o	*X					
P5		o									o						
P6											o		*			*X	
P7											o						
f?			X	X						X		X				X	

Die gesuchte Schlussfolgerung `11` wurde gefunden. Der Aufbau des Testfalls führt zusätzlich zu anderen Schlussfolgerungen.

A.12. Allquantor statt Existenzquantor verwendet

Analysierte Datei: `12.owl`

A.12.1. Ontologie

```
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: : <http://www.w3.org/2002/07/owl#>
```

```
Prefix: s: <http://dke.jku.at/2010/student.owl#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: skos: <http://www.w3.org/2004/02/skos/core#>

Ontology: <http://dke.jku.at/2010/student.owl>

Import: <http://dke.jku.at/2010/ingredients.owl>

AnnotationProperty: rdfs:label

ObjectProperty: b:hasPizzaTopping

ObjectProperty: b:hasIngredient

Class: b:Pizza

Class: b:TomatoTopping

Class: b:FishTopping

Class: s:CheesyPizza

    Annotations:
        rdfs:label "CheesyPizza"@

    EquivalentTo:
        b:Pizza
         and (b:hasPizzaTopping only b:CheeseTopping)

Class: b:NamedPizza

Class: b:GarlicTopping

Class: b:FruitTopping

Class: b:MeatTopping

Class: b:MozzarellaTopping

Class: b:PizzaTopping

Class: b:CheeseTopping

Class: b:MixedSeafoodTopping

Class: b:Food
```

A.12.2. Ergebnis der Analyse

	01	02	03	04	05	06	07	09	10	11	12	13	14	15	16	17	OK
K0		o															
K1					*												o
K2															*		
K3												*					
K4								*									
K5											*X						
K6													*				
K7								*					*				
K8											*X						
K9														*X			
N0	*X																*
N1						o											
N2		*				o											
O1						*											o
O2			*X														
O3							*										
P1									*								
P2	*X										oX			oX			
P3					*						o			o			
P4				*						*	o	*					
P5		o									oX						
P6											oX		*			*X	
P7											oX						
f?	X		X								X			X		X	

Die gesuchte Schlussfolgerung 12 wurde gefunden. Der Aufbau des Testfalls führt zusätzlich zu anderen Schlussfolgerungen.

A.13. Probleme mit logischem Oder

Analysierte Datei: 13.owl

A.13.1. Ontologie

```
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: s: <http://dke.jku.at/2010/student.owl#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: skos: <http://www.w3.org/2004/02/skos/core#>

Ontology: <http://dke.jku.at/2010/student.owl>

Import: <http://dke.jku.at/2010/ingredients.owl>

AnnotationProperty: rdfs:label

ObjectProperty: b:hasPizzaTopping

ObjectProperty: b:hasIngredient

Class: b:Pizza

Class: b:TomatoTopping

Class: b:FishTopping

Class: b:NamedPizza

Class: b:GarlicTopping

Class: b:FruitTopping

Class: b:MeatTopping

Class: b:MozzarellaTopping

Class: b:PizzaTopping

Class: b:CheeseTopping

Class: s:VegetarianPizza

    Annotations:
        rdfs:label "VegetarianPizza"@

    EquivalentTo:
        b:Pizza
         and ((not (b:hasPizzaTopping some b:FishTopping))
         or (not (b:hasPizzaTopping some b:MeatTopping)))

Class: b:MixedSeafoodTopping

Class: b:Food
```

A.13.2. Ergebnis der Analyse

	01	02	03	04	05	06	07	09	10	11	12	13	14	15	16	17	OK
K0		o															
K1					*												o
K2															*		
K3												*X					
K4								*X									
K5											*						
K6													*X				
K7								*X					*X				
K8											*						
K9														*			
N0	*																*
N1						o											
N2		*				o											
O1						*											o
O2			*X														
O3							*										
P1									*								
P2	*										o			o			
P3					*						o			o			
P4				*X						*X	o	*X					
P5		o									o						
P6											o		*X			*X	
P7											o						
f?			X	X				X		X		X	X			X	

Die gesuchte Schlussfolgerung 13 wurde gefunden. Der Aufbau des Testfalls führt zusätzlich zu anderen Schlussfolgerungen.

A.14. Open World Assumption bei Klassen nicht beachtet

Analysierte Datei: 14.owl

A.14.1. Ontologie

```
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: s: <http://dke.jku.at/2010/student.owl#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: skos: <http://www.w3.org/2004/02/skos/core#>

Ontology: <http://dke.jku.at/2010/student.owl>

Import: <http://dke.jku.at/2010/ingredients.owl>

AnnotationProperty: rdfs:label

ObjectProperty: b:hasPizzaTopping

ObjectProperty: b:hasIngredient

Class: b:Pizza

Class: s:Margherita

    Annotations:
        rdfs:label "Margherita"@

    SubClassOf:
        b:NamedPizza,
        b:hasPizzaTopping some b:MozzarellaTopping,
        b:hasPizzaTopping some b:TomatoTopping

Class: b:TomatoTopping

Class: b:FishTopping

Class: b:NamedPizza

Class: b:GarlicTopping

Class: b:FruitTopping

Class: b:MeatTopping

Class: b:MozzarellaTopping

Class: b:PizzaTopping

Class: b:CheeseTopping

Class: b:MixedSeafoodTopping

Class: b:Food
```

A.14.2. Ergebnis der Analyse

	01	02	03	04	05	06	07	09	10	11	12	13	14	15	16	17	OK
K0		o															
K1					*												o
K2															*		
K3												*					
K4								*									
K5											*						
K6													*X				
K7								*					*X				
K8											*						
K9														*			
N0	*																*
N1						o											
N2		*				o											
O1						*											o
O2			*X														
O3							*										
P1									*								
P2	*										o			o			
P3					*						o			o			
P4				*X						*X	o	*					
P5		o									o						
P6											o		*X			*X	
P7											o						
f?			X	X						X			X			X	

Die gesuchte Schlussfolgerung `14` wurde gefunden. Der Aufbau des Testfalls führt zusätzlich zu anderen Schlussfolgerungen.

A.15. Einschränkungen von Werte- und Zielbereich sind auch Axiome

Analysierte Datei: `15.owl`

A.15.1. Ontologie

```
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: s: <http://dke.jku.at/2010/student.owl#>
```

```
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: skos: <http://www.w3.org/2004/02/skos/core#>

Ontology: <http://dke.jku.at/2010/student.owl>

Import: <http://dke.jku.at/2010/ingredients.owl>

AnnotationProperty: rdfs:label

ObjectProperty: b:hasPizzaTopping

ObjectProperty: b:hasIngredient

Class: b:Pizza

Class: b:TomatoTopping

Class: b:FishTopping

Class: b:NamedPizza

Class: b:GarlicTopping

Class: b:FruitTopping

Class: s:NoFoodForMe

    DisjointWith:
        b:Food

Class: b:MeatTopping

Class: b:MozzarellaTopping

Class: b:PizzaTopping

Class: b:CheeseTopping

Class: b:MixedSeafoodTopping

Class: b:Food

    DisjointWith:
        s:NoFoodForMe

Class: s:Icecream
```

```
    Annotations:
        rdfs:label "Icecream"@

    SubClassOf:
        s:NoFoodForMe,
        b:hasIngredient some b:FruitTopping

DisjointClasses:
    b:Pizza,b:PizzaTopping,s:Icecream
```

A.15.2. Ergebnis der Analyse

```
   |01|02|03|04|05|06|07|09|10|11|12|13|14|15|16|17|OK|
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
K0 |  |oX|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
K1 |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |o |
K2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |* |  |  |
K3 |  |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |
K4 |  |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |
K5 |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |
K6 |  |  |  |  |  |  |  |  |  |  |  |  |*X|  |  |  |  |
K7 |  |  |  |  |  |  |  |* |  |  |  |  |*X|  |  |  |  |
K8 |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |
K9 |  |  |  |  |  |  |  |  |  |  |  |  |  |*X|  |  |  |
N0 |* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |* |
N1 |  |  |  |  |  |o |  |  |  |  |  |  |  |  |  |  |  |
N2 |  |*X|  |  |  |o |  |  |  |  |  |  |  |  |  |  |  |
O1 |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |o |
O2 |  |  |*X|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
O3 |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |
P1 |  |  |  |  |  |  |  |  |*X|  |  |  |  |  |  |  |  |
P2 |* |  |  |  |  |  |  |  |  |  |o |  |  |o |  |  |  |
P3 |  |  |  |  |*X|  |  |  |  |  |o |  |  |oX|  |  |  |
P4 |  |  |  |* |  |  |  |  |  |* |o |* |  |  |  |  |  |
P5 |  |oX|  |  |  |  |  |  |  |  |o |  |  |  |  |  |  |
P6 |  |  |  |  |  |  |  |  |  |  |o |  |*X|  |  |*X|  |
P7 |  |  |  |  |  |  |  |  |  |  |o |  |  |  |  |  |  |
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
f? |  | X| X|  | X|  |  |  | X|  |  |  | X| X|  | X|  |
```

Die gesuchte Schlussfolgerung `15` wurde gefunden. Der Aufbau des Testfalls führt zusätzlich zu anderen Schlussfolgerungen.

A.16. Open World Assumption bei Properties nicht beachtet

Analysierte Datei: `16.owl`

A.16.1. Ontologie

```
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: s: <http://dke.jku.at/2010/student.owl#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: skos: <http://www.w3.org/2004/02/skos/core#>

Ontology: <http://dke.jku.at/2010/student.owl>

Import: <http://dke.jku.at/2010/ingredients.owl>

AnnotationProperty: rdfs:label

ObjectProperty: b:hasPizzaTopping

ObjectProperty: b:hasIngredient

Class: b:Pizza

Class: b:TomatoTopping

Class: b:FishTopping

Class: s:CheesyPizza

    Annotations:
        rdfs:label "CheesyPizza"@

    SubClassOf:
        b:Pizza
         and (b:hasPizzaTopping some b:CheeseTopping)

Class: b:NamedPizza

Class: b:GarlicTopping

Class: b:FruitTopping

Class: b:MeatTopping

Class: b:MozzarellaTopping

Class: b:PizzaTopping

Class: b:CheeseTopping
```

Class: b:MixedSeafoodTopping

Class: b:Food

A.16.2. Ergebnis der Analyse

```
   |01|02|03|04|05|06|07|09|10|11|12|13|14|15|16|17|OK|
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
K0 |  |o |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
K1 |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |o |
K2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |*X|  |  |
K3 |  |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |
K4 |  |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |
K5 |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |
K6 |  |  |  |  |  |  |  |  |  |  |  |  |*X|  |  |  |  |
K7 |  |  |  |  |  |  |  |* |  |  |  |  |*X|  |  |  |  |
K8 |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |
K9 |  |  |  |  |  |  |  |  |  |  |  |  |  |*X|  |  |  |
N0 |* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |* |
N1 |  |  |  |  |  |o |  |  |  |  |  |  |  |  |  |  |  |
N2 |  |* |  |  |  |o |  |  |  |  |  |  |  |  |  |  |  |
O1 |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |o |
O2 |  |  |*X|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
O3 |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |
P1 |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |
P2 |* |  |  |  |  |  |  |  |  |  |o |  |  |o |  |  |  |
P3 |  |  |  |  |*X|  |  |  |  |  |o |  |  |oX|  |  |  |
P4 |  |  |  |* |  |  |  |  |  |* |o |* |  |  |  |  |  |
P5 |  |o |  |  |  |  |  |  |  |  |o |  |  |  |  |  |  |
P6 |  |  |  |  |  |  |  |  |  |  |o |  |*X|  |  |*X|  |
P7 |  |  |  |  |  |  |  |  |  |  |o |  |  |  |  |  |  |
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
f? |  |  | X|  | X|  |  |  |  |  |  |  | X| X| X| X|  |
```

Die gesuchte Schlussfolgerung `16` wurde gefunden. Der Aufbau des Testfalls führt zusätzlich zu anderen Schlussfolgerungen.

A.17. Überlappung von Klassen nicht beachtet

Analysierte Datei: `17.owl`

A.17.1. Ontologie

```
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: s: <http://dke.jku.at/2010/student.owl#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: skos: <http://www.w3.org/2004/02/skos/core#>

Ontology: <http://dke.jku.at/2010/student.owl>

Import: <http://dke.jku.at/2010/ingredients.owl>

AnnotationProperty: rdfs:label

ObjectProperty: b:hasPizzaTopping

ObjectProperty: b:hasIngredient

Class: b:Pizza

Class: s:Margherita

    Annotations:
        rdfs:label "Margherita"@

    SubClassOf:
        b:NamedPizza,
        b:hasPizzaTopping some b:MozzarellaTopping,
        b:hasPizzaTopping some s:NewVegetarianTopping,
        b:hasPizzaTopping only
            (b:MozzarellaTopping
             or s:NewVegetarianTopping)

Class: b:TomatoTopping

Class: b:FishTopping

Class: b:NamedPizza

Class: b:GarlicTopping

Class: b:FruitTopping

Class: b:MeatTopping

Class: b:MozzarellaTopping
```

```
Class: b:PizzaTopping

Class: b:CheeseTopping

Class: s:NewVegetarianTopping

    SubClassOf:
        b:PizzaTopping

Class: b:MixedSeafoodTopping

Class: b:Food
```

A.17.2. Ergebnis der Analyse

```
   |01|02|03|04|05|06|07|09|10|11|12|13|14|15|16|17|OK|
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
K0 |  |oX|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
K1 |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |o |
K2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |* |  |  |
K3 |  |  |  |  |  |  |  |  |  |  |  |*X|  |  |  |  |  |
K4 |  |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |
K5 |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |
K6 |  |  |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |
K7 |  |  |  |  |  |  |  |* |  |  |  |  |* |  |  |  |  |
K8 |  |  |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |
K9 |  |  |  |  |  |  |  |  |  |  |  |  |  |* |  |  |  |
N0 |* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |* |
N1 |  |  |  |  |  |o |  |  |  |  |  |  |  |  |  |  |  |
N2 |  |*X|  |  |  |o |  |  |  |  |  |  |  |  |  |  |  |
O1 |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |o |
O2 |  |  |*X|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
O3 |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |  |  |
P1 |  |  |  |  |  |  |  |  |* |  |  |  |  |  |  |  |  |
P2 |* |  |  |  |  |  |  |  |  |  |o |  |  |o |  |  |  |
P3 |  |  |  |  |* |  |  |  |  |  |o |  |  |o |  |  |  |
P4 |  |  |  |*X|  |  |  |  |  |*X|o |*X|  |  |  |  |  |
P5 |  |oX|  |  |  |  |  |  |  |  |o |  |  |  |  |  |  |
P6 |  |  |  |  |  |  |  |  |  |  |o |  |* |  |  |*X|  |
P7 |  |  |  |  |  |  |  |  |  |  |o |  |  |  |  |  |  |
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
f? |  | X| X| X|  |  |  |  |  | X|  | X|  |  |  | X|  |
```

Die gesuchte Schlussfolgerung 17 wurde gefunden. Der Aufbau des Testfalls führt zusätzlich zu anderen Schlussfolgerungen.

Literaturverzeichnis

[1] Beckett, D. (Hrsg.) ; Berners-Lee, T. (Hrsg.): *Turtle - Terse RDF Triple Language, W3C Team Submission.* http://www.w3.org/TeamSubmission/turtle/. Version: 2008

[2] Hitzler, P. (Hrsg.) ; Krötzsch, M. (Hrsg.) ; Parsia, B. (Hrsg.) ; Patel-Schneider, P. F. (Hrsg.) ; Rudolph, S. (Hrsg.): *OWL 2 Web Ontology Language: Primer, W3C Recommendation.* http://www.w3.org/TR/2009/REC-owl2-primer-20091027/, 2009

[3] Motik, B. (Hrsg.) ; Grau, B. C. (Hrsg.) ; Horrocks, I. (Hrsg.) ; Wu, Z. (Hrsg.) ; Fokoue, A. (Hrsg.) ; Lutz, C. (Hrsg.): *OWL 2 Web Ontology Language: Profiles, W3C Recommendation.* http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/, 2009

[4] Bao, J. (Hrsg.) ; Kendall, E. F. (Hrsg.) ; McGuinness, D. L. (Hrsg.) ; Patel-Schneider, P. F. (Hrsg.): *OWL 2 Web Ontology Language: Quick Reference Guide, W3C Recommendation.* http://www.w3.org/TR/2009/REC-owl2-quick-reference-20091027/, 2009

[5] Motik, B. (Hrsg.) ; Patel-Schneider, P. F. (Hrsg.) ; Parsia, B. (Hrsg.): *OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax, W3C Recommendation.* http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/, 2009

[6] *Curriculum Bachelor- und Masterstudium Wirtschaftsinformatik.* http://www.win.jku.at/documents/CurriculumWIN_2008-2010.pdf, 2010. – Der Studienplan ist seit 1. Oktober 2010 in Kraft.

[7] Baader, F. (Hrsg.) ; Calvanese, D. (Hrsg.) ; McGuinness, D. L. (Hrsg.) ; Nardi, D. (Hrsg.) ; Patel-Schneider, P. F. (Hrsg.): *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003 . – ISBN 0–521–78176–0

[8] Baader, F. ; Nutt, W. : Basic Description Logics. In: [7], S. 43–95

[9] Bechhofer, S. ; Harmelen, F. van ; Hendler, J. ; Horrocks, I. ; McGuinness, D. L. ; Patel-Schneider, P. F. ; Stein, L. A. ; Dean, M. (Hrsg.) ; Schreiber, G. (Hrsg.): *OWL Web Ontology Language Reference, W3C Recommendation.* http://www.w3.org/TR/2004/REC-owl-ref-20040210/, 2004

[10] Biron, P. V. ; Malhotra, A. : *XML Schema Teil 2: Datentypen: Deutsche Übersetzung, W3C Empfehlung.* http://www.edition-w3c.de/TR/2001/REC-xmlschema-2-20010502/, 2001

[11] Borgida, A. ; Brachman, R. J.: Conceptual Modeling with Description Logics. In: [7], S. 349–372

[12] Choi, N. ; Song, I.-Y. ; Han, H. : A Survey on Ontology Mapping. In: *SIGMOD Record* 35 (2006), Nr. 3, S. 34–41

[13] Euzenat, J. ; Valtchev, P. : Similarity-Based Ontology Alignment in OWL-Lite. In: Mántaras, R. L. (Hrsg.) ; Saitta, L. (Hrsg.): *ECAI*, IOS Press, 2004. – ISBN 1–58603–452–9, S. 333–337

[14] Fischer, C. : *Integration des intelligenten tutoriellen Systems eTutor in die Lernumgebung Moodle*, Johannes Kepler Universität Linz, Diplomarbeit, 2010

[15] Freeman, E. ; Freeman, E. ; Sierra, K. ; Bates, B. : *Head First Design Patterns.* O'Reilly, 2004

[16] Gosling, J. ; Joy, B. ; Steele, G. L. ; Bracha, G. : *The Java Language Specification.* Third Edition. Addison-Wesley http://java.sun.com/docs/books/jls/. – ISBN 978–0–321–24678–3

[17] Hofer, A. : *E-Exercises in Data & Knowledge Engineering*, Johannes Kepler Universität Linz, Diplomarbeit, 2005

[18] Horridge, M. ; Bechhofer, S. : The OWL API: A Java API for Working with OWL 2 Ontologies. In: Hoekstra, R. (Hrsg.) ; Patel-Schneider, P. F. (Hrsg.): *OWLED* Bd. 529, CEUR-WS.org, 2009 (CEUR Workshop Proceedings)

[19] Horridge, M. ; Drummond, N. ; Goodwin, J. ; Rector, A. ; Wang, H. H.: The Manchester OWL Syntax. In: *In Proc. of the 2006 OWL Experiences and Directions Workshop (OWL-ED2006)*, 2006

[20] Horridge, M. ; Patel-Schneider, P. F.: *OWL 2 Web Ontology Language: Manchester Syntax, W3C Working Group Note.* http://www.w3.org/TR/2009/NOTE-owl2-manchester-syntax-20091027/, 2009

[21] Knublauch, H. ; Fergerson, R. W. ; Noy, N. F. ; Musen, M. A.: The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In: McIlraith, S. A. (Hrsg.) ; Plexousakis, D. (Hrsg.) ; Harmelen, F. van (Hrsg.): *International Semantic Web Conference* Bd. 3298, Springer, 2004 (Lecture Notes in Computer Science). – ISBN 3–540–23798–4, S. 229–243

[22] Knublauch, H. ; Musen, M. A. ; Rector, A. L.: Editing Description Logic Ontologies with the Protégé OWL Plugin. In: Haarslev, V. (Hrsg.) ; Möller, R. (Hrsg.): *Description Logics* Bd. 104, CEUR-WS.org, 2004 (CEUR Workshop Proceedings)

[23] Kodaganallur, V. ; Weitz, R. R. ; Rosenthal, D. : Tools for Building Intelligent Tutoring Systems. In: *HICSS*, IEEE Computer Society, 2006. – ISBN 0–7695–2507–5

[24] Koenings, B. : *Konzeption und Entwicklung eines halbautomatisierten Systems zur Bewertung und Beurteilung von konzeptuellen Datenbankschemata im Rahmen des intelligenten tutoriellen Systems E-Tutor*, Johannes Kepler Universität Linz, Diplomarbeit, 2011

[25] Konev, B. ; Lutz, C. ; Walther, D. ; Wolter, F. : CEX and MEX: Logical Diff and Semantic Module Extraction in a Fragment of OWL. In: Clark, K. (Hrsg.) ; Patel-Schneider, P. F. (Hrsg.): *In Proceedings of the OWLED 2008 DC Workshop on OWL: Experiences and Directions*, 2008

[26] Lusti, M. : *Intelligente tutorielle Systeme*. Oldenbourg, 1992

[27] McGuinness, D. L. ; Harmelen, F. van: *OWL Web Ontology Language Overview: Deutsche Übersetzung, W3C Empfehlung*. `http://www.semaweb.org/dokumente/w3/TR/2004/REC-owl-features-20040210-DE.html`, 2004

[28] Motik, B. ; Shearer, R. ; Horrocks, I. : Hypertableau Reasoning for Description Logics. In: *Journal of Artificial Intelligence Research* 36 (2009), S. 165–228

[29] Noy, N. F. ; McGuinness, D. L.: *Ontology Development 101: A Guide to Creating Your First Ontology*. `http://protege.stanford.edu/publications/ontology_development/ontology101.pdf`

[30] Parsia, B. ; Sirin, E. ; Kalyanpur, A. : Debugging OWL ontologies. In: Ellis, A. (Hrsg.) ; Hagino, T. (Hrsg.): *WWW*, ACM, 2005. – ISBN 1–59593–046–9, S. 633–640

[31] Rector, A. L. ; Drummond, N. ; Horridge, M. ; Rogers, J. ; Knublauch, H. ; Stevens, R. ; Wang, H. ; Wroe, C. : OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns. In: Motta, E. (Hrsg.) ; Shadbolt, N. (Hrsg.) ; Stutt, A. (Hrsg.) ; Gibbins, N. (Hrsg.): *EKAW* Bd. 3257, Springer, 2004 (Lecture Notes in Computer Science). – ISBN 3–540–23340–7, S. 63–81

[32] Sattler, U. : Reasoning in Description Logics: Basics, Extensions, and Relatives. In: Antoniou, G. (Hrsg.) ; Assmann, U. (Hrsg.) ; Baroglio, C. (Hrsg.) ; Decker, S. (Hrsg.) ; Henze, N. (Hrsg.) ; Patranjan, P.-L. (Hrsg.) ; Tolksdorf, R. (Hrsg.): *Reasoning Web* Bd. 4636, Springer, 2007 (Lecture Notes in Computer Science). – ISBN 978–3–540–74613–3, S. 154–182

[33] Shah, H. ; Kumar, A. N.: A tutoring system for parameter passing in programming languages. In: Caspersen, M. E. (Hrsg.) ; Joyce, D. T. (Hrsg.) ; Goelman, D. (Hrsg.) ; Utting, I. (Hrsg.): *ITiCSE*, ACM, 2002. – ISBN 1–58113–499–1, S. 170–174

[34] Shvaiko, P. ; Euzenat, J. : Ten Challenges for Ontology Matching. In: Meersman, R. (Hrsg.) ; Tari, Z. (Hrsg.): *OTM Conferences (2)* Bd. 5332, Springer, 2008 (Lecture Notes in Computer Science). – ISBN 978–3–540–88872–7, S. 1164–1182

[35] Sirin, E. ; Parsia, B. ; Grau, B. C. ; Kalyanpur, A. ; Katz, Y. : Pellet: A Practical OWL-DL Reasoner. In: *J. Web Sem.* 5 (2007), Nr. 2, S. 51–53

[36] W3C OWL Working Group: *OWL 2 Web Ontology Language: Document Overview, W3C Recommendation.* `http://www.w3.org/TR/2009/REC-owl2-overview-20091027/`, 2009