UNIVERSITÄT LEIPZIG
Fakultät für Mathematik und Informatik
Institut für Informatik

Anwendung von Lernalgorithmen an Phänotyp-Ontologien einer Hausmaus mit Hilfe von DL-Learner

Diplomarbeit

Leipzig, 01.03.2012

vorgelegt von

Eduard Metzler
geb. am. 25.02.1983
Studiengang
Informatik

Betreuer:
Prof. Dr. Klaus-Peter Fähnrich
Dr. Jens Lehmann

Kurzfassung

Ontologie ist heute kein Fremdwort mehr und gehört bei vielen Wissenschaftlern zu ihrem Alltag. Es werden viele unterschiedliche Ontologien in nahezu allen Bereichen des Lebens entwickelt und verwaltet, so auch in Biologie und Medizin. Sie helfen dabei, einen Wissensbereich mit Hilfe einer standardisierenden Terminologie zu beschreiben. Aufgrund von vordefinierten und angewendeten Standards zur Wissensmodellierung bieten Ontologien eine hervorragende Grundlage zum Einsetzen von Werkzeugen aus dem Bereich des maschinellen Lernens. Bei dieser Arbeit wird *DL-Learner* Framework zum überwachten maschinellen Lernen eingesetzt. Zwei Ontologien zur Darstellung von phänotypischen Informationen dienen als Gegenstand der Untersuchung und Daten über gesunde und kranke Phänotypen werden als Beispiele verwendet. Dabei kommen verschiedene *Reasoner* und Lernalgorithmen zum Einsatz.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die rasant verlaufende Entwicklung im Bereich der Informatik beeinflusst alle *Life Sciences*, speziell die Biologie und die Medizin. Jeder Biowissenschaftler kommt heutzutage ohne Werkzeuge der Informatik nicht mehr aus und nutzt die vielfältigen Hardware- und Software-Entwicklungen. Bioinformatische Techniken wie Sequenzsuche, Sequenzvergleiche und Abfragen biologischer Datenbanken gehören zu den alltäglichen Aufgaben eines jeden Biowissenschaftlers.

Der Durchbruch des World Wide Web und somit des Internets bietet eine Möglichkeit Daten in riesigen Mengen zu vernetzen, öffentlich abzulegen und somit für alle Interessanten frei verfügbar zu machen. Dies erfordert eine neue Denkweise, und zwar muss man wissen, wo die erforderlichen Informationen in Form von Daten zu suchen sind bzw. welche Hilfsmittel zu einem konkreten Problem im Internet existieren. Das frei verfügbare Wissen ist schwer zu verwalten, wenn man keine Standards bzw. Einschränkungen beachtet, die dieses Wissen für die computergestützte Verarbeitung effektiv und effizient machen. Einen vielversprechenden Ansatz bietet *Semantic Web*, als eine Erweiterung des World Wide Web. Das semantische Web setzt sich als Ziel, die Bedeutung von Informationen für Computer interpretierbar zu machen und ermöglicht somit eine automatische Weiterverarbeitung. Es wird ein neues Konzept der Wissensrepräsentation eingeführt. Das Wissen wird mit Hilfe von definierten Standards (z.B.: RDF, OWL) und definierten Beziehungen modelliert. Dies ermöglicht logisches Schlussfolgern und somit einen Zugriff auf implizites Wissen.

Besonders nützlich erweist sich die Beachtung der Standards im Bereich der wissenschaftlichen Massendaten, also beim Umgang mit wissenschaftlichen Texten und Datensätzen. Als Folge dessen wurde 2001 die *Open Biological und Biomedical Ontology Foundry*[1] (kurz: OBO Foundry) gegründet. Die Hauptaufgabe der OBO Foundry liegt in der geeigneten Partitionierung der biologischen und biomedizinischen Domäne und in der Sicherstellung logischer Kohärenz zwischen Ontologien des Fachgebiets (vgl. [Mungall et al.,

[1] *http://www.obofoundry.org/*

2010]). Dabei werden verschiedene Ontologien entwickelt und verwaltet, unter anderem *Mammalian Phenotype Ontology* (kurz: MP Ontology), *Mouse adult gross anatomy* (kurz: MA) oder *Phenotype and Trait Ontology* (kurz: PATO).
Die MP Ontologie und die MP Logical Definitions Subq Ontologie stehen im Mittelpunkt dieser Arbeit und werden im Kapitel 3.2„Ontologien“ noch detaillierter beschrieben.

1.2 Ziel

Das Ziel dieser Arbeit ist, eine Konzeptbeschreibung zu entwickeln, anhand deren bei einem gegebenen Phänotyp einer Hausmaus bestimmt werden kann, ob sie Diabetes hat. Dazu werden kranke und gesunde Phänotypen als positive und negative Lernbeispiele einem Lernverfahren unterzogen. Daraus können Parallelen zu anderen Phänotypen, z.B. vom Menschen gezogen werden.

1.3 Überblick über Inhalt und Struktur der Arbeit

Kapitel 1 hat einen einleitenden Charakter und beschreibt Motivation und Ziel dieser Arbeit. Ein kurzer Überblick über die Struktur der Arbeit wird ebenfalls gegeben.
Kapitel 2 befasst sich mit den Grundlagen. Dabei werden die zentralen Begriffe erläutert und die Ontologiesprachen RDF und OWL eingeführt, die zur Modellierung komplexen Wissens dienen. Danach wird es eine kleine Einführung in die Beschreibungslogiken geben, da diese eine Voraussetzung für das implizite Wissensextraktion sind. Am Ende des Kapitels wird das eingesetzte Framework „DL-Learner“ zum maschinellen überwachten Lernen vorgestellt, sowie seine Komponenten.
Kapitel 3 gibt kurz und knapp einen Überblick über die Repräsentation von Phänotypen. Dazu werden die verwendeten Formate vorgestellt und mit Beispielen erläutert. Die verwendeten Ontologien (MP Ontologie und MP Logical Definitions Subq Ontologie) werden ebenfalls detailliert anhand von Beispielen beschrieben.
In Kapitel 4 werden die eingesetzten Komponenten erläutert. Dazu werden Reasoner kurz vorgestellt, sowie die Lernalgorithmen.

Kapitel 5 befasst sich mit der Vorverarbeitung der Daten. Dabei werden die vorgenommenen Änderungen an Daten erläutert.

In Kapitel 6 wird es einen Überblick über den eigentlichen Lernprozess geben. Ergebnisse werden dargestellt und zusammengefasst.

Kapitel 7 gibt einen Ausblick über die Lösungsansätze, sowie die Literaturhinweise dazu.

In Kapitel 8 werden die Arbeitsschritte bei dieser Arbeit in einer Zusammenfassung dargestellt.

2 Grundlagen

2.1 Maschinelles überwachtes Lernen

Es gibt viele verschiedene Ansichten darüber, was *Maschinelles Lernen* ist. Eine mögliche Charakterisierung des Begriffs ist die folgende: *„Die Forschung zu maschinellem Lernen beschäftigt sich mit der Entwicklung von Computer-Programmen, die in der Lage sind, durch die Benutzung von Eingabeinformationen neues Wissen zu konstruieren oder bereits vorhandenes Wissen zu verbessern.“* [Michalski und Kodratoff, 1990]. Demzufolge dienen maschinelle Lernsysteme zum Erwerb bzw. Generierung von Wissen. Dabei benutzen sie nicht nur die Eingabeinformationen, sondern auch schon vorhandenes Hintergrundwissen.

Überwachtes Lernen ist eine Lernmethode, bei der dem Lernsystem eine feste Menge von richtigen (positiven) und falschen (negativen) Beispielen vorgegeben wird und somit ein Lernverfahren gestartet. Diese ist die üblichste Lernsituation beim Lernen aus Beispielen. Die positiven Beispiele sorgen dafür, dass das abgeleitete Konzept allgemein genug ist, während die negativen Beispiele verhindern, dass das Konzept zu allgemein wird. Somit sollte die entwickelte Konzeptbeschreibung alle positiven Beispiele einschließen und alle negativen Beispiele ausschließen.

2.2 Ontologie

Der Begriff „Ontologie“ setzt sich aus zwei griechischen Wörtern on, „seiend“, und logos, „Lehre, Wort“, zusammen und ist eine Disziplin der Philosophie, die auch als Lehre vom Sein bezeichnet wird. In der Informatik tauchte der Begriff in den 1990er Jahren und bezeichnete formale Repräsentationssysteme. Ein Definitionsversuch stammt von Tom Gruber. Er bezeichnet Ontologie als eine „*explizite formale Spezifikation einer gemeinsamen Konzeptualisierung*“ [Gruber, 1993]. Daraus folgt, dass eine Ontologie einen „*Wissensbereich*

(knowledge domain) mit Hilfe einer standardisierenden Terminologie sowie Beziehungen und ggf. Ableitungsregeln zwischen den dort definierten Begriffen“ [Hesse, 2002] beschreibt. Kurz gefasst ist Ontologie eine Beschreibung eines Anwendungsgebiets mit dazugehörigen Konzepten und Zusammenhängen. Diese Beschreibung ist ein in einer Ontologiesprache (RDF(S) oder OWL) erstelltes Dokument, welches Wissen einer Anwendungsdomäne modelliert.

Ontologien bestehen aus Konzepten, Individuen, Eigenschaften, Relationen und Axiomen.

Konzepte bzw. Begriffe repräsentieren Objekte der realen Welt und stellen eine Zusammenfassung der Eigenschaften dar. Sie entsprechen den Klassen in der objektorientierten Programmierung (OOP). Ein Individuum bzw. eine Instanz bildet ein bestimmtes Objekt eines Konzepts ab und ist mit Objekten in der OOP vergleichbar. Eigenschaften und Relationen beschreiben die einzelnen Objekte und ihre Beziehungen untereinander. Axiome sind Aussagen innerhalb der Ontologie, die immer wahr sind.

2.3 Ontologiesprachen

Das *Resource Description Framework* (RDF) ist eine formale Sprache für die Darstellung von Informationen über Ressourcen im World Wide Web und ist ein W3C[2] Standard (vgl. [Manola und Miller, 2004]). Solche Informationen, auch als Metadaten bezeichnet, sind z.B. der Titel, der Autor oder auch das Datum der Erstellung einer Webseite. Durch die Verallgemeinerung des Konzepts „Web-Ressource“ kann man das RDF zum Repräsentieren von all möglichen „Dingen“ benutzen, die im Web identifiziert werden können.

RDF basiert auf der Idee der Identifizierung von Ressourcen mit sogenannten URIs (*Uniform Resource Identifiers*) und nutzt einfache Eigenschaften und Eigenschaftswerte für die Beschreibung. Dies ermöglicht RDF, einfache Aussagen über Ressourcen in Form von Graphen mit Knoten und gerichteten Kanten darzustellen. Aussagen in RDF werden als Triple modelliert, und zwar als Subjekt, Prädikat und Objekt. In der Abbildung 1 wird die Ressource *Eric Miller* als eine Person mit Namen, Mailbox und Titel beschrieben.

[2] World Wide Web Consortium, http://www.w3.org/

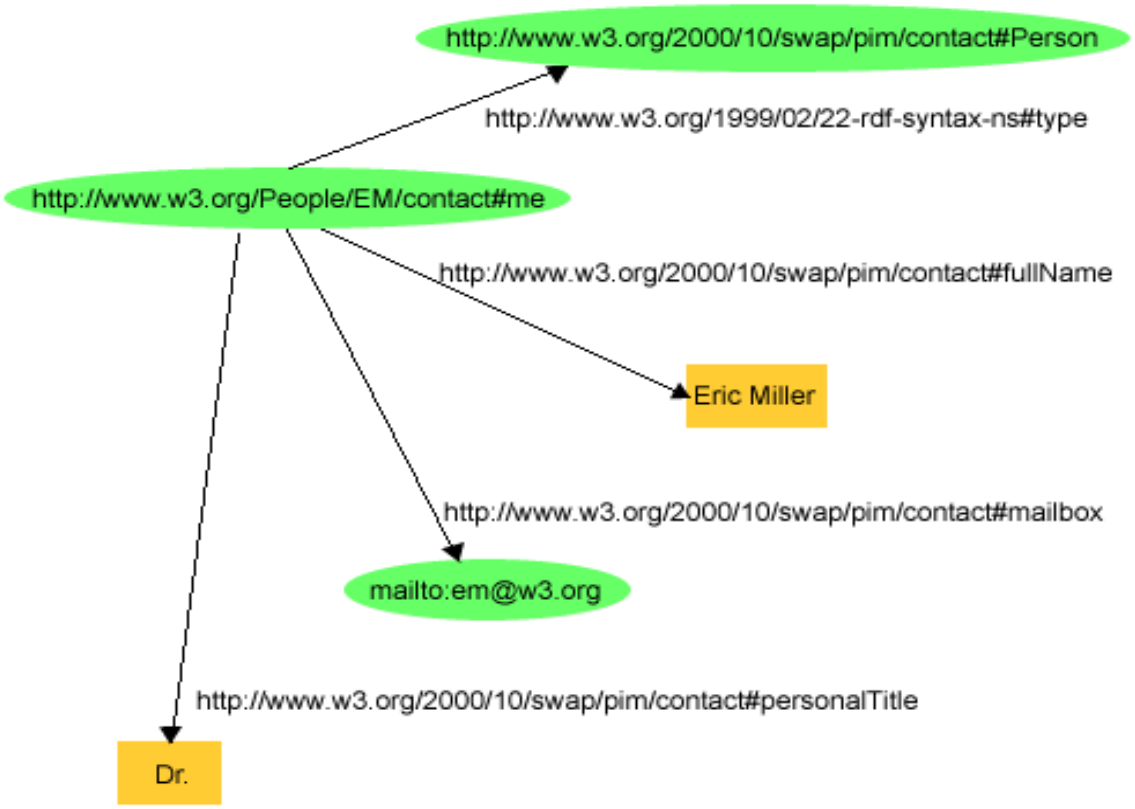

Abbildung 1: Ein RDF-Graph zur Beschreibung von „Eric Miller“ aus [Manola und Miller, 2004]

Zum Erfassen und Austauschen von Graphen bietet RDF eine XML-basierte Syntax, genannt RDF/XML.

```
<?xml version="1.0"?>
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">
        <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
                <contact:fullName>Eric Miller</contact:fullName>
                <contact:mailbox rdf:resource="mailto:em@w3.org"/>
                <contact:personalTitle>Dr.</contact:personalTitle>
        </contact:Person>
</rdf:RDF>
```

Abbildung 2: RDF/XML Beschreibung von „Eric Miller“ aus [Manola und Miller, 2004]

RDF Schema macht es möglich, sogenanntes terminologisches Wissen bzw. Schemawissen über die in einem Vokabular verwendeten Begriffe zu spezifizieren (vgl. [Hitzer et al., 2008]). RDF(S) ist geeignet zur Modellierung einfacher Ontologien und erlaubt die Ableitung des

impliziten Wissens. Jedoch für die Darstellung komplexerer Zusammenhänge ist RDF(S) nicht ausreichend und besitzt nur beschränkte Ausdruckmittel.

Die *Web Ontology Language* (OWL) ist eine Spezifikation des W3C seit Februar 2004 und ist ein wesentlicher Bestandteil von Semantic Web (vgl. [McGuinness und van Harmelen, 2004]). OWL wurde entwickelt, um die Erstellung und die Verteilung von Ontologien anhand einer Beschreibungssprache zu ermöglichen. OWL ist eine Erweiterung von XML, RDF und RDF Schema und bietet eine bessere maschinelle Interpretation von Daten durch Anwendungen. OWL basiert auf der Prädikatenlogik erster Stufe. Durch logisches Schlussfolgern wird der Zugriff auf implizites Wissen ermöglicht.

OWL bietet drei verschiedene Sprachebenen mit den unterschiedlichen Ausdrucksstärken an vgl. [Hitzer et al., 2008]:

- OWL Lite:
    - Teilsprache von OWL DL und OWL Full,
    - weniger ausdrucksstark,
    - entscheidbar.
- OWL DL:
    - Teilsprache von OWL Full, enthält OWL Lite,
    - ausdrucksstärker als OWL Lite,
    - entscheidbar.
- OWL Full:
    - enthält OWL DL und OWL Lite,
    - sehr ausdruckstark,
    - unentscheidbar.

OWL DL ist die gängige Teilsprache und wird von aktuellen Softwarewerkzeugen fast vollständig unterstützt. Sie bietet den Benutzern die maximale Ausdrucksstärke, wobei die Entscheidbarkeit beibehalten wird. D.h. die Frage, ob eine Aussage gefolgert werden kann, ist mit einem Algorithmus beantwortbar, der in endlicher Zeit stets terminiert. Alle OWL Sprachkonstrukte werden von OWL DL eingeschlossen, wobei diese unter bestimmten Bedingungen verwendet werden. Der Name OWL DL bezieht sich auf die Beschreibungslogik (s. 2.4 Beschreibungslogiken (DL)).

2.4 Beschreibungslogiken (DL)

Beschreibungslogiken (engl. *Description Logics*) bezeichnen eine Familie von Wissensrepräsentationssprachen, die aus semantischen Netzwerken und Framelogiken hervorgegangen ist (vgl. [Baader et al., 2003]). Sie lassen sich als Fragmente der Prädikatenlogik erster Stufe auffassen. Beschreibungslogiken sind nicht so ausdrucksstark wie die Prädikatenlogik, bieten dafür aber entscheidbare Inferenzprobleme und eine benutzerfreundliche, variablenfreie Syntax an. OWL DL entspricht der ausdrucksstarken und entscheidbaren Beschreibungslogik *SHOIN(D).* Diese wird noch nach der Einführung in die Beschreibungslogik *ALC* kurz erläutert.

Beschreibungslogiken repräsentieren Wissen in Form von Konzepten (Klassen), Rollen und Objekten (Individuen), die miteinander in Beziehung gesetzt werden können. Ein Konzept beschreibt eine Menge von Objekten mit gemeinsamen Eigenschaften und entspricht einstelligen Prädikaten in der Prädikatenlogik erster Stufe. Objekte sind konkrete Ausprägungen von Konzepten und entsprechen Konstanten. Rollen beschreiben Beziehungen zwischen Objekten und sind zweistellige Prädikate in der Prädikatenlogik erster Stufe.

Wissen über eine Domäne wird in Beschreibungslogik in einer Wissensbasis abgespeichert. Diese hat typischerweise zwei Komponenten – *TBox* und *ABox*. Die TBox („*terminology*“) enthält terminologisches Schemawissen (in Form von Konzepten und Relationen), indem sie das Vokabular in einem Anwendungsgebiet definiert. Die ABox („*assertions*“) enthält assertionales Instanzwissen (in Form von Objekten), d.h. Annahmen über einzelne Objekte ausgedrückt mit Hilfe des Vokabulars. Eine Terminologie mit Konzepten über Familie könnte wie in der Tabelle 1 beginnen.

TBOX	ABOX
FRAU ≡ PERSON ⊓ WEIBLICH MUTTER ≡ FRAU ⊓ ∃ hatKind. ⊤ …	FRAU(Anna) hatKind(Anna, Peter) …

Tabelle 1 : Unterteilung des Wissens in TBox und ABox

Eine Frau kann als eine weibliche Person definiert werden.

Eine Mutter kann als eine Frau mit Kind definiert werden.

Anna ist eine Frau und hat ein Kind Peter.

Auf Beschreibungslogik basierende Systeme enthalten neben der Terminologie und den Annahmen noch verschiedene Schlussfolgerungsmechanismen (*Inference System*s). Typische Inferenzaufgaben für die Terminologie sind (vgl. [Baader et al., 2003]):

- Erfüllbarkeit von Konzepten: Ist das Konzept erfüllbar?
  Gibt es ein Modell für das Konzept?
- Subsumption von Konzepten: Ist Konzept A allgemeiner als Konzept B?

Wichtige Probleme für die ABox sind:

- Konsistenz: Ist die Menge von Annahmen konsistent?
  Gibt es ein Modell zu den Annahmen?
- Instanzzugehörigkeit: Ist ein Objekt a eine Instanz von Konzept A?
- Retrieval-Problem: Sind alle Objekte zu einem Konzept gefunden?

Die Struktur der auf Beschreibungslogiken basierenden Systeme sieht folgenderweise aus (vgl. [Horrocks]):

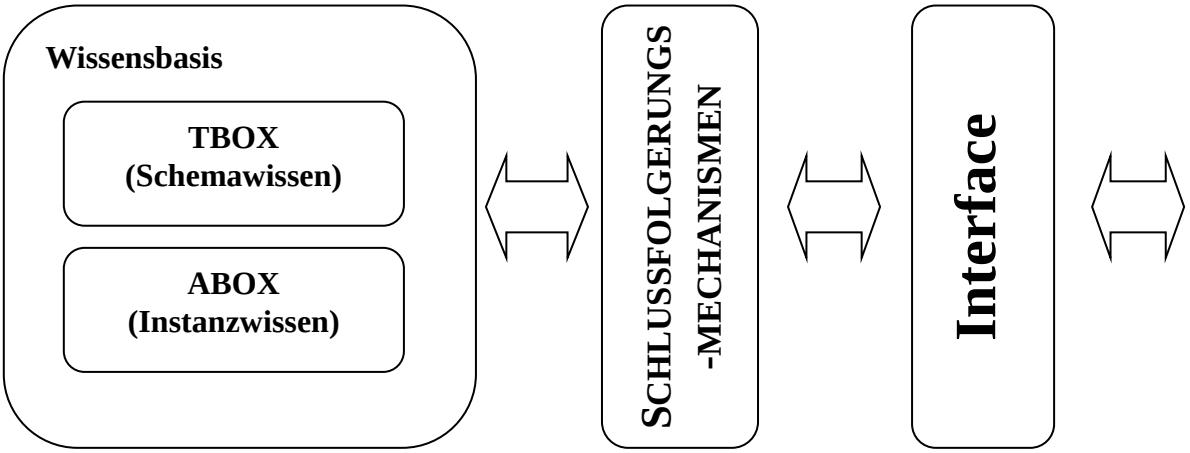

Abbildung 3: Beschreibungslogik Systemarchitektur

Im Folgenden wird die Beschreibungslogik *ALC* eingeführt, an der die Grundlagen für Beschreibungslogiken erläutert werden. *ALC* steht für das englische *Attributive Language with Complement* und erlaubt komplexe Konzeptbeschreibungen aus einfachen mittels verschiedenen Konzeptkonstruktoren zu bilden.

Syntax von ALC-Konzepten

Es sei N_c eine Menge von Konzeptnamen (Konzeptbezeichner) und N_r eine Menge von Rollennamen. Beide Mengen sind disjunkt. Die Elemente von N_c werden als atomare

Konzepte bezeichnet (vgl. [Lehmann und Hitzler, 2010]).

Die Menge von *ALC* Konzepten ist induktiv wie folgt definiert:

1. Jedes atomare Konzept ist ein *ALC*-Konzept.
2. Falls C und D *ALC*-Konzepte sind und r $\in$ N_r eine Rolle, dann sind folgende Ausdrücke *ALC*-Konzepte:
   - $\top$ (universelles Konzept), $\bot$ (Bottom Konzept)
   - $C \sqcup D$ (Disjunktion), $C \sqcap D$ (Konjunktion), $\neg C$ (Negation)
   - $\forall r.C$ (universelle Restriktion), $\exists r.C$ (existentielle Restriktion)

Beispiele von komplexen Konzepten sind in der Tabelle 1 zu sehen.

Semantik von ALC-Konzepten

Um die formale Semantik von ALC-Konzepten zu definieren, betrachtet man *Interpretationen I*, die aus einer nichtleeren Menge Δ^I (Domäne von der Interpretation) und einer Interpretationsfunktion bestehen, welche

- jedem atomaren Konzept *A* eine Menge $A^I \subseteq \Delta^I$ und
- jeder atomarer Rolle *r* eine binäre Relation $r^I \subseteq \Delta^I \times \Delta^I$ zuweist.

Interpretationsfunktion wird wie folgt auf die Konzeptbeschreibungen erweitert:

- $\top^I = \Delta^I$
- $\bot^I = \emptyset$
- $(\neg A)^I = \Delta^I \setminus A^I$
- $(C \sqcap D)^I = C^I \cap D^I$
- $(C \sqcup D)^I = C^I \cup D^I$
- $(\forall r.C)^I = \{a \in \Delta^I \mid \forall b.(a, b) \in r^I \rightarrow b \in C^I\}$
- $(\exists r.\top)^I = \{a \in \Delta^I \mid \exists b.(a, b) \in r^I \text{ und } b \in C^I\}$

Die Tabelle 2 zeigt eine Zusammenfassung von Syntax und Semantik von *ALC*-Konzepten

construct	syntax	semantics
atomic concepts	A	$A^I \subseteq \Delta^I$
role	r	$r^I \subseteq \Delta^I \times \Delta^I$
top concept	$\top$	Δ^I
bottom concept	$\bot$	Ø
conjunction	$C \sqcap D$	$(C \sqcap D)^I = C^I \cap D^I$
disjunction	$C \sqcup D$	$(C \sqcup D)^I = C^I \cup D^I$
negation	$\neg C$	$(\neg C)^I = \Delta^I \setminus C^I$
exists restriction	$\exists r.C$	$(\exists r.C)^I = \{a \mid \exists b.(a, b) \in r^I \text{ and } b \in C^I\}$
value restriction	$\forall r.C$	$(\forall r.C)^I = \{a \mid \forall b.(a, b) \in r^I \text{ implies } b \in C^I\}$

Tabelle 2: Zusammenfassung von Syntax und Semantik von ALC-Kontepten aus [Lehmann und Hitzler, 2010]

Bezeichnungen bei Beschreibungslogiken drücken ihre Ausdrucksstärke aus. Dabei je ausdrucksstärker die Logik ist, desto schwerer verläuft das Reasoning. Das Reasoning und die einzelnen Reasoner werden im Kapitel 4.1 behandelt. Die Beschreibungslogik, die OWL DL abdeckt, ist *SHOIN(D)*. Jeder Buchstabe steht für einen Konzeptkonstruktor:

- *S* steht für *ALC* mit Rollentransitivität,
- *H* für Unterrollenbeziehung,
- *O* für abgeschlossene Klassen,
- *I* für inverse Rollen,
- *N* für Zahlenrestriktionen,
- *D* für Datentypen.

Somit besitzt *SHOIN(D)* nicht nur alle Sprachkonstrukte von *ALC,* sondern fügt weitere noch hinzu. Die kurze Einführung in ALC-Konzepte genügt für eine theoretische Grundlage in Bereich der Beschreibungslogiken und für tiefere Einblicke wird auf [Lehmann, 2010] und [Baader et al., 2003] verwiesen.

CEL Reasoner hat *EL* als zugrundeliegende Sprache, so dass diese hier kurz erwähnt wird. Die Beschreibungslogik *EL* ist eine Teilsprache von *ALC*. Es sind nur die Konstruktoren $\top$, Konjunktion und Existenzrestriktion erlaubt. Durch die geringe Komplexität der Sprache *EL* sollte das Subsumptionsproblem in polynomieller Zeit lösbar sein. In der Tabelle 1 wird die Syntax und Semantik von EL zusammengefasst dargestellt.

Konstruktor	Syntax	Semantik
Top	$\top$	Δ^I
Konzeptbezeichnung	A	A^I
Konjunktion	$C \sqcap D$	$C^I \cap D^I$
Existenzrestriktion	$\exists r.C$	$\{x \in \Delta^I \mid \exists y \in C^I \text{ mit } (x,y) \in r^I\}$

Tabelle 3: EL Syntax und Semantik

2.5 DL-Learner Framework

DL-Learner [Lehmann, 2011] ist ein Open-Source-Framework zum überwachten maschinellen Lernen in OWL und Beschreibungslogik. Das Framework wird zum Lernen von Konzepten aus Beispielen eingesetzt. Äquivalent können Klassen in OWL-Ontologien aus Objekten gelernt werden. DL-Learner erweitert Induktive logische Programmierung[3] (kurz: ILP) zu Beschreibungslogik und Semantic Web.
Bei den meisten Szenarien wird von einer Wissensbasis in OWL/DLs und positiven sowie negativen Beispielen ausgegangen. Das Ziel ist dabei eine OWL Konzeptbeschreibung (*Class Expression*) zu finden, die alle bzw. viele positive Beispiele als Instanzen beinhaltet und dabei keine bzw. wenige negative Beispiele als Instanzen. Die entwickelte Konzeptbeschreibung sollte in der Lage sein, unbekannte Objekte, die nicht zu den Beispielen gehören, richtig zu klassifizieren.

Architektur

Das DL-Learner Framework stellt Basisfunktionalität zur Verfügung. Diese leistet Unterstützung für verschiedene Formate für Wissensbasen und bietet bestimmte Lernalgorithmen und Reasoner Interfaces an. DL-Learner baut auf einem komponentenbasierten Modell auf. Dies macht das Framework sehr flexibel in Bezug auf Integration neuer Komponenten. Es gibt vier Arten von Komponenten. Für jede Art stehen mehrere implementierte Komponenten zur Verfügung, die sich alle einzeln konfigurieren

[3] Bereich des maschinellen Lernens, in dem Verfahren zur automatischen Erstellung von logischen Programmen untersucht werden.

lassen. In der Abbildung 4 ist die Architektur von DL-Learner zu sehen.

Arten von Komponenten:

a) Wissensbasis - Komponente

   DL-Learner unterstützt unter anderem OWL-Dateien in verschiedenen Formaten. RDF/XML und N-Triples sind typische Formate für OWL-Dateien.

b) Lernalgorithmus - Komponente

   Die implementierten Algorithmen variieren in ihrer Komplexität. Der eingesetzte Algorithmus für die Suche nach dem Klassifikator wird später noch einmal erwähnt und detaillierter beschrieben.

c) Lernproblem - Komponente

   Neben der schon erwähnten Lernsituation mit positiven und negativen Beispielen kann es vorkommen, dass es nur positive Beispiele zur Verfügung stehen. Dies kann in der Komponente explizit angegeben werden.

d) Reasoner – Komponente

   DL-Learner bietet eine OWL API Reasoner Interface, die verschiedene implementierte Reasoner zur Verfügung stellt. Die benutzten Reasoner werden später noch genauer erläutert.

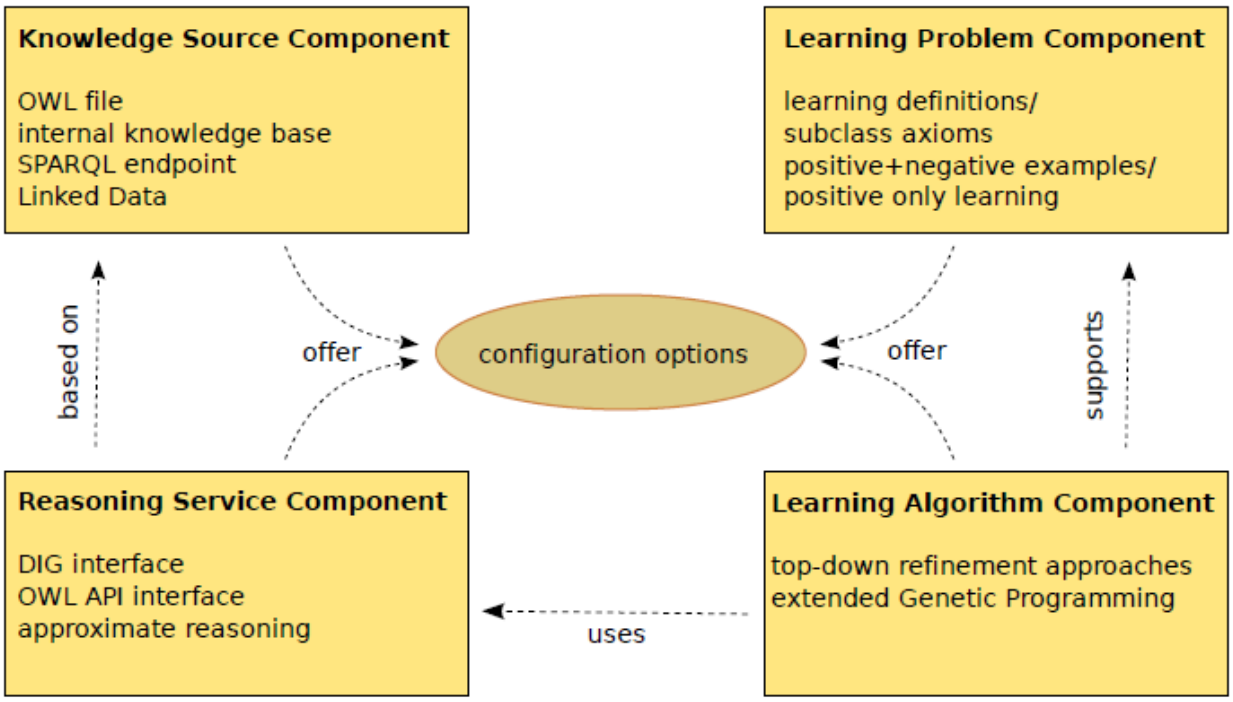

Abbildung 4: Architektur von DL-Learner [Lehmann, 2011]

3 Repräsentation von Phänotypen

3.1 Formate

3.1.1 OBO Flat File Format

Das OBO Flat File Format ist eine Sprache zur Repräsentation von Ontologien. Das Format wurde von der OBO Foundry spezifiziert und modelliert eine Untermenge von Konzepten der OWL Beschreibungslogik-Sprache, mit mehreren Erweiterungen für Metadaten-Modellierung. Mit dem Format wurden folgende Ziele realisiert: menschliche Lesbarkeit, einfache Analyse, Erweiterbarkeit und minimale Redundanz. Das OBO Flat File Format wird vom Ontologie-Editor „OBO-Edit“, als auch von allen, von der OBO Foundry verwalteten, Ontologien eingesetzt, sodass auch die Mammalian Phenotype Ontologie und die MP Equivalence Axioms Subq Ontologie in diesem Format vorliegen.
Da für das Lernen in DL-Learner Framework nur das OWL-Format eingesetzt werden kann, bleibt es hier bei der Beschreibung und stattdessen wird auf [Day, 2006] und [Horrocks, 2007] verwiesen. Zur besseren Lesbarkeit werden jedoch einige Beispiele in OBO-Format angegeben.

3.1.2 OWL/RDF Format

OWL wurde schon im Kapitel 2.3 als eine Beschreibungssprache zur Erstellung von Ontologien eingeführt. Die verwendete Syntax ist die OWL/RDF-Syntax, die auch als das offizielle Austauschformat gilt. DL-Learner bietet Unterstützung von verschiedene OWL-Formate, sodass die MP Ontologie und die MP Equivalence Axioms Subq Ontologie in OWL/RDF Syntax zum Lernen eingesetzt werden. Hier werden nur kurz einige *Tags* vorgestellt, die bei den Angaben von Beispielen relevant sein werden. Dazu wird die Abbildung 5 als ein Beispiel für eine OWL-Klasse auseinandergenommen. Eine OWL-Klasse entspricht einer Menge von Individuen und wird mit *<owl:Class>*-Tag eingeführt und mit </owl:Class>-Tag abgeschlossen. Das einleitende Tag besitzt ein Attribut *rdf:about* mit dem

die beschriebene Ressource angegeben wird. <rdfs:label> steht für die Bezeichnung der Klasse und definiert mit Attribut *rdf:datatype=http://www.w3.org/2001/XMLSchema#string* den Datentyp der Angabe. Das Tag *<rdfs:subClassOf>* zeigt eine Subsumptionsbeziehung (auch: Subklassen-) und ist für das Lernen von Bedeutung. Andere Tags wie *<oboInOwl:id>*, *<oboInOwl:hasExactSynonym>* oder *<oboInOwl:hasOBONamespace* versehen die Klasse mit zusätzlichen Informationen, werden aber bei der späteren Vorverarbeitung der Daten entfernt. Zur besseren Darstellung wird eine Abbreviatur rdf:datatype=string für rdf:datatype=http://www.w3.org/2001/XMLSchema#string verwendet.

```
<owl:Class rdf:about="http://purl.obolibrary.org/obo/MP_0000813">
  <rdfs:label rdf:datatype=string>
        abnormal hippocampus layer morphology
  </rdfs:label>
  <rdfs:subClassOf
        rdf:resource="http://purl.obolibrary.org/obo/MP_0000807"/>
  <oboInOwl:id rdf:datatype=string>
        MP:0000813
  </oboInOwl:id>
  <oboInOwl:hasOBONamespace rdf:datatype=string>
        MPheno.ontology
  </oboInOwl:hasOBONamespace>
  <oboInOwl:hasExactSynonym rdf:datatype=string>
        abnormal hippocampal laminar structure
  </oboInOwl:hasExactSynonym>
  <oboInOwl:hasExactSynonym rdf:datatype=string>
        abnormal laminar structure of hippocampus
  </oboInOwl:hasExactSynonym>
  <obo:IAO_0000115 rdf:datatype=string>
        any structural anomaly of the layers of the laminar structure of the hippocampus
  </obo:IAO_0000115>
</owl:Class>
```

Abbildung 5: OWL-Beschreibung der Klasse „MP:0000813“

```
id: MP:0000813
name: abnormal hippocampus layer morphology
def: "any structural anomaly of the layers of the laminar structure of the hippocampus"
synonym: "abnormal hippocampal laminar structure" EXACT []
synonym: "abnormal laminar structure of hippocampus" EXACT []
is_a: MP:0000807 ! abnormal hippocampus morphology
```

Abbildung 6: OBO-Beschreibung der Klasse „MP:0000813“

In der MP Equivalence Axioms Subq Ontologie treten noch weitere Tags auf. Diese werden in der Tabelle 4 kurz erklärt:

<owl:equivalentClass>	Äquivalenzbeziehung
<owl:Restriction>	Spezielle Art der Klassenbeschreibung; beschreibt eine anonyme Klasse von Individuen, die dieser *Restriction* genügt
<owl:onProperty>	Eigenschaften von der Klasse
<owl:someValuesFrom>	Existenzquantifizierung
<owl:intersectionOf>	AND-Beziehung(Durchschnitt)
<rdf:Description>	Angabe der Ressource

Tabelle 4: zusätzliche Tags aus Equivalence Axioms Subq Ontologie

3.2 Ontologien

3.2.1 Mammalian Phenotype Ontology

Die Mammalian Phenotype (MP) Ontologie dient als Werkzeug zum Annotieren, Analysieren und Vergleichen von phänotypischen Informationen.

Der Ausdruck „*Phänotyp*“ (auch „Erscheinungsbild“) stammt aus der Genetik und steht für die Gesamtheit der morphologischen, physiologischen und psychologischer Merkmale, die ein Organismus aufgrund des Zusammenspiels von Erbanlagen und Umwelteinflüssen im Laufe seiner Entwicklung ausbildet. In [Dugas und Schmidt, 2003] wird unter Phänotyp folgendes definiert: „*Beim Phänotyp eines Organismus […] handelt sich um seine tatsächlichen körperlichen Merkmale wie Größe, Gewicht und Haarfarbe“*. Der Genotyp (auch „Erbbild“) eines Organismus dagegen bezeichnet seine exakte genetische Ausstattung, den individuellen Satz von Genen.

Die MP-Ontologie wird von Cynthia L. Smith, Susan M. Bello, Carroll W. Goldsmith und Janan T. Eppig, als Teil von Mouse Genome Database (MGD) Project und Mouse Genome Informatics (MGI) entwickelt und steht in OBO- und in OWL-Format zur Verfügung. Beide

Formate wurden im Kapitel 3.1 genauer betrachtet. MP Ontologie ist frei verfügbar[4].

Die MP-Ontologie enthält vorab definierte Phänotyp-Terme, Definitionen und Synonyme. Diese können zur Beschreibung von Phänotypen benutzt werden. So wird eine Definition durch einen Bezeichner: *http://purl.obolibrary.org/obo/IAO_0000115* gekennzeichnet, welcher aus der *Information Artifact Ontology* (IAO) stammt. Diese findet eine breite Verbreitung unter Ontologien zur Beschreibung von Entitäten. In den Abbildung 7 und Abbildung 8 sind Annotationen zu Definition und Synonym zu sehen.

```
<owl:AnnotationProperty rdf:about="http://purl.obolibrary.org/obo/IAO_0000115">
  <rdfs:label rdf:datatype=http://www.w3.org/2001/XMLSchem#string>
    definition
  </rdfs:label>
</owl:AnnotationProperty>
```

Abbildung 7: Annotation von Definition

```
<owl:AnnotationProperty rdf:about="http://www.geneontology.org/formats/
oboInOwl#hasExactSynonym">
  <rdfs:label rdf:datatype="http:www.w3.org/2001/XMLSchem#string">
        has_exact_synonym
  </rdfs:label>
</owl:AnnotationProperty>
```

Abbildung 8: Annotation von Synonym

Die MP-Ontologie wird ein als ein gerichteter, azyklischer Graph repräsentiert. Phänotyp-Begriffe werden in die wichtigen biologischen Systeme wie das Nervensystem und das Atmungssystem organisiert. In der Abbildung 9 wird dies graphisch dargestellt.

[4] http://obofoundry.org/cgi-bin/detail.cgi?id=mammalian_phenotype

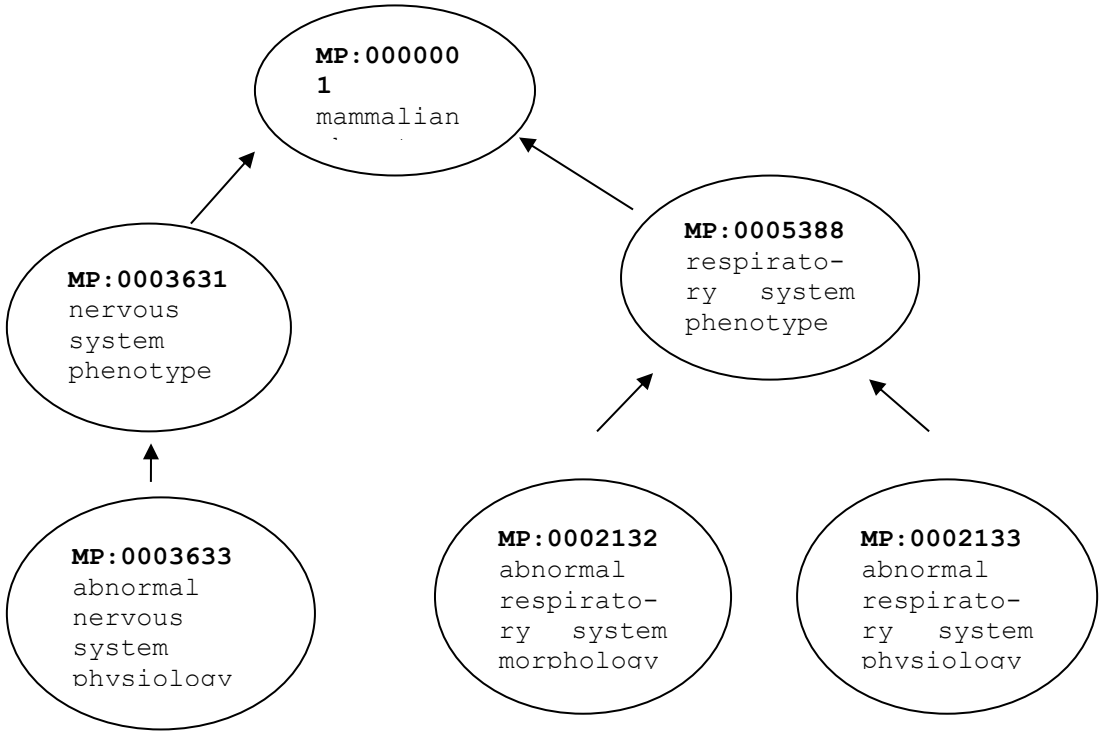

Abbildung 9: Darstellung von Phänotyp-Konzepten

Verhalten (MP:0004924, *abnormal behavior*) und Letalität (MP:0002080, *prenatal lethality*) werden ebenfalls repräsentiert. Aus der Abbildung 9 lässt sich erkennen, dass jedes Konzept lediglich über eine *subClassOf*-Beziehung (in OBO-Format: is_a) mit seinen Oberkonzepten verbunden ist. Somit handelt sich bei MP-Ontologie um eine einfache Taxonomie. Die Mammalian Phenotype Ontologie befindet sich in der ständigen Entwicklung und neue Einträge stehen unter Mouse Genome Informatics[5] und OBO Foundry[6] zur Verfügung

3.2.2 MP Equivalence Axioms Subq Ontology

Die Ontologie enthält äquivalente Axiome zu MP-Ontologie und hat folgenden Aufbau wie in der Abbildung 10. Bei der Beschreibung von Klassen wurden andere Ontologien einbezogen. Diese sind in der Tabelle 5 aufgelistet. Auf die Beschreibung jeder verwendeten Ontologie wird hier verzichtet und stattdessen werden drei Ontologien (BFO, CHEBI und PATO) kurz

[5] ftp://ftp.informatics.jax.org/pub/reports/index.html#pheno
[6] http://obofoundry.org/

mit Beispielen vorgestellt.

BFO	Basic Formal Ontology
ChEBI	Chemical Entities of Biological Interest
CL	Cell Ontology
FMA	Foundational Model of Anatomy
GO	Gene Ontology
MA	Mouse adult gross anatomy
MPATH	Mouse pathology
NBO	Neuro Behavior Ontology
PATO	Phenotype and Trait Ontology
PR	Protein Ontology
UBERON	Uber anatomy ontology

Tabelle 5: Liste der Ontologien

```
<owl:Class rdf:about="http://purl.obolibrary.org/obo/MP_0000813">
 <owl:equivalentClass>
   <owl:Restriction>
     <owl:onProperty
rdf:resource="http://purl.obolibrary.org/obo/BFO_0000051"/>
     <owl:someValuesFrom>
       <owl:Class>
         <owl:intersectionOf rdf:parseType="Collection">
            <rdf:Description
               rdf:about="http://purl.obolibrary.org/obo/PATO_0000051"/>
            <owl:Restriction>
               <owl:onProperty

rdf:resource="http://purl.obolibrary.org/obo/BFO_0000052"/>
               <owl:someValuesFrom

rdf:resource="http://purl.obolibrary.org/obo/MA_0002427"/>
            </owl:Restriction>
            <owl:Restriction>
               <owl:onProperty

rdf:resource="http://purl.obolibrary.org/obo/RO_0002180"/>
               <owl:someValuesFrom

rdf:resource="http://purl.obolibrary.org/obo/PATO_0000460"/>
            </owl:Restriction>
          </owl:intersectionOf>
       </owl:Class>
     </owl:someValuesFrom>
   </owl:Restriction>
  </owl:equivalentClass>
  <oboInOwl:id rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      MP:0000813
```

```
  </oboInOwl:id>
  <oboInOwl:hasNamespace
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      mammalian_phenotype_xp
  </oboInOwl:hasOBONamespace>
</owl:Class>
```

Abbildung 10: Äquivalente Beschreibung der Klasse MP_0000813 in OWL

```
[Term]
id: MP:0000813 ! abnormal hippocampus layer morphology
intersection_of: PATO:0000051 ! morphology
intersection_of: qualifier PATO:0000460 ! abnormal
intersection_of: inheres_in MA:0002427 ! hippocampus layer
```

Abbildung 11: Äquivalente Beschreibung der Klasse MP_0000813 in OBO

Basic Formal Ontology[7] (BFO) wurde als eine „obere Ebene"-Ontologie entwickelt. Sie dient zur Unterstützung von Domain-Ontologien in der wissenschaftlichen Forschung. BFO-Ontologie enthält keine physikalischen, chemischen, biologischen oder andere Terme, die in spezielle fachliche Domänen fallen würden.

```
<owl:onProperty rdf:resource="http://purl.obolibrary.org/obo/BFO_0000051"/>
```

Abbildung 12: "has_part"-Eigenschaft

Eine weitere verwendete Ontologie, mit der sich die komplexe Klassifizierung chemischer Verbindungen anhand einer Baumstruktur veranschaulichen lässt, ist die ChEBI-Ontologie[8]. ChEBI steht für *Chemical Entities of Biological Interest* und bietet Nomenklatur, Terminologie und Symbolik, die internationalem Standard entsprechen.

```
<owl:Class rdf:about="http://purl.obolibrary.org/obo/CHEBI_15035"/>
```

Abbildung 13: Chemisches Element "retinal"

Phenotype and Trait Ontology (PATO) ist die einzige Ontologie von Qualitäten zur Beschreibung von Phänotypen. Mit dem Zweck der Definition phänotypischer Charakteristika beschreibt PATO Qualitäten (auch: Eigenschaften) und deren Beziehungen untereinander. Eine Qualität bezeichnet in PATO Attribute, wie *color* (PATO_0000014) oder *concentration*

[7] http://www.ifomis.org/bfo
[8] http://www.ebi.ac.uk/chebi/init.do

of (PATO_0000033) und deren korrespondierende Werte, wie *brown* (PATO_0000952) oder *increased concentration* (PATO_0001162).

Die Klasse *MP_0000813* aus der Abbildung 10 wird nun beschrieben als Durchschnitt (*intersectionOf:*) von:
morphology + inheres in at all times(hippocampus layer) + has component(abnormal). Daraus ergibt sich auch die Bezeichnung für die Klasse als *abnormal hippocampus layer morphology.*
PATO_0000051 steht für *morphology*
BFO_0000052 steht für *inheres in at all times*
MA_0002427 steht für *hippocampus layer*
RO_0002180 steht für *has component*
PATO_0000460 steht für *abnormal*

4 Eingesetzte Komponenten

4.1 Reasoning und Reasoner

Die Zielstellungen beim *Reasoning* sind das „Berechnen“ von Subsumptionen und daraus folgende Aufstellung der Klassifikation, sowie das Feststellen von Inkonsistenzen. Daraus folgt, dass das Reasoning sich auf die Lösung des Subsumptionsproblems zurückführen lässt. Wobei je ausdrucksstärker die Logik ist, desto schwerer verläuft das Reasoning. Somit für besonders ausdrucksstarke Logiken kann es zu einem Problem mit exponentiellem Zeitaufwand werden oder ganz unentscheidbar sein.
Ein Reasoner ist das Schlüsselkomponent für das Arbeiten mit OWL Ontologien. In der Tat sollten alle Abfragen einer OWL Ontologie mit einem Reasoner durchgeführt werden. Dies liegt daran, dass das Wissen in einer Ontologie möglicherweise nicht explizit vorliegt und ein logischer „Denker“ erforderlich ist, um implizites Wissen abzuleiten und somit korrekte Ergebnisse bei Anfragen zu liefern. Die OWL API enthält verschiedene Interfaces für OWL Reasoner. Folgende Reasoner-Implementierungen sind vorhanden und werden vom DL-Learner unterstützt:

- FaCT++.
- HermiT
- Pellet

Mehrere Reasoner, die OWL als Sprache benutzen, stehen zur Verfügung. Aufgrund der Größe der verwendeten OWL Ontologien arbeiten aber nicht alle effizient. Unter Effizienz ist dabei die Laufzeit zu verstehen. Der Vergleich in [Shearer et al.] zeigt, dass der HermiT OWL Reasoner einige Vorteile gegenüber von Pellet und FaCT++ aufweist und somit für diese Arbeit relevant sein wird. HermiT[9] 1.3.6 stellt dabei die derzeit aktuellste Version dar. Die Version ist mit Java 1.5 kompatibel, verwendet die OWL API und ist ebenfalls Open-Source. Wird dem Reasoner ein OWL File zur Interpretierung übergeben, ist er in der Lage, die Ontologie auf ihre Konsistenz zu prüfen und Klassenbeziehungen zu erkennen. Laut Entwickler ist der HermiT Reasoner aufgrund der verwendeten Algorithmen imstande, eine Vielzahl unterschiedlicher Ontologien binnen kürzester Zeit zu klassifizieren. Es wird das

[9] http://hermit-reasoner.com/

sogenannte „core blocking“-Verfahren eingesetzt, mit dem Ziel, die vom Reasoner benötigte Speicherkapazität auf das Nötigste zu reduzieren.

Es existieren auch weitere Reasoner, die effizienter sein sollten, aber noch in der Entwicklung stehen. Der ELK[10] Reasoner unterstützt nur einen Teil der OWL EL Ontologiesprache und kann nur eingeschränkt benutzt werden. Folgende Konstrukte der OWL-Sprache sind in ELK 0.3.1 implementiert:

Axiomtypen:	*SubClassOf, EquivalentClasses, DisjointClasses, SubObjectPropertyOf, EquivalentObjectProperties, TransitiveObjectProperty, ReflexiveObjectProperty, ObjectPropertyDomain, ClassAssertion, ObjectPropertyAssertion*
Klassenkonstrukte:	*owl:Thing, owl:Nothing, ObjectIntersectionOf, ObjectSomeValuesFrom, ObjectHasValue, DataHasValue*
Property-Konstrukte:	*ObjectPropertyChain*
Individuum-Konstrukte:	*NamedIndividual*

Abbildung 14: Übersicht über alle aktuell unterstützten Features von ELK

Komplexe Konzeptabfragen (nach Subklassen, Superklassen und Instanzen) sind zurzeit nicht möglich. Dies führt entweder zu Fehlern oder zu der niedrigen Performanz. Zwar können einige, nicht implementierte Methoden mit *try-catch*-Blöcken wie in der Abbildung 15 abgefangen und ein *Thing* als Top-Konzept zurückgegeben werden, ist es aber keine akzeptable Lösung für eine effiziente Arbeitsweise. Beim Aufruf der Methode *reasoner.getObjectPropertyRanges()* würde der ELK-Reasoner eine Meldung über *not supported methode* bringen und sofort seine Arbeitsweise beenden, sodass der Lernalgorithmus gar nicht zum Einsatz kommt.

```
@Override
public Description getRangeImpl(ObjectProperty objectProperty) {
        OWLObjectProperty prop = OWLAPIConverter.getOWLAPIObjectProperty(objectProperty);
        NodeSet<OWLClass> set;
```

[10] http://www.cs.ox.ac.uk/isg/tools/ELK/

```
        try {
                set = reasoner.getObjectPropertyRanges(prop, true);
                if (set.isEmpty())
                        return new Thing();
                OWLClass oc = set.iterator().next().getRepresentativeElement();
                if (oc.isOWLThing()) {
                        return Thing.instance;
                }
                return new NamedClass(oc.toStringID());
        } catch (Exception e) {
                return Thing.instance;
        }
}
```

Abbildung 15: Abfangen der Methode getObjectPropertyRanges

Einen Lösungsansatz für komplexe Konzeptabfragen bietet ein *Workaround*, eine zu implementierende Klasse namens *QueryingUnnamedClassExpressions*. Diese ist im Anhang zu finden. Dabei wird eine neue Klasse äquivalent zu der gewünschten komplexen Konzeptabfrage eingeführt und nach der neuen Klasse abgefragt. Die Implementierung scheiterte aber dabei an der "sauberen", fehlerfreien Übergabe von Konzepten. Die Entwicklung von ELK Reasoner verläuft sehr aktiv, was an der *issue list*[11] zu sehen ist, sodass in naher Zukunft gewiss ein Reasoner mit Top-Performanz für alle neue Features zur Verfügung stehen wird.

Der CEL-Reasoner findet einen breiten Einsatz auf großen medizinischen und biologischen Ontologien. CEL steht für *„A polynomial-time **C**lassifier for the description logic **EL**+”* und hat EL als zugrundeliegende Sprache. Aufgrund der Struktur der Ontologien, speziell im biologischen und medizinischen Bereich, ist EL+ ausreichend, um diese Ontologien darzustellen. Durch die geringe Komplexität der Sprache EL+ sollte das Subsumptionsproblem in polynomieller Zeit lösbar sein. Beim Testen des Reasoners wurde jedoch schnell festgestellt, dass die nicht unterstützten Konzeptkonstrukte ähnlich wie bei ELK Reasoner zu Fehlern führen und der Reasoner abbricht.

[11] http://code.google.com/p/elk-reasoner/issues/list

4.2 Lernalgorithmen

4.2.1 Refinementoperatoren in OWL/DLs

Die eingesetzten Algorithmen basieren auf Refinementoperatoren. Diese werden hier kurz erläutert. Das Ziel des Lernvorgangs ist, eine Konzeptbeschreibung in Abhängigkeit von positiven und negativen Beispielen zu finden. Dieser Vorgang kann als ein Suchprozess im Raum von Konzepten betrachtet werden. Als Ordnung gilt die Subsumption, d.h. wenn ein Konzept C subsumiert D, dann erfasst C auch alle Beispiele, die vom Konzept D erfasst werden (vgl. [Lehmann und Hitzler, 2010]).

Definition **Refinementoperator**
Gegeben sei eine Beschreibungssprache *L* und ein quasi-geordneter Raum *[C(L), $\subseteq_T$]* über Konzepten in *L*. Ein *downward L refinement operator p* ist eine Abbildung *p:C(L)* $\rightarrow 2^{C(L)}$, wenn für alle C $\in$ *C(L)* gilt: D $\in$ p(C) impliziert D $\subseteq_T$ *C*.

Eine Refinementkette von Refinementoperator p der Länge n von Konzept C zu Konzept D ist eine endliche Sequenz von Konzepten $C_0, C_1, ..., C_n$, so dass $C = C_0, C_1 \in p(C_0), C_2 \in p(C_1), ..., C_n \in p(C_{n-1}), D = C_n$. Oft notiert durch $C \rightarrow_p C_0 \rightarrow_{p \ldots} \rightarrow_p D$.

Refinementkette: $\top \rightarrow_p$ Person $\rightarrow_p$ Frau $\rightarrow_p$ Mutter

Refinementoperatoren haben bestimmte Eigenschaften, die genutzt werden können, um deren Einsatz für die Lernhypothesen zu optimieren.
Eigenschaften von Refinementoperatoren:

- *endlich*, falls $p(C_0)$ für jedes C $\in$ *C(L)* endlich ist.
- *redundant* falls mehrere *p*-Refinementketten von einem Konzept C zu einem Konzept D existieren
- *echt*, falls C $\rightarrow_p$ D impliziert C $\neq_p$ D
- *vollständig*, falls es für alle C,D $\in$ *C(L)* mit D $\subset_T$ C ein Konzept E $\equiv_T$ D mit C $\rightarrow_{p \ldots} \rightarrow_p$ E gibt
- *schwach vollständig*, falls es für alle C $\in$ *C(L)* mit C $\subset_T \top$

ein Konzept D ≡ C mit $\top \rightarrow_p \ldots \rightarrow_p D$ gibt

- *ideal* = endlich + echt + vollständig

Für weitere theoretische Grundlagen und für ein tieferes Eintauchen in die Theorie der Refinement Operatoren wird hier auf [Lehmann und Hitzler, 2010] und [Lehmann und Hitzler, 2008] verwiesen.
In der Praxis ist es schwer einen idealen Operator in OWL und in den meisten Beschreibungslogiken zu finden, ausgenommen die *EL*-Familie. Deswegen kommt es auch auf den Lernalgorithmen an, wie gut er Nachteile von Operatoren durch seine internen Maßnahmen kompensieren kann. Ein Lernalgorithmus wird als eine Kombination aus Refinementoperator und Suchalgorithmus konstruiert.

4.2.2 OCEL

Wie schon erwähnt, wird ein Lernalgorithmus als eine Kombination aus Refinementoperator, der den Aufbau des Suchbaumes bestimmt und aus Suchalgorithmus, der die Suche kontrolliert, aufgebaut. Wird z.B. der top-down Ansatz benutzt wie in der Abbildung 16, startet der Algorithmus mit Top-Konzept und nutzt den Operator, um Verfeinerungen vorzunehmen. Knoten *Person*, *Werkzeug* und *Gebäude* werden bewertet. Mit Hilfe der heuristischen Suche wird der beste Knoten im nächsten Schritt erweitert. Erweiterung des Knotens zieht wieder die Anwendung des Operators nach sich, so dass wieder Verfeinerungen vorgenommen werden und so weiter. Wenn das Abbruchkriterium erfüllt wird, terminiert der Algorithmus und gibt die Lösung aus. OCEL und CELOE arbeiten nach diesem Basisprinzip.

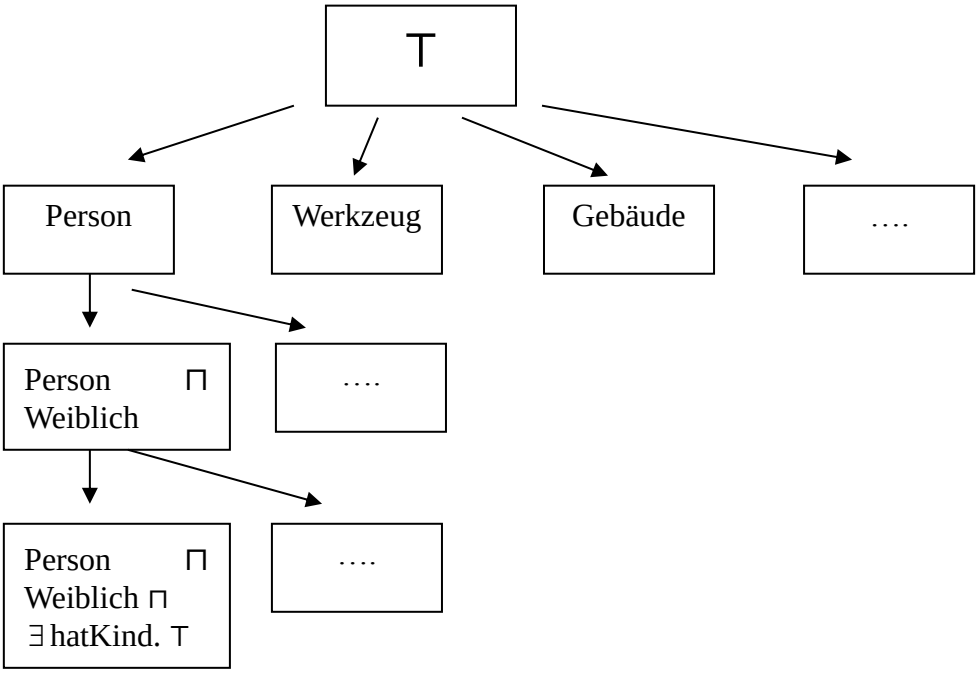

Abbildung 16: Suchbaum mit Top-Down Ansatz

OCEL (*OWL Class Expression Learner*) gilt als Standard-Lernalgorithmus und nutzt Refinementoperator *p* für die top-down Suche. OCEL ist vollständig, d.h. wenn eine Lösung existiert, wird diese auch gefunden. OCEL implementiert Redundanzeliminierung (vgl. [Lehmann, 2010]) mit niedriger Komplexität bzgl. der Größe des Suchbaums. Dies erfolgt durch die Überprüfung jedes Knotens auf eine eventuelle Existenz eines vorhandenen Knotens mit der gleichen niedrigen Komplexität. Falls der Vergleich zutrifft, wird der jeweilige Knoten nicht mehr erweitert und somit eliminiert. Unendliche Operatoren können durch schrittweise längenbeschränkte Knotenexpansion verwenden werden. Für weitere Angaben wird auf [Lehmann, 2010] verwiesen.

4.2.3 CELOE

CELOE (Class Expression Learning for Ontology Engineering) ist eine Erweiterung von OCEL und enthält einige spezifische Änderungen bezüglich des Klassen-Lernproblems (vgl. [Lehmann et al., 2011]) Der Algorithmus nutzt denselben Ansatz wie OCEL. Der Unterschied liegt in der Definition von unterschiedlicher Heuristik, die für die breite Suche optimiert ist, so dass häufig komplexe Konzepte, die bessere Lösungen darstellen würden, nicht gefunden

werden. Dafür werden einfache Lösungen schnell berechnet. Somit eignet sich CELOE zum Einsetzen bei Ontologie-Editoren, bei denen die langen, komplexen Konzepte unwahrscheinlich sind. Abschließend wird noch einmal festgestellt, dass der CELOE-Algorithmus weniger geeignet ist, um die komplexen Konzeptbeschreibungen zu entwickeln. Dies hat sich beim Lernverfahren auch bestätigt.

5 Vorverarbeitung der Wissensbasen

Aufgrund der Größe der MP-Ontologie und der MP Equivalence Axioms Subq Ontologie wurden diese vor dem Einsatz im DL-Learner „bereinigt“. Mit Protégé[12], einem Editor zur Modellierung von Ontologie, wurden alle Annotationen entfernt, sodass jede Klasse in der MP-Ontologie den folgenden Aufbau hat:

```
<owl:Class rdf:about="&obo;MP_0000813">
  <rdfs:subClassOf rdf:resource="&obo; MP_0000807"/>
</owl:Class>
```

Abbildung 17: Klasse MP_0000813 ohne Annotationen

Dazu die entsprechende, äquivalente Klasse aus der MP Equivalence Axioms Subq Ontologie:

```
<owl:Class rdf:about="&obo;MP_0000813">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&obo;BFO_0000051"/>
      <owl:someValuesFrom>
        <owl:Class>
          <owl:intersectionOf rdf:parseType="Collection">
            <rdf:Description rdf:about="&obo;PATO_0000051"/>
            <owl:Restriction>
              <owl:onProperty rdf:resource="&obo;BFO_0000052"/>
              <owl:someValuesFrom rdf:resource="&obo;MA_0002427"/>
            </owl:Restriction>
            <owl:Restriction>
              <owl:onProperty rdf:resource="&obo;RO_0002180"/>
              <owl:someValuesFrom rdf:resource="&obo;PATO_0000460"/>
            </owl:Restriction>
```

[12] http://protege.stanford.edu/

```
            </owl:intersectionOf>
          </owl:Class>
        </owl:someValuesFrom>
      </owl:Restriction>
    </owl:equivalentClass>
  </owl:Class>
```

Abbildung 18: äquivalente Klasse MP_0000813 ohne Annotationen

Wenn man die Abbildung betrachtet, wird es ersichtlich, dass es nur die Subsumptionsbeziehung, die für uns relevant ist. Nach Entfernen von Annotationen stehen die beiden Ontologien zum Einsatz bereit.

Positive und negative Beispiele stehen unter folgender Adresse zur Verfügung: *ftp://ftp.informatics.jax.org/pub/reports/index.html*.

MGI_Geno_Disease.rpt – positive Beispiele

MGI_Geno_NotDisease.rpt – negative Beispiele

Bei einer Datei mit RPT Endung handelt es sich um einen Report, der in Text Format verfasst ist. Zum Öffnen der Dateien kann jeder beliebiger Texteditor benutzt werden. Die Dateien haben folgende Struktur:

Spaltenbezeichnung	Beispiel
Allelic Composition	Ctnna2<tm1Mta>/Ctnna2<tm1Mta>
Allele Symbol(s)	Ctnna2<tm1Mta>
Allele ID(s)	MGI:3623589
Genetic Background	Not Specified
Mammalian Phenotype ID	MP:0000813
PubMed ID	12123610
MGI Marker Accession ID	MGI:88275
OMIM ID	222100

Tabelle 6: Struktur des Beispiels MGI:3623589

Ein ganzer Datensatz ist in einer Zeile und die Informationen Tabulator getrennt gespeichert.

Für das Lernen werden nicht alle Daten genutzt, sodass die Komposition von Allele[13], Allele Symbol, genetischer Hintergrund, PubMed ID, MGI Marker Accession ID und OMIM ID entfernt werden. Notepad++ ist ein passendes Werkzeug dazu und beruht ebenfalls auf Open-Source. Für die Angabe der Beispiele werden Allele-ID (*MGI_3623589*) und Mammalian Phenotype ID (*MP_0000813*) genutzt. Ein MGI-Eintrag entspricht einer Allele-ID aus *Mouse Genome Informatics*[14], einer internationalen Online-Datenbank, in der die wichtigsten Informationen über das Genom und den Stoffwechsel der Maus (*Mus musculus*) gesammelt und frei verfügbar gemacht werden. Ein MP-Eintrag entspricht einem Phänotyp aus der Mammalian Phenotype Ontologie. Diese bilden ein Triple in Notation3 (N3), einer lesbaren RDF-Syntax, sodass die Beispiele folgende Struktur aufweisen:

<http://www.informatics.jax.org/obo/MGI_3623589> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://purl.obolibrary.org/obo/MP_0000813>.

Abbildung 19: Beispiel MGI_3623589 in N3-RDF-Syntax

Mit *http://www.w3.org/1999/02/22-rdf-syntax-ns#type* wird angegeben, dass *<http://purl.obolibrary.org/obo/MP_0000813>* eine Instanz von *rdfs:Class*, also eine Klasse (ein Konzept) ist und *<http://www.informatics.jax.org/obo/MGI_3623589>* eine Instanz dieser Klasse ist.

Positive und negative Beispiele werden in einer OWL-Datei (*examples.owl*) hintereinander abgespeichert und später in der Konfigurationsdatei "*train.conf*" im DL-Learner durch MGI-Einträge gekennzeichnet.

[13] eine mögliche Ausprägung eines Gens, das sich an einem bestimmten Ort auf einem Chromosom befindet.
[14] http://www.informatics.jax.org/

6 Lernprozess

Im Folgenden wird die Konfigurationsdatei erläutert, mit Hilfe derer alle Konfigurationen vorgenommen wurden und anschließend werden Ergebnisse dargestellt.
Zum Testen wird eine Text-Datei "*train.conf*" genutzt, die alle Konfigurationen beinhaltet. Als Erstes wird ein Präfix definiert, um die Eingabe von Beispielen zu vereinfachen. In unserer Konfigurationsdatei steht "*kb*" für "*http://www.informatics.jax.org/obo/*". Danach folgen die Definitionen von Wissensbasen, die beim Testen eingesetzt werden. Für diese ist es genügend, einen Datentyp *(ks1.type = "OWL File"*) und einen Namen (*ks1.fileName = "mp.owl"*) zu deklarieren. In unserem Fall werden drei OWL-Dateien miteinbezogen. Die erste Datei (*ks*) enthält positive als auch negative Beispiele. Die zweite und die dritte Dateien sind die eingesetzten Ontologien. Als Nächstes folgt die Definition von Reasoner. Es wird eingegeben, dass es sich um einen OWL API Reasoner handelt, und zwar um HermiT-Reasoner in diesem Fall. Die definierten Wissensbasen werden dem Reasoner übergeben. Mit *alg.type = "OCEL"* wird der Lernalgorithmus bestimmt und durch weitere Attribute wie *noisePercentage*(näherungsweise, Anteil des Rauschens innerhalb der Beispiele) verfeinert. Als Letztes wird das Lernproblem (*lp.type = "posNegStandard"*) definiert, sowie die positiven und negativen Beispiele eingegeben. Die Reihenfolge der Konfigurationen ist dabei nicht relevant.

```
/** conf-datei  */
prefixes = [ ("kb","http://www.informatics.jax.org/obo/") ]

// knowledge source definition
ks.type = "OWL File"
ks.fileName = "expamples.owl"
ks1.type = "OWL File"
ks1.fileName = "mp.owl"
ks2.type = "OWL File"
ks2.fileName = "mp-equivalence-axioms-subq2.owl"
// reasoner
```

```
reasoner.type = "OWL API Reasoner"
reasoner.sources = { ks, ks1, ks2}
reasoner.reasonerTypeString = "hermit"
// learning algorithm
alg.type = "OCEL"
alg.noisePercentage = 30
alg.writeSearchTree = true
alg.replaceSearchTree = true
alg.searchTreeFile = "log/searchTree.log"
//learning problem
lp.type = "posNegStandard"
lp.positiveExamples = { "kb:MGI_1855930", ... , "kb:MGI_1855937"}
lp.negativeExamples = { "kb:MGI_3849598", ... , "kb:MGI_1855937"}
```

Abbildung 20: Konfigurationsdatei train.conf

Alle Tests wurden mit dem HermiT-Reasoner durchgeführt. Andere Reasoner wie ELK und CEL wurden ebenfalls getestet. Aufgrund von nicht unterstützten Konstrukten kam es aber dabei zu Fehlern, so dass diese nicht weiter eingesetzt wurden. Zuerst werden in der Tabelle 22 einige Informationen zu den Beispielen und Ontologien gemacht.

Anzahl von Konzepten in mp.owl	9052
Anzahl von Konzepten in axioms.owl	16509
Anzahl pos. Beispiele	28656
Anzahl neg. Beispiele	1144
Gesamtanzahl der Beispiele	29800

Tabelle 7: Statistik über Wissensbasen

Der folgende Test wurde mit OCEL-Algorithmus und mit 100 positiven und 100 negativen Beispiele durchgeführt:

```
//Initianlisierung
DL-Learner command line interface
Initializing Component "OWL File"... OK (0ms)
Initializing Component …
```

//Starten des Lernvorgangs
Running algorithm instance "alg" (OCEL)
starting top down refinement with: Thing (35,294% accuracy)
start node: TOP [acc:35,294% h:0,473 q:0p-11n (START), he:0 c:0]
currently best node: TOP [acc:35,294% h:0,473 q:0p-11n (START), he:0 c:0]
currently best node KBSyntax: TOP
next expanded node: TOP [acc:35,294% h:0,473 q:0p-11n (START), he:0 c:0]
algorithm runtime 0ms
size of candidate set: 1
subsumption time: 0ms
instance check time: 0ms
retrieval time: 0ms
properness tests (reasoner/short concept/too weak list): 0/0/0
concept tests (reasoner/too weak list/overly general list/redundant concepts): 0/0/0/0
--- loop 0 started ---

//Ausgabe der Lösungen
solutions (at most 20 are shown):
1: obo::MP_0003631 (accuracy 82,353%, length 1, depth 1)
2: obo::MP_0003632 (accuracy 82,353%, length 1, depth 1)
3: obo::MP_0003861 (accuracy 76,471%, length 1, depth 1)
4: obo::MP_0002108 (accuracy 76,471%, length 1, depth 1)
5: obo::MP_0001186 (accuracy 70,588%, length 1, depth 1)
6: obo::MP_0002075 (accuracy 70,588%, length 1, depth 1)
7: obo::MP_0000371 (accuracy 70,588%, length 1, depth 1)
….
19: obo::MP_0002152 (accuracy 70,588%, length 1, depth 1)
20: obo::MP_0000913 (accuracy 70,588%, length 1, depth 1)

size of candidate set: 265
properness tests (reasoner/short concept/too weak list): 0/0/0
concept tests (reasoner/too weak list/overly general list/redundant concepts): 5174/0/0/0
Algorithm terminated successfully (5174 descriptions tested).

number of instance checks: 25911 (0 multiple)
instance check reasoning time: 1s 602ms (0ms per instance check)
overall reasoning time: 11s 873ms

Wie die Ausgabe zeigt terminiert der Algorithmus erfolgreich. Als beste Lösung werden dabei die Konzepte MP_0003631 und MP_0003632 vorgeschlagen. Insgesamt werden 25911 Instanzen überprüft und 5174 Konzepte getestet.

Mit einer größeren Menge von Beispielen (jeweils 1000) erfolgt die folgende Ausgabe:
1: obo::MP_0005381 (accuracy 73,545%, length 1, depth 1)
2: obo::MP_0000462 (accuracy 73,545%, length 1, depth 1)
3: obo::MP_0003743 (accuracy 72,487%, length 1, depth 1)
4: obo::MP_0002116 (accuracy 71,429%, length 1, depth 1)
5: obo::MP_0005367 (accuracy 71,429%, length 1, depth 1)
6: obo::MP_0001186 (accuracy 70,899%, length 1, depth 1)
7: obo::MP_0005382 (accuracy 70,899%, length 1, depth 1)
8: obo::MP_0000428 (accuracy 70,899%, length 1, depth 1)
9: obo::MP_0000477 (accuracy 70,37%, length 1, depth 1)
….
Es werden wieder nur einfache Konzepte (MP_0005381 und MP_0000462) als Lösung ausgegeben, sodass das Ziel eine komplexe Konzeptbeschreibung zu entwickeln, nicht erreicht wird. Bei diesem Test wurden 242878 Instanzen überprüft und 5195 Konzepte getestet.

Bei weiteren Tests wurden alle negativen Beispiele eingesetzt und die Menge von positiven Beispielen wurde kontinuierlich erhöht. Folgender Test enthält 4000 Beispiele (davon 1000 negative und 3000 positive, Instanz-checks: 606601, getestete Konzepte: 4152, Zeit: ca. 2,5h):
1: obo::MP_0000001 (accuracy 66,89%, length 1, depth 1)
2: obo::MP_0010771 (accuracy 57,86%, length 1, depth 1)
3: obo::MP_0001764 (accuracy 55,518%, length 1, depth 1)
4: obo::MP_0002089 (accuracy 54,849%, length 1, depth 1)
5: obo::MP_0005376 (accuracy 54,515%, length 1, depth 1)
6: obo::MP_0005378 (accuracy 53,512%, length 1, depth 1)
7: obo::MP_0004924 (accuracy 53,177%, length 1, depth 1)

8: obo::MP_0010769 (accuracy 51,839%, length 1, depth 1)

9: obo::MP_0001186 (accuracy 50,836%, length 1, depth 1)

…

Die weiteren Tests führten zum selben Ergebnis. Es wurden immer nur die einfachen Konzepte als beste Lösungen ausgegeben. Diese werden hier nicht weiteraufgezeigt, da sie für die Auswertung nicht ausschlaggebend sind. Beim Testen mit dem CELOE-Algorithmus wurde festgestellt, dass dieser nur einen einzigen Knoten (MP_0000001) als beste Lösung ausgibt und keine weiteren, komplexen Konzepte findet.

Bei der abschließenden Betrachtung der Testergebnisse kommt man zu der Erkenntnis, dass die Ergebnisse nicht zufriedenstellend sind. Dies liegt einerseits an den Eingabedaten. Durch die große Menge von Konzepten wird das Reasoning ineffizient. Der Einsatz von ELK und CEL Reasoner, die das Reasoning übernehmen sollten, ist gescheitert, so dass der HermiT Reasoner zum Einsatz kam. Dieser braucht aber schon bei 4000 Beispielen ungefähr drei Stunden Zeit. Somit wurde nur ein Teil von Beispielen zum Testen eingesetzt. Dieser reichte trotzdem aus, um die Tendenz zu erkennen. Andererseits liegt bei den Wissensbasen ein etwas anderes Lernproblem vor. Das Wissen zum Extrahieren ist durch Schemawissen und nicht durch Instanzen, wie das üblich der Fall ist, vorgegeben. Dies erfordert eine Änderung der Arbeitsweise der Lernalgorithmen. Statt eines Instanz-checks wird eine Subsumption effizienter.

7 Ausblick

Um eine komplexe Konzeptbeschreibung aus MP Ontologie und MP Logical Definitions Subq Ontologie zu entwickeln, werden folgende Lösungsansätze vorgeschlagen. Um das Problem zu lösen, ist es nicht notwendig die ganze Wissensbasis zu betrachten. Es kann dazu ein Fragment aus der Ontologie genutzt werden, welches einerseits genug Informationen hat, um die guten Ergebnisse zu ermitteln und andererseits klein genug ist, um das effiziente Reasoning zu ermöglichen. Die Fragmentextraktion soll unter Beachtung einiger Richtlinien durchgeführt werden. So muss das Fragment alle Beispielinstanzen selbst enthalten. Dazu kommen alle Konzepte von Instanzen und die jeweiligen Abhängigkeiten, z.B. durch Eigenschaften. Das Basisprinzip der Fragmentextraktion wird in der Abbildung 21 graphisch dargestellt. Die genauere Beschreibung ist in [Hellmann et al., 2009].

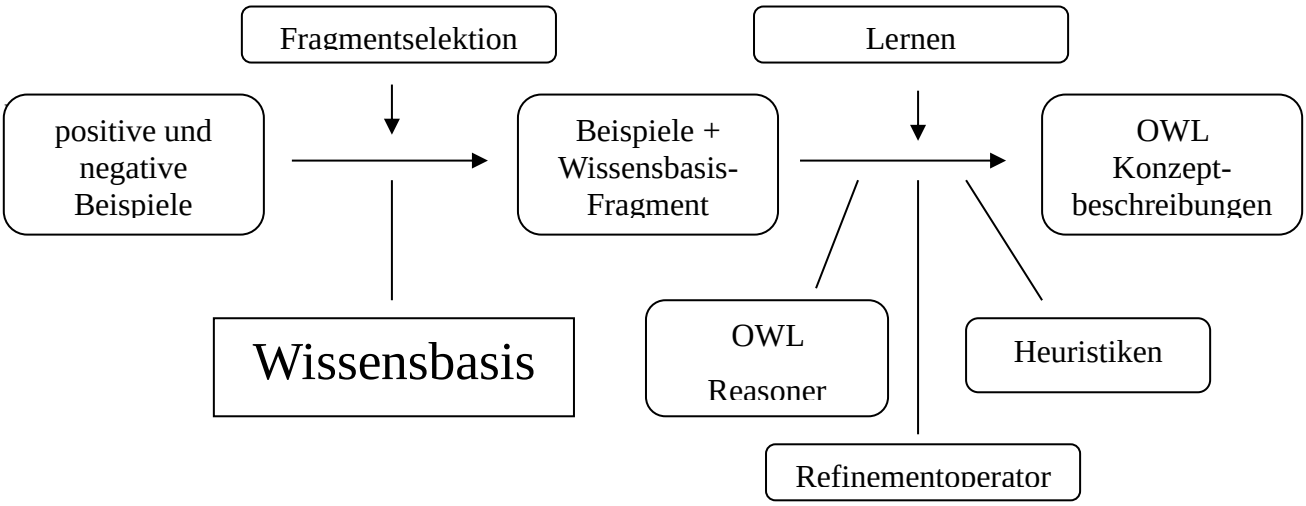

Abbildung 21: Extraktion von Fragmenten

Die Laufzeit von Lernprozessen ist abhängig einerseits von der Größe der Wissensbasen und andererseits von den eingesetzten Komponenten. Als Komponenten wurden HermiT Reasoner, sowie die beiden Lernalgorithmen OCEL und CELOE eingesetzt. Diese arbeiten mit Heuristiken und diese wiederum benötigen oft viele Instanzüberprüfungen, was die Algorithmen sehr langsam macht.

Ein anderer Lösungsansatz ist der Einsatz vom idealen *EL* Refinementoperator [Lehmann und Haase, 2009] und deren Anpassung an das vorliegende Problem. EL-Konzepte können als direkte, markierte Bäume repräsentiert werden [Baader et al., 1999]. Das Reasoning erfolgt dann durch Simulationen auf Bäumen. Der Refinementoperator ist eine Funktion, welche einen Baum $t \in T_{min}$ auf eine Teilmenge von T_{min} abbildet, wobei T_{min} die Menge von minimalen EL Bäumen ist. Die Abbildungsfunktion kann auf das Subsumptionsproblem zurückgeführt werden, so dass der Refinementoperator die Subsumption als Basisoperation durchführt.

Ein anderer Lösungsvorschlag basiert auf dem LGG[15]- Algorithmus. Bei diesem Ansatz wird Subsumption als eine Relation zwischen Hypothesen definiert und erlaubt eine Überprüfung, ob eine Hypothese ein Beispiel subsumiert und damit allgemeiner ist als eine andere. Bei einer gegebenen Menge *E* von Beispielen wird eine Hypothese $h \in H$ (H = Menge von Hypothesen) als eine *least general generalisation* von E bezeichnet falls,

- *h* alle Beispiele von E subsumiert und
- es keine weitere Hypothese existiert, die alle Beispiele von E subsumiert und selbst von h subsumiert wird.

Der Lernalgorithmus berechnet die *least general generalisations* für alle Mengen von Beispielen. Für weitere Informationen wird hier auf [Tantini et al., 2010] und [Muggleton, 1991] verwiesen.

[15] *least general generalisations*

8 Zusammenfassung

Bei dieser Arbeit wurden zwei Ontologien, MP Ontologie und die MP Logical Definitions Subq Ontologie, als Wissensbasen zum Lernen in DL-Lerner, einem Framework zum maschinellen überwachten Lernen in OWL und Beschreibungslogiken, eingesetzt. Dazu wurden kranke und gesunde Phänotypen von Hausmäusen als positive und negative Lernbeispiele einem Lernprozess unterzogen.

Bei den Grundlagen gab es eine Einführung in die wichtigen Grundbegriffe und in die Beschreibungslogiken. Das eingesetzte Framework wurde mit seinen Komponenten kurz beschrieben.

Bei der Repräsentation von Phänotypen wurden zwei Formate eingeführt und die beiden eingesetzten Ontologien anhand von Beispielen erläutert.

In der Vorverarbeitungsphase wurden bei Ontologien die Annotationen entfernt und die Beispiele in eine N3-RDF-Syntax überführt, so dass diese im DL-Learner eingesetzt werden könnten.

Beim Lernprozess wurden verschiedene Komponenten von DL-Learner eingesetzt und bezüglich ihrer Effizienz getestet. Als Reasoner kamen dabei ELK Reasoner, CEL Reasoner und HermiT Reasoner zum Einsatz, wobei die ersten beiden aufgrund von nicht unterstützten Sprachkonstrukten nicht verwendet werden könnten. Alle Tests wurden mit HermiT Reasoner durchgeführt. Als Lernalgorithmen wurden OCEL und CELOE eingesetzt.

Im Ausblick wurden drei Lösungsansätze vorgeschlagen und kurz beschrieben mit entsprechenden Literaturhinweisen.

9 Anhang

Klasse *QueryingUnnamedClassExpressions*

```
import org.semanticweb.elk.owlapi.ElkReasonerFactory;
import org.semanticweb.owlapi.apibinding.OWLManager;
import org.semanticweb.owlapi.model.IRI;
import org.semanticweb.owlapi.model.OWLAxiom;
import org.semanticweb.owlapi.model.OWLClass;
import org.semanticweb.owlapi.model.OWLClassExpression;
import org.semanticweb.owlapi.model.OWLDataFactory;
import org.semanticweb.owlapi.model.OWLObjectProperty;
import org.semanticweb.owlapi.model.OWLOntology;
import org.semanticweb.owlapi.model.OWLOntologyCreationException;
import org.semanticweb.owlapi.model.OWLOntologyManager;
import org.semanticweb.owlapi.model.PrefixManager;
import org.semanticweb.owlapi.reasoner.OWLReasoner;
import org.semanticweb.owlapi.reasoner.OWLReasonerFactory;
import org.semanticweb.owlapi.util.DefaultPrefixManager;

import java.io.File;

public class QueryingUnnamedClassExpressions {

  public static void main(String[] args) throws OWLOntologyCreationException {
    OWLOntologyManager man = OWLManager.createOWLOntologyManager();
    OWLDataFactory dataFactory = man.getOWLDataFactory();

    // Load your ontology.
    OWLOntology ont = man.loadOntologyFromOntologyDocument(new
File("c:/ontologies/ontology.owl"));

    // Create an ELK reasoner.
    OWLReasonerFactory reasonerFactory = new ElkReasonerFactory();
    OWLReasoner reasoner = reasonerFactory.createReasoner(ont);

    // Create your desired query class expression. In this example we
    // will query ObjectIntersectionOf(A ObjectSomeValuesFrom(R B)).
    PrefixManager pm = new DefaultPrefixManager("http://example.org/");
    OWLClass A = dataFactory.getOWLClass(":A", pm);
    OWLObjectProperty R = dataFactory.getOWLObjectProperty(":R", pm);
    OWLClass B = dataFactory.getOWLClass(":B", pm);
    OWLClassExpression query = dataFactory.getOWLObjectIntersectionOf(A, dataFacto-
ry.getOWLObjectSomeValuesFrom(R, B));

    // Create a fresh name for the query.
    OWLClass newName = dataFactory.getOWLClass(IRI.create("temp001"));

    // Make the query equivalent to the fresh class
    OWLAxiom definition = dataFactory.getOWLEquivalentClassesAxiom(newName, query);
    man.addAxiom(ont, definition);

    // Remember to either flush the reasoner after the ontology change
    // or create the reasoner in non-buffering mode. Note that querying
    // a reasoner after an ontology change triggers re-classification of
    // the whole ontology which might be costly. Therefore, if you plan
    // to query for multiple complex class expressions, it will be more
    // efficient to add the corresponding definitions to the ontology at
    // once before asking any queries to the reasoner.
    reasoner.flush();

    // You can now retrieve subclasses, superclasses, and instances of
    // the query class by using its new name instead.
```

```
    reasoner.getSubClasses(newName, true);
    reasoner.getSuperClasses(newName, true);
    reasoner.getInstances(newName, false);

    // After you are done with the query, you should remove the definition
    man.removeAxiom(ont, definition);

    // You can now add new definitions for new queries in the same way

    // After you are done with all queries, do not forget to free the
    // resources occupied by the reasoner
    reasoner.dispose();
  }
}
```

10 Literaturverzeichnis

[Baader et al., 2003] F. Baader; D. Calvanese; D. L. McGuinness; D. Nardi und P. F. Patel-Schneider, editors (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press. 2003

[Baader et al., 1999] F. Baader; R. Molitor and S. Tobies. *Tractable and decidable fragments of conceptual graphs.* In Seventh International Conference on Conceptual Structures, Springer Verlag, 1999

[Day, 2006] J. Day-Richter: *The OBO Flat File Format Specification*, version 1.2. 2006. Verfügbar unter: http://www.geneontology.org/GO.format.obo-1_2.shtml

[Dugas und Schmidt, 2003] M. Dugas und K. Schmidt: *Medizinische Informatik und Bioinformatik* 2003, s. 21

[Gruber, 1993] T. R. Gruber, *A Translation Approach to Portable Ontology Specifications*. 1993. Verfügbar unter: http://ksl-web.stanford.edu/KSL_Abstracts/KSL-92-71.html

[Hellmann et al., 2009] S. Hellmann; J. Lehmann; S. Auer: *Learning of OWL Class Descriptions on Very Large Knowledge Bases*. International Journal On Semantic Web and Information Systems, 2009. Verfügbar unter: http://jens-lehmann.org/files/2009/dllearner_sparql.pdf

[Hesse, 2002] W. Hesse: *Ontologie(n). In: Informatik Spektrum 25* (2002), S. 477–480. Verfügbar unter: http://www.gi.de/nc/service/informatiklexikon/detailansicht/article/ontologien/druckversion.html

[Hitzer et al., 2008] P. Hitzler; M. Krötzsch; S. Rudolph und Y. Sure: *Semantic Web Grundlagen.* Springer-Verlag, 2008

[Horrocks]I. Horrocks: *Description Logic Reasoning*. University of Manchester, Manchester, UK. Verfügbar unter: http://www.kde.cs.uni-kassel.de/conf/iccs05/horrocks_iccs05.pdf

[Horrocks, 2007] I. Horrocks: *OBO Flat File Format Syntax and Semantics and Mapping to OWL Web Ontology Language.* 2007. Verfügbar unter: http://www.cs.man.ac.uk/~horrocks/obo

[Lehmann, 2010] J.Lehmann: *Learning OWL Class Expressions* Verfügbar unter: http://jens-lehmann.org/files/2010/phd_thesis.pdf

[Lehmann, 2011] J. Lehmann: *DL-Learner Manual*. July 8, 2011 Verfügbar unter: http://dl-learner.org/files/dl-learner-manual.pdf

[Lehmann et al., 2011] J. Lehmann; S. Auer; L. Bühmann; S. Tramp (geb. Dietzold): *Class Expression Learning for Ontology Engineering* Journal of Web Semantics, , volume 9, pages 71-81, Springer, 2011 Verfügbar unter: http://jens-lehmann.org/files/2011/celoe.pdf
[Lehmann und Haase, 2009] J. Lehmann und C. Haase: *Ideal downward refinement in the EL description logic.* Technical report, University of Leipzig, 2009. Verfügbar unter: http://jens-lehmann.org/files/2009/el_ilp.pdf

[Lehmann und Hitzler, 2008] J. Lehmann; P. Hitzler "*Foundations of Refinement Operators for Description Logics*", ILP conference, 2008. Verfügbar unter: http://jens-lehmann.org/files/2008/operator_analysis.pdf

[Lehmann und Hitzler, 2010] J. Lehmann; P. Hitzler*: "Concept Learning in Description Logics Using Refinement Operators",* Machine Learning journal, 2010. Verfügbar unter: http://jens-lehmann.org/files/2010/concept_learning_mlj.pdf

[Manola und Miller, 2004] F. Manola und E. Miller. *Resource Description Framework (RDF).* Primer. W3C Recommendation 10 February 2004, 2004.
Verfügbar unter: http://www.w3.org/TR/rdf-primer

[McGuinness und van Harmelen, 2004] D. L. McGuinness und F. van Harmelen. *OWL Web Ontology Language Overview.* W3C Recommendation 10 February 2004, 2004.
Verfügbar unter: http://www.w3.org/TR/owl-features/

[Michalski und Kodratoff, 1990] R.S. Michalski und Y. Kodratoff. *Machine Learning Vol. III in: Research in Machine Learning; Recent Progress, Classification of Methods, and Future Directions.* 1990

[Muggleton, 1991] S. Muggleton: *Inductive Logic Programming,* New Generation Computing 8 (1991) no.4, 295--318.

[Mungall et al., 2010] , Christopher J. Mungall; Georgios V. Gkoutos; Cynthia L Smith; Melissa A. Haendel; Suzanna E. Lewis; Michael Ashburner: *Integrating phenotype ontologies across multiple species*. In: Genome Biology. PubMed Central, Bethesda 2010: v.11(1).
Verfügbar unter: http://genomebiology.com/2010/11/1/R2

[Shearer et al.]R. Shearer; B. Motik; und I. Horrocks: *HermiT: A Highly-Efficient OWL Reasoner.* Oxford University Computing Laboratory. Verfügbar unter: http://ceur-ws.org/Vol-432/owled2008eu_submission_12.pdf

[Tantini et al., 2010] F. Tantini; A. Terlutte; F. Torre: *Sequences Classification by Least General Generalisations.* Universite Lille Nord de France, 2010

Erklärung

"Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe".

Ort, Datum Unterschrift