TUHH
Technische Universität Hamburg-Harburg

Untersuchungen für Approximationen von Ontologien

Projektarbeit

von
Gürkan Günay
Matrikelnummer: 28901

im

Institut für Softwaresysteme

betreut von
Prof. Dr. rer.-nat. habil. Ralf Möller
Dipl.-Inform. Alissa Kaplunova

Danksagung

Diese Projektarbeit entstand am Institut für Softwaresysteme der Technischen Universität Hamburg-Harburg unter der Leitung von Herrn Prof. Dr. Ralf Möller und ich möchte mich hiermit bei allen bedanken, die zur Entstehung dieser Arbeit beigetragen haben.
Mein besonderer Dank gilt an dieser Stelle Herrn Prof. Dr. Ralf Möller für die Betreuung und Unterstützung während dieser Projektarbeit. Außerdem bedanke ich mich herzlich bei Frau Dipl.-Inform. Alissa Kaplunova für die überaus engagierte Betreuung und die produktiven Ratschläge und Hilfestellungen. Bedanken möchte ich mich auch bei Herrn Dip.-Inform. Tuvshintur Tserendorj vom Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB) der Universität Karlsruhe (TH) für die Hilfestellungen bei Fragen bezüglich des approximativen Reasoners Screech.

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Hamburg, den 19. März 2009 Gürkan Günay

Inhaltsverzeichnis

1 Einleitung 1
- 1.1 Motivation . . . 1
- 1.2 Aufgabenstellung . . . 1
- 1.3 Gliederung der Arbeit . . . 2

2 Grundlagen 3
- 2.1 Semantisches Web . . . 3
- 2.2 Modellierungsparadigmen . . . 3
  - 2.2.1 Klassisches Paradigma . . . 4
  - 2.2.2 Datalog Paradigma . . . 5
  - 2.2.3 Vergleiche . . . 6

3 Systeme 11
- 3.1 Kaon2 . . . 11
- 3.2 Screech . . . 14
  - 3.2.1 Algorithmen . . . 15
  - 3.2.2 Tests . . . 19

4 Experiment 22
- 4.1 OWL Tools Screech . . . 23
- 4.2 Eigene Implementierung . . . 25

5 Zusammenfassung und Ausblick 28
- 5.1 Zusammenfassung . . . 28
- 5.2 Ausblick . . . 28

Literaturverzeichnis 31

Abbildungsverzeichnis 33

Tabellenverzeichnis 34

Listings 35

Abkürzungsverzeichnis 36

A CD-ROM 37
A.1 Inhalt 37

KAPITEL 1

Einleitung

1.1 Motivation

Approximative Schlussfolgerungen (engl. approximate reasoning), auf der Grundlage von approximierten Ontologien, nehmen mit der steigenden Bedeutung vom Semantischem Web (engl. semantic web) eine immer wichtiger werdende Rolle ein. Das Semantische Web sieht eine aus OWL[1] Ontologien bestehende verteilte Wissensquelle vor, die das Wissen so miteinander verbindet, wie heute das Web die Web-Seiten [HV05]. Die große Datenmenge und die hohe Ausdrucksstärke der Modellierungssprache führen dabei zu einer schlechten Skalierbarkeit und einer hohen Komplexität bei der Schlussfolgerung. Da für bestimmte Anwendungen die Geschwindigkeit der Antworten (bzw. der Schlussfolgerungen) jedoch wichtiger als die hundertprozentige Richtigkeit ist, wird mit der approximativen Schlussfolgerung eine Steigerung der Geschwindigkeit zur Lasten der Zuverlässigkeit (engl. soundness) und/oder der Vollständigkeit (engl. completeness) erreicht. Hierzu wird die Ausdrucksstärke der Modellierungssprache und somit die Komplexität reduziert. Um dies möglichst optimal und mit geringsten Einbußen zu erreichen gibt es in diesem Feld zahlreiche Ansätze und Forschungen.

1.2 Aufgabenstellung

Das Ziel meiner Projektarbeit ist es, Approximationen von Ontologien zu untersuchen. Deshalb werden im Rahmen dieser Projektarbeit mögliche Approximierungsansätze, am Beispiel des approximativen Reasoners (engl. approximate reasoner) Screech[2], analysiert. Da Screech selber auf den Reasoner Kaon2[3] basiert, wird hierbei ebenfalls Kaon2 vorgestellt. Kaon2 wandelt OWL DL Ontologien in disjunktive Datalog-Programme (-Regeln) um und zieht seine Schlüsse an Hand dieser Programme. Screech hingegen approximiert diese disjunktiven Datalog-Programme, mit einen von drei verschiedenen Algorithmen, durch nichtdisjunktive Datalog-Programme. Diese Approximationen reduzieren die Ausdrucksstärke der Modellierungssprache und führen dabei zur Steigerung der Geschwindigkeit der

1 Web-Ontologiesprache, http://www.w3.org/2004/OWL/
2 http://logic.aifb.uni-karlsruhe.de/wiki/Screech
3 http://kaon2.semanticweb.org/

Schlussfolgerungen. Die Geschwindigkeitszunahme wird, in Abhängigheit von dem gewählten Algorithmus, auf Kosten der Zuverlässigkeit und/oder der Vollständigkeit erreicht. Ich werde die Umwandlungsalgorithmen mit Beispielen erläutern und schließlich mittels eines selbstimplementierten Java-Programms die Funktionsweise von Screech (an Hand von Instanzenabfragen) demonstrieren.

1.3 Gliederung der Arbeit

Diese Projektarbeit ist in fünf Kapiteln gegliedert. Im nächsten Kapitel gibt es eine kurze Einführung in die, für diese Projektarbeit relevanten, Grundlagen. Es werden dabei das Semantische Web und die zugrunde liegenden Modellierungsparadigmen beschrieben. Auf diesen Grundlagen aufbauend werden im dritten Kapitel die Approximationsansätze am Beispiel von Screech analysiert. Hierzu werden die Umwandlungsalgorithmen und die Architektur erläutert. Da Kaon2 die Grundlage für Screech bietet, wird hierbei zuerst Kaon2 vorgestellt. Im vierten Kapitel wird dann die Funktionsweise von Screech experimentell demonstriert. Abschließend liefert das fünfte Kapitel eine Zusammenfassung und einen Ausblick auf eine mögliche Erweiterung der Architektur.

KAPITEL 2

Grundlagen

2.1 Semantisches Web

Das Semantische Web ist die Umsetzung der Vision von Berners-Lee, dem Erfinder des World Wide Webs [BLHL05]. Die Aufgabe des Semantischen Webs besteht darin, das heutige Web um semantische Technologien zu erweitern und so eine Interoperabilität zwischen den Entitäten im Web zu bieten. Somit würden die Daten von Maschinen nicht nur angezeigt, sondern auch verstanden werden. Dabei würden Informationen von verschiedenen Programmen untereinander verstanden und verarbeitet werden, obwohl diese Programme völlig unabhängig voneinander geschrieben wurden wären. Darüber hinaus würde das Semantische Web eine, auf maschinenverstehbaren Wissen basierende, verteilte Wissensquelle bieten. Auf dieser Wissensquelle basierend könnten Schlussfolgerungssysteme fehlende Informationen herleiten oder Inkonsistenzen aufdecken. Die Anwendungsmöglichkeiten sind so vielfältig, wie die des Webs und reichen von privaten bis hin zu geschäftlichen Anwendungsfeldern: So würde beispielsweise eine maschinenverstehbare Beschreibung des Inhalts und der Funktionsweise von Webdiensten die Interoperabilität entscheidend erweitern.

2.2 Modellierungsparadigmen

Um die Einheitlichkeit im Semantischen Web zu erreichen, muss die Struktur der Wissensbeschreibung bestimmten Beschreibungsmodellen und Konzepten entsprechen bzw. gehorchen. Hierzu wurden Hauptsächlich zwei Paradigmen als zu Grunde liegende Formalismen des Semantischen Webs vorgeschlagen. Während im Klassischen Paradigma auf Beschreibungslogiken basierende Ontologiesprachen wie OWL benutzt werden, bietet das sogenannte Datalog Paradigma hingegen von der Datenbankwelt bekannte Regelsprachen wie Datalog [PH06]. Im Bezug auf die Eignung der beiden Paradigmen in der Repräsentation und in der Schlussfolgerung im Semantischen Web wurden von beiden Seiten viele konkurrierende Behauptungen gemacht: Obwohl sich die von der W3C[1] empfohlene Sprache OWL weitestgehend durchgesetzt hat, wird ihr von Kritikern vorgeworfen Probleme in praktischen Anwendungen zu haben und deshalb nicht den erhofften Durchbruch

1 Das World Wide Web Konsortium, http://www.w3.org/

des Semantischen Webs erreicht zu haben [BLPF05]. Da sich die beiden Formalismen grundlegend unterscheiden, werde ich diese kurz vorstellen und dann an Hand von einigen Beispielen vergleichen (hauptsächlich basierend auf [BLPF05, PH06]).

2.2.1 Klassisches Paradigma

Im klassischen Paradigma wird die zu modellierende Domäne auf abstrakte Weise als eine Menge von Objekten und die Beziehungen zwischen diesen Objekten als Relationen modelliert. Es gibt (potentiel unendlich) viele Beziehungszustände, sogenannte Interpretationen, die jeweils einen möglichen Zustand der Domäne beschreiben. Beispielsweise könnten Personen wie „Bernd“ oder „Lisa“ und Tiere wie „Lassie“ als Objekte und Beziehungen (zwischen Objekten) wie „füttern“ oder „verheiratetSein“ als Relationen repräsentiert werden. Ohne weitere Informationen gibt es für jede Kombinationsmöglichkeit der Objekte und Relationen eine Interpretation. Dabei existieren auch sinnlose Interpretationen wie „Bernd ist verheiratet mit Lassie“. Eine Ontologie besteht aus Axiomen, welche Eigenschaften beschreiben, die von sinnvollen Zuständen der zu modellierenden Welt erfüllt werden müssen. In unserem Beispiel könnte die Ontologie Axiome beinhalten, welche besagen, dass es ein Konzept (bzw. eine Klasse) „Person“ existiert und dass die Relation „verheiratetSein“ Objekte aus dem Konzept „Person“ auf Objekte des selben Konzepts abbildet. Solch eine Ontologie würde dann z.B. Interpretationen ausschließen, wo Personen mit Tieren Verheiratet sind.

Die Ontologiesprache OWL basiert auf Beschreibungslogiken (engl. description logics). Die Beschreibungslogiken sind eine Familie von Formalismen zur Wissensrepräsentation und bilden eine entscheidbare Untermenge der unentscheidbaren Prädikatenlogik erster Stufe. Die Entscheidbarkeit ermöglicht es über eine Beschreibungslogik schlusszufolgern, d.h. aus vorhandenem Wissen neues Wissen zu gewinnen. Daher lässt es sich auch auf Basis der Beschreibungslogik über Ontologien schlussfolgern. Die Wissensbasis (engl. knowledge base) einer Beschreibungslogik wird in der Regel in eine TBox (terminological box) und eine ABox (assertional box) unterteilt [Wik, BCM03]:

Die TBox enthält das sogenannte terminologische Wissen, welches das Wissen über die Konzepte (Mengen von Objekten) einer Domäne und Rollen (binäre Relationen) darstellt. Die Grundform einer Deklaration in einer TBox ist eine Konzeptdefinition, wobei neue Konzepte aus vorher definierten Konzepten bestehen. Beispielsweise kann eine Frau in forgender Weise als eine weibliche Person definiert werden:

$$\text{Frau} \equiv \text{Weiblich} \sqcap \text{Person}$$

Solch eine Deklaration wird im Allgemeinen als eine logische Equivalenz interpretiert und liefert die hinreichende und notwendige Bedingung, um ein Indviduum als eine Frau zu klassifizieren. Als ein weiteres Beispiel beschreibt die folgende Deklaration, dass Mütter Instanzen aus dem Konzept „Frau“ sind, die mindestens in einer „hatKind“ Relation zu einer Instanz aus dem Konzept „Person“ stehen. Mit anderen Worten sind Mütter Frauen, die mindestens eine Person als Kind haben:

$$\text{Mutter} \equiv \text{Frau} \sqcap \exists \text{hatKind.Person}$$

Die ABox hingegen enthält das Wissen über die Instanzen der Konzepte und die

Beziehungen zwischen den Instanzen. Somit wird der Zustand der modellierten Welt repräsentiert. Das folgende Beispiel sagt aus, dass das Individuum Anna eine weibliche Person ist:

$$\text{Weiblich} \sqcap \text{Person}(\text{Anna})$$

Hierbei kann man angesichts der Deklaration des Konzepts der Frau (in der TBox) ableiten, dass Anna eine Instanz des Konzepts „Frau“ ist. Ähnlich beschreibt das folgende Beispiel, dass Anna Dennis als Kind hat:

$$\text{hatKind}(\text{Anna},\text{Dennis})$$

2.2.2 Datalog Paradigma

Im dem, auf Datenbanken basierenden, Datalog Paradigma wird die zu modellierende Domäne ebenfalls als Objekte und Beziehungen modelliert, aber mit einigen Unterschieden. So wird es z.B. angenommen, dass die einzigen, in der Domäne existierenden, Objekte und Relationen diejenigen sind, welche auch explizit in der Datenbank vorhanden sind (closed world assumption). Desweiteren wird angenommen, dass Objekte in der Domäne durch eindeutige Namen identifiziert werden (unique name assumption). Diese Annahmen führen dazu, dass es nur eine Interpretation gibt (die Datenbank selbst), wobei Objekte und Relationen in einer eins-zu-eins Beziehung mit den Daten in der Datenbank stehen.

$$\text{person}(y) \leftarrow \text{person}(x) \wedge \text{elternteil}(x,y)$$

Die obere Regel kann gelesen werden als: für beliebige x und y gelten, dass wenn x eine Person ist und y der Elternteil von x ist, dann ist y auch eine Person (wobei x und y Variablen sind). Dabei ist in diesem Fall „person(y)“ der Kopf und „person(x)∧elternteil(x,y)“ der Rumpf der Regel. Eine Regel ohne Kopf wird oft Beschränkung (engl. constraint) genannt und besagt, dass Situationen, welche die Konditionen im Rumpf erfüllen, nicht zugelassen sind:

$$\leftarrow \text{person}(x) \wedge \text{elternteil}(x,x)$$

Diese Regel sagt aus, dass Personen nicht ihre eigenen Elternteile sein können. Regeln ohne Rümpfe werden dazu benutzt, um Daten zuerfassen. Die folgende Regel besagt zum Beispiel, dass Bernd mit Lisa verheiratet ist:

$$\text{verheiratet}(\text{Bernd}, \text{Lisa}) \leftarrow$$

Man bezeichnet Regeln als sicher (engl. safe), wenn alle Variablen im Kopf der Regel auch (in positiven Atomen) im Rumpf vertreten sind. Deshalb ist die folgende Regel unsicher:

$$\text{mutter}(x,y) \leftarrow \text{person}(x)$$

Aus der einzigen Interpretation resultiert zwar eine geringe Komplexität, dies führt aber auch zugleich zu Einschränkungen beim Modellieren. Daher können nur relativ einfache Situationen modelliert werden, in der die vollständige Information verfügbar

ist. Deshalb kann Datalog beispielsweise nicht ausdrücken, dass alle Personen entweder weiblich oder männlich sind oder dass alle Personen genau zwei Elternteile haben. Es gibt aber auch erweiterte Datalogversionen, wie Datalog$^{\neg\vee}$ mit Negationen im Rumpf und Disjunktionen im Kopf der Regeln. Für solche Ausdrucksstarke Erweiterungen gibt es zwar mehr als eine Interpretation, zugleich steigt damit aber auch die Komplexität. Mit so einer Erweiterung kann man dann zwar einige unvollständige Informationen, wie dass alle Personen entweder weiblich oder männlich sind, modellieren, es ist aber nach wie vor nicht möglich zu modellieren, dass alle Personen genau zwei Elternteile haben. Eine besondere Beziehung zwischen der Beschreibunglogik SHIQ und Datalog ist die, dass eine Schlussfolgerung in der Beschreigungslogik SHIQ auf eine Schlussfolgerung in Datalog$^{\vee}$ (mit Disjunktionen im Kopf der Regeln) reduziert werden kann [HMS04]. Die im 3. Kapitel vorgestellten Reasoner basieren auf dieser Eigenschaft.

2.2.3 Vergleiche

An dieser Stelle werden die beiden Paradigmen durch einige kurze Beispiele verglichen (umfassender Vergleich siehe [PH06]).

Unique Name Assumption

Wie bereits oben beschrieben, werden im Datalog Paradigma Objekte durch eindeutige Namen identifiziert. Eine Person, auf dem mit dem Bezeichner (engl. identifier) Michael_Möller referenziert wird, ist zwangsläufig verschieden von einer Person, auf dem mit dem Bezeichner Mike_Möller referenziert wird. Unter der Annahme, dass diese beiden Bezeichner notwendigerweiser auf verschiedene Personen referenzieren, wird die Möglichkeit ausgeschlossen, dass diese zwei verschiedene Namen der selben Person sind. Im allgemeinen ist das Semantische Web, in dem es verschiedene Informationsquellen gibt, welche ihre eigenen Bezeichner frei wählen dürfen, nicht für die strickte Forderung dieser Annahme geeignet. In einigen Situationen ist diese Annahme jedoch sehr wünschenswert: z.B. bei der Sitzplatzbelegung im Flugzeug. Hierbei möchte man Doppelbelegungen ausschliessen und die eindeutige Zuordnung der Namen auf die einzelnen Sitzplätze erreichen. Im klassischen Paradigma müsste man, um so eine Eindeutigkeit zu erreichen, expliziet für alle Objekte angeben, dass diese untereinander verschieden sind:

$$\text{Michael_Möller} \neq \text{Timo_Schulz}$$

$$\text{Michael_Möller} \neq \text{Andreas_Läufer}$$

$$\text{Timo_Schulz} \neq \text{Andreas_Läufer}$$

Auch wenn es im klassischen Paradigma nicht die Annahme gibt, dass verschiedene Bezeichner verschiede Objekte identifizieren, ist es somit (auch wenn etwas mühsam) möglich zu beschreiben, dass eine gegebene Menge an Objekten diese Eigenschaften haben. Im Datalog Paradigma dagegen identifizieren verschiedene Bezeichner immer verschiedene Objekte.

Closed World Assumption

Im Datalog Paradigma wird zusätzlich angenommen, dass die einzigen in der Domäne existierenden Objekte und Relationen diejenigen sind, welche auch explizit in der Datenbank vorhanden sind. Dabei ist das Semantische Web eine sehr offene Umgebung, wo es nicht vorgeschrieben ist, dass die Informationquellen umfassend sein müssen. Die Annahme, dass das fehlen einer Information äquivalent zur negativen Information ist, führt meist zu falschen Ergebnissen. So könnte es beispielsweise bekannt sein, dass John_Smith zwei verschiedenen Personen kennt:

kennt(John_Smith,Bill_Jones)

kennt(John_Smith,William_Jones)

Bill_Jones $\neq$ William_Jones

John_Smith könnte aber auch noch weitere Personen kennen, von denen wir nicht wissen, dass John_Smith sie kennt. Deshalb wäre es nicht richtig zu behaupten, dass John_Smith nur zwei Personen kennt, nur weil uns zwei Personen, welche er kennt, bekannt sind. Beim klassischen Paradigma dagegen gibt es diese Annhame nicht. Deshalb würden Interpretationen, in der John_Smith weitere Personen kennt nicht ausgeschlossen werden. Dabei gibt es im Datalog Paradigma eine einzige Interpretation, wo John_Smith genau zwei Personen kennt.

Das closed world Verhalten kann man aber im klassischen Paradigma erreichen, wenn man die Anfrageergebnisse entsprechend interpretiert. Auf die Anfrage nach den Personen, die John_Smith kennt, würde man als Antwort Bill_Jones und William_Jones bekommen. Daraus könnte man dann sagen, dass es (nur) bekannt ist, dass John_Smith zwei Personen kennt. Wenn man die Anzahl der Personen, die John_Smith kennt beschränken möchte, kann man dies im klassischen Paradigma auf folgende Weise machen:

John_Smith $\in \leq 2$ kennt

Von der closed world Annahme wird hauptsächlich bei Applikationen profitiert, welche auf Datenbanken basieren. Bei einer Datenbank, in der z.B. die Flugdaten gelistet werden, ist es angemessen anzunehmen, dass alle Reservationen eines Fluges in der Informationsquelle enthalten sind. Wie oben beschrieben, ist es mühsammer das closed world Verhalten im klassischen Paradigma zu erreichen, als im Datalog Paradigma. Beim Datalog Paradigma ist es dagegen jedoch der Fall, dass diese Annahme immer gemacht wird.

Wertebeschränkung (Value Restrictions)

Im klassischen Paradigma ist es möglich, den Bildbereich (engl. range) der Relationen anzugeben:

Passagier $\sqsubseteq \forall$besetzt.Sitzplatz

Das obere Beispiel besagt, dass Passagiere nur Sitzplätze besetzen dürfen. Aus der folgenden „besetzt“ Beziehung kann somit abgeleitet werden, dass Sitzplatz_123 ein Sitzplatz ist,

ohne es vorher angegeben bzw. gefolgert zu haben:

besetzt(John_Smith,Sitplatz_123)

An dieser Stelle kann das Ableiten der Klassenzugehörigkeit über die Definition des Bildbereichs (der Relation) auch kritisiert werden [BLPF05]. Es kann nämlich passieren, dass man z.B. aus der folgenden Beziehung fälschlicherweise folgert, dass William_Jones ein Sitzplatz ist:

besetzt(John_Smith,William_Jones)

Um diese Inkonsistenz aufzudecken müsste man festlegen, dass die beiden Klassen „Passagier“ und „Sitzplatz“ disjunkt sind.

Auf der anderen Seite bietet das Ableiten der Klassenzugehörigkeit über die Definition des Bildbereichs einige Vorteile gegenüber dem Datalog Paradigma. Das folgende Beispiel im klassischen Paradigma besagt z.B., dass Personen nur Personen kennen:

$$\text{Person} \sqsubseteq \forall \text{kennt.Person}$$

Die entsprechende Beschränkungsregel (siehe oben: Regel ohne Kopf) im Datalog Paradigma ist die folgende:

$$\leftarrow \text{person(x)} \wedge \text{kennt(x,y)} \wedge \neg \text{person(y)}$$

Im klassischen Paradigma kann man einfach die folgende „kennt“ Beziehung aufstellen ohne vorher angegeben bzw. gefolgert zu haben, dass Fred_Smith eine Person ist:

kennt(John_Smith,Fred_Smith)

Wie bereits beschrieben, wird die Klassenzugehörigkei von Fred_Smith über die Definition des Bildbereichs abgeleitet. Im Datalog Paradigma hingegen ist die obere Aussage ungültig, da Fred_Smith nicht als ein Objekt der Personenklasse bekannt ist. Man könnte auch eine klassische Regel statt der Beschränkungsregel benutzen:

$$\text{person(y)} \leftarrow \text{person(x)} \wedge \text{kennt(x,y)}$$

So eine Regel kann aber auch nur auf bekannte Objekte angewendet werden, da es im Datalog Paradigma keine Objekte geben kann, die nicht eindeutig bekannt sind.

Wie ich am Anfang geschrieben habe, sieht das Semantische Web eine aus Ontologien bestehende verteilte Wissenquelle vor, wodurch Informationen aus verschieden Quellen abstammen können. Es könnte sein, dass eine Ontologie die Klasse Person und die Beziehung Elternteil bietet und dabei fordert, dass der Elternteil einer Person ebenfalls eine Person ist:

$$\text{Person} \sqsubseteq \forall \text{elternteil.Person} \tag{2.1}$$

Diese Ontologie könnte durch eine weitere Ontologie erweitert werden, welche zusätzlich die Klasse Erwachsener bietet und verlangt, dass der Elternteil einer Person ein Erwachsener

ist:

$$\text{Person} \sqsubseteq \forall \text{elternteil.Erwachsener} \tag{2.2}$$

Dabei können im klassischen Paradigma Informationsquellen, die für die erste Ontologie geschrieben wurden sind, ebenfalls in der zweiten Ontologie benutzt werden:

Person(John_Smith)

elternteil(John_Smith,Fred_Smith)

In diesem Beispiel würde die zusätzliche Einschränkung, dass der Elternteil einer Person ein Erwachsener ist, lediglich zu der zusätzlichen Folgerung führen, dass Fred_Smith ein Erwachsener ist:

Erwachsener(Fred_Smith)

Im Datalog Paradigma gibt es dagegen Einschränkungen beim Kombinieren von Informationsquellen. Beispielsweise wäre die entsprechende Beschränkungsregel vom Axiom (2.1) die folgende Regel:

$\leftarrow$ elternteil(x,y)$\wedge\neg$Person(y)

Diese Regel besagt, dass niemand ein Elternteil sein kann, ohne auch gleichzeitig eine Person zu sein. Dabei wissen wir, dass ein Individuum, welches nicht beweisbar in der Personenklasse ist, als kein Objekt der Personenklasse betrachtet wird. Damit die Regel nicht verletzt wird, muss zusätzlich angegeben werden, dass Fred_Smith eine Person ist:

Person(Fred_Smith)

Wenn man nun die erweiterte Ontologie betrachten, welche zusätzlich verlangt, dass der Elternteil einer Person ein Erwachsener ist, so stellt man fest, dass man erneut neue Informationen hinzufügen muss, um die Regel nicht zu verletzen. Die Beschränkteregel vom (2.2) lautet:

$\leftarrow$ elternteil(x,y)$\wedge\neg$Erwachsener(y)

Damit auch diese Regel nicht verletzt wird, muss zusätzlich angegeben werden, dass Fred_Smith ein Erwachsener ist:

Erwachsener(Fred_Smith)

Kardinalitätseinschränkung (Cardinality Restrictions)

Als ein letztes Beispiel möchte ich die beiden Paradigmen an Hand der Kardinalitätseinschränkung vergleichen. Für einen ausführlichen Vergleich verweise ich an dieser Stelle nochmals auf [BLPF05]. Mit Hilfe der Kardinalitätseinschränkung kann man im klassischen Paradigma erreichen, dass mehrere Bezeichner das selbe Objekt referenzieren. Wie bereits weiter oben erläutert, spielt dies eine wichitge Rolle im Semantischen Web, wo verschiedene

Quellen verschiedene Namen für das selbe Objekt benutzen dürfen. Im Datalog Paradigma hingegen ist das bekanntlich nicht möglich (unique name assumption). Diese wichitge Eigenschaft des klassischen Paradigmas kann aber auch zu Fehlern führen, wenn man nicht darauf achtet ungleiche Individuen als ungleich zu definieren. In dem folgenden Beispiel hat jeder Sitzplatz höchstens einen Passagier. Es gibt einen Sitzplatz Sitzplatz_1 und zwei hatPassagier Beziehungen zu diesem einen Sitzplatz:

$$\text{Sitzplatz} \sqsubseteq \leq 1 \text{ hatPassagier}$$

Sitzplatz(Sitzplatz_1)

hatPassagier(Sitzplatz_1,Fred_Smith)

hatPassagier(Sitzplatz_1,John_Smith)

Durch die Kardinalitätseinschränkung würde man an dieser Stelle fälschlicherweise folgern, dass Fred_Smith und John_Smith die selbe Person sind, wobei hier offensichtlich ein Sitzplatz zweimal vergeben wurde. Um Fehlbuchungen zu vermeiden und die Inkonsistenz aufzudecken müsste man festlegen, dass Fred_Smith und John_Smith nicht die gleichen Objekte sind:

$$\text{Fred_Smith} \neq \text{John_Smith}$$

KAPITEL 3

Systeme

In diesem Kapitel werden, am Beispiel des approximativen Reasoners Screech, mögliche Approximierungsansätze analysiert. Da Screech selber auf den Reasoner Kaon2 basiert, wird erst einmal Kaon2 kurz vorgestellt.

3.1 Kaon2

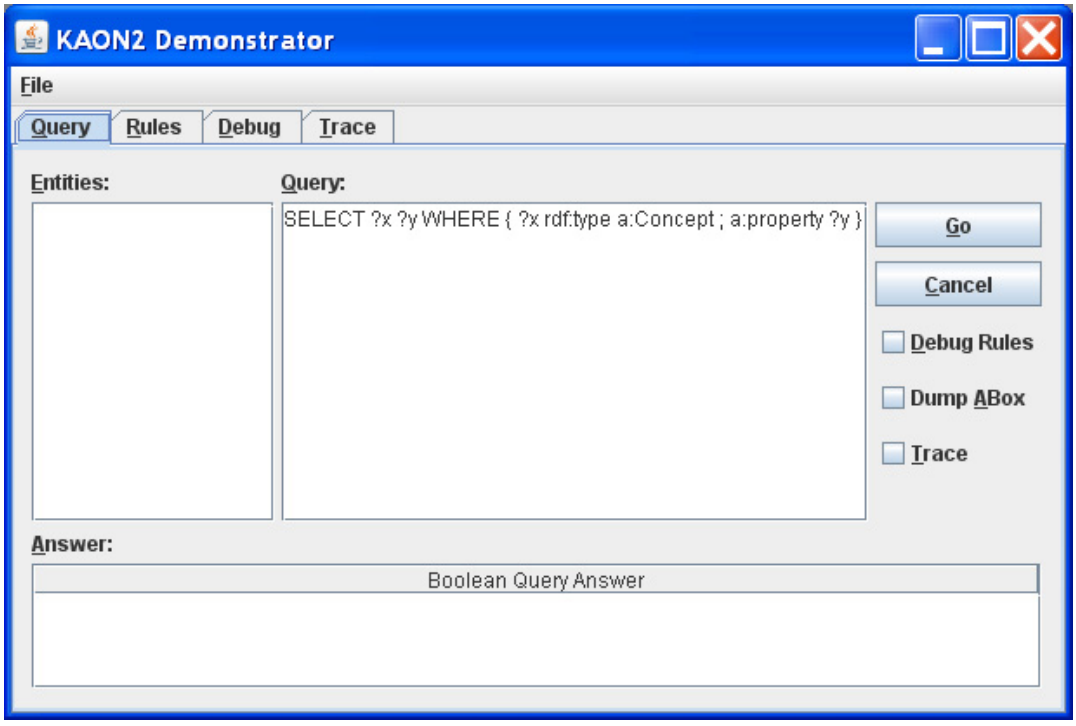

Abbildung 3.1: Kaon2 GUI

Kaon2 ist ein Reasoner, welches in Zusammenarbeit mit der Forschungsgruppe Infor-

mation Process Engineering (IPE[1]) des Forschungszentrums für Informatik in Karlsruhe, dem Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB[2]) der Universität Karlsruhe und der Forschungsgruppe Information Management Group (IMG[3]) der Universität von Manchester entwickelt wurde. Für Forschungszwecke ist Kaon2 kostenlos erhältlich, wobei die kommerzielle Version von der Karlsruher Ontoprise GmbH[4] vertrieben wird. Kaon2 wurde in Java 1.5 implementiert und bietet neben einer GUI (siehe Abbildung 3.1 auf Seite 11), ein DIG Interface[5] für Tools wie Protégé[6] und eine API. Kaon2 kann unteranderem OWL DL Ontologien manupulieren und unterstützt die Schlussfolgerung über OWL Lite und über die SHIQ Untermenge von OWL DL (OWL DL ohne Nominals). Für die komplette Liste der Eigenschaften verweise ich an dieser Stelle auf die Webseite von Kaon2[7].

Im Gegensatz zu vielen anderen Reasonern basiert die Schlussfolgerung im Kaon2 nicht auf dem Tablauverfahren, wobei viele Reasoner auf die Schlussfolgerung über TBoxen optimiert sind. Typische TBox Schlussfolgerung basieren darauf, zu Überprüfen, ob ein Konzept von einem anderen Konzept subsumiert wird (Subsumptions-Check) und zu Überprüfen, ob es ein Modell existiert, in der die Beschreibung nicht auf die leere Menge abgebildet wird (Erfüllbarkeits-Check). Diese beiden Überprüfungen können ineinander überführt werden. Die typische ABox Schlussfolgerung basiert dagegen auf Instanzenüberprüfungen. Üblicherweise wird die ABox Schlussfolgerung auf die TBox Schlussfolgerung reduziert: Um zu Überprüfen, ob eine bestimmte Instanz i einem Konzept D zugehört, muss der Wissensbasis $\neg i \in D$ hinzugefügt und der Erfüllbarkeits-Check durchgeführt werden. Wenn die Wissenbasis unerfüllbar ist, dann folgt daraus, dass $i \in D$ ist. Bei Instanzenabfragen über alle Instanzen einer Klasse muss dieser Check für jedes Individuum der Wissenbasis durchgeführt werden und dies ist sehr aufwendig [BPLF04]. Das Semantischem Web beruht dabei hauptsächlich auf Instanzenüberprüfung von großen ABoxen. Um das Schlussfolgerungsproblem über ABoxen zu optimieren, gibt es dabei verschiedene Ansätze, wie z.B. das auf rationalen Datenbanken basierende Instanzen Lager (engl. instance store) [BHT02]. Auch aktuelle Reasoner wie Racer[8] bieten einige Optimierungen wie dynamische Indizierungen und das Cachen von Anfrageergebnissen und Klassenzugehörigkeitsbeziehungen [HM03, HM04].

Kaon2 geht dabei einen ganz anderen Weg: Er wurde mit Algorithmen implementiert, welche eine SHIQ(D) Wissenbasis in fünf Transformationsschritten in ein negations- und funktionsfreies disjunktives Datalog-Programm umwandeln (bzw. reduzieren). Durch diese Transformation ist es möglich, wohlbekannte deduktive Datenbanktechniken, wie magic-sets oder join-order Optimierungen anzuwenden. Für die Details der Transformation verweise ich auf [HMS04, Mot06]. Eine Übersicht der Kaon2 Architektur ist in Abbildung 3.2 auf Seite 13 dargestellt.

1 http://www.fzi.de/ipe/eng/index.html
2 http://www.aifb.uni-karlsruhe.de/
3 http://img.cs.manchester.ac.uk/
4 http://www.ontoprise.de/
5 standardtisierte XML Schnittstelle für Beschreibungslogische Systeme, http://dig.sourceforge.net/
6 http://protege.stanford.edu/
7 http://kaon2.semanticweb.org/
8 http://www.racer-systems.com/index.phtml

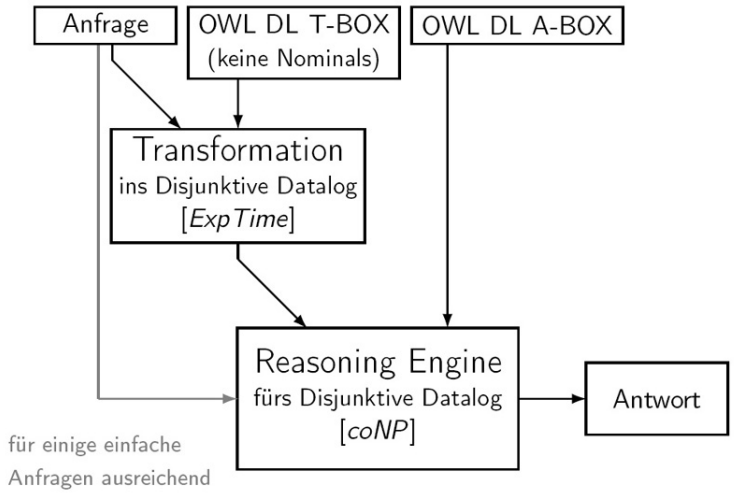

Abbildung 3.2: Kaon2 Architektur

Wie bereits oben beschrieben kann Kaon2 mit SHIQ(D) Ontologien umgehen, wobei dies in etwa OWL DL ohne Nominals entspricht. Die TBox wird zusammen mit der Anfrage in ein disjunktives Datalog-Programm transformiert. Das resultierende Programm wird zusammen mit der ABox in die Reasoning Engine „gefüttert“, welches dann das Ergebnis liefert. Das resultierende Programm der Transformation ist zwar nicht logisch äquivalent mit der eingegebenen TBox, es bestehen dabei aber die im Theorem 3.1.1 beschriebenen und für unsere Absicht ausreichenden Beziehungen.

Theorem 3.1.1 *Sei K eine SHIQ(D) TBox und $D(K)$ das Datalog-Resultat der Transformation von K, so gelten die folgenden Aussagen:*

(a) K ist genau dann unerfüllbar, wenn $D(K)$ unerfüllbar ist.
(b) $K \models \alpha$ genau dann, wenn $D(K) \models \alpha$, wobei α in der Form $A(a)$ oder $R(a,b)$ ist (mit A als genanntes Konzept und R als Rolle (??? Relation, Beziehung ???)).
(c) $K \models C(a)$ für nichtatomares Konzept C genau dann, wenn für Q ein neues atomares Konzept, $D(K \cup \{C \sqsubseteq Q\}) \models Q(a)$.

In einigen Anwendungsfällen, wie der Instanzenabfrage der genannten Klassen (engl. named classes), braucht die Anfrage nicht mit der Tbox transformiert zu werden. Hierzu ist es ausreichend, wenn die Anfrage bei der Schlussfolgerung von der Reasoning Engine berücksichtigt wird. Dadurch kann man die Transformation in das disjunktive Datalog-Programm im vorraus (offline) durchführen, so dass die Reasoning Engine dann nur noch beim Bearbeiten der Anfragen aufgerufen werden muss (vorrausgesetzt die TBox hat sich nach der Transformation nicht geändert) [TRKH08].

3.2 Screech

Screech ist ein approximativer Reasoner, der auf Kaon2 basiert und gehört zu den Kaon2 OWL Tools[1] (Siehe Abbildung 3.3). Kaon2 OWL Tools ist ein Kommandozeilenprogramm und bietet eine Sammlung von kleinen (auf Kaon2 basierenden) Programmen, um OWL Dateien zu bearbeiten. Mehr zu Kaon2 OWL Tools folgt in Kapitel 4.

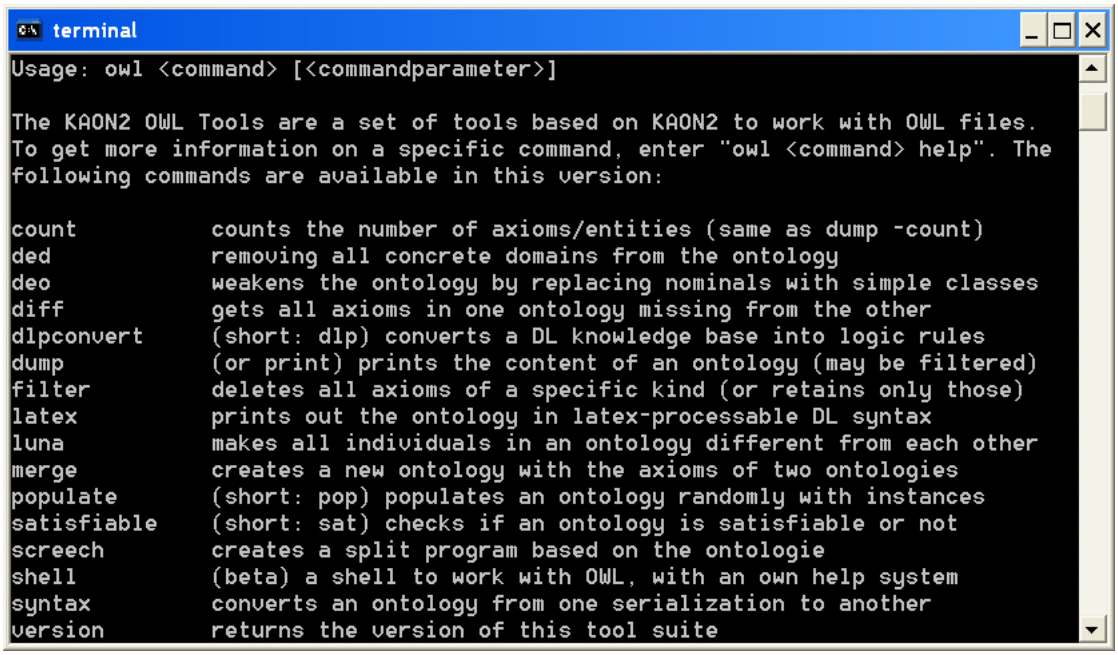

Abbildung 3.3: Kaon2 OWL Tools

Die Architektur von Screech (siehe Abbildung 3.4 auf Seite 15) ist analog zur Architektur von Kaon2. Screech benutzt die API von Kaon2 und verwendet so die Transformationsalgorithmen von Kaon2 um SHIQ(D) Ontologien in disjunktives Datalog-Programm zu transformieren. Danach verwendet Screech seine Approximationsalgorithmen und approximiert das disjunktive Datalog-Programm durch ein sogenanntes nichtdisjunktives Datalog Splittprogramm (engl. split program). Das Spiltprogramm wird dann zusammen mit der ABox in die Reasoning Engine von Kaon2 zugeführt. Auch hierbei gilt wieder, dass bei der Instanzenabfrage der genannten Klassen die Anfrage nicht mit der Tbox transformiert werden muss. Dabei ist es ausreichend, wenn die Anfrage bei der Schlussfolgerung von der Reasoning Engine berücksichtigt wird. Daher kann die Transformation in das nichtdisjunktive Datalog Splittprogramm wieder im vorraus (offline) erfolgen, wodurch die Reasoning Engine dann nur noch aufgerufen werden muss, um die Anfrage zu bearbeiten (vorrausgesetzt die TBox hat sich nach der Transformation nicht geändert). Die, in diesem Kapitel berichteten, Tests (bzw. Instanzenabfragen) wurden auf diese Weise gemacht [TRKH08]. Die Komplexität der Schlussfolgerung sinkt durch die Approximation von [CoNP] auf

1 http://owltools.ontoware.org/

[P]. Mit der Komplexität sinkt dabei auch die Ausdrucksstärke der Modellierungssprache (wird unten genauer beschrieben). Es gibt drei verschiedene Screech-Algorithmen (bzw. Varianten) für die Approximation des disjunktiven Datalog-Programms. In dem folgenden Beispiel werde ich zuerst den ursprüngliche Screech Algorithmus vorstellen (Screech-All). Ursrünglich gab es nur einen Screech Algorithmus, welches nicht zuverlässig aber dafür vollständig war [HV05] - in zwischen gibt es drei verschiedene Schreech Algorithmen [TRKH08]. Ich werde am Ende des Beispiels die Unterschiede zwischen den drei Algorithmen und ihre Eigenschaften beschreiben.

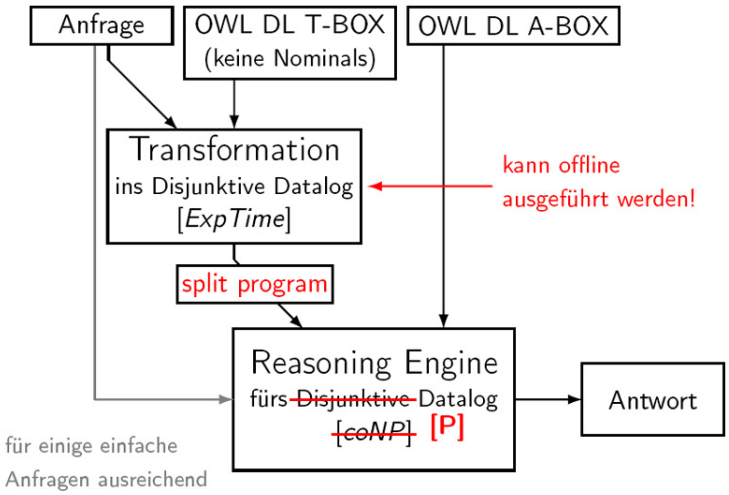

Abbildung 3.4: Screech Architektur

3.2.1 Algorithmen

An dieser Stelle werde ich die Approximationsalgorithmen von Screech an Hand eines einfachen Beispiels erläutern. Das Beispiel basiert auf den Forschungsberichten von Screech [HV05, TRKH08], worauf ich für Einzelheiten referenzieren möchte. Für die Grundlagen von Datalog und den Beschreibungslogiken verweise ich auf Kapitel 2.

Die OWL DL Ontologie in unserem Beispiel besteht aus einer Wissensbasis (Citizen-KB) mit einer TBox (Tabelle 3.1) und einer ABox (Tabelle 3.2 auf Seite 16).

serbian $\sqcup$ croatian $\sqsubseteq$ european
eucitizen $\sqsubseteq$ european
german $\sqcup$ french $\sqcup$ beneluxian $\sqsubseteq$ eucitizen
beneluxian $\equiv$ luxembourgian $\sqcup$ dutch $\sqcup$ belgian

Tabelle 3.1: Citizen-KB (TBox)

Die TBox repräsentiert dabei das terminologische Wissen an Hand von Konzeptdefinitionen und beschreibt z.B., dass EU-Bürger Europäer sind. Die ABox repräsentiert dagegen das Wissen über die Instanzen der Konzepte und beinhaltet Informationen wie z.B., dass Rudi ein Deutscher ist.

serbian(ljiljana)	serbian(nenad)	german(pascal)
french(julien)	croatian(boris)	german(markus)
german(stephan)	croatian(denny)	india(sudhir)
belgian(saartje)	german(rudi)	german(york)

Tabelle 3.2: Citizen-KB (ABox)

In der Tabelle 3.3 sehen wir die Transformation der TBox in das disjunktive Datalog-Programm. Die Regel mit der Disjunktion im Kopf ist hervorgehoben (rot markiert), weil diese Regel im nächsten Schritt durch den Screech Algorithmus verändert wird.

european(x) ← serbian(x)	european(x) ← croatian(x)
european(x) ← eucitizen(x)	
eucitizen(x) ← german(x)	eucitizen(x) ← french(x)
eucitizen(x) ← beneluxian(x)	
beneluxian(x) ← luxembourgian(x)	beneluxian(x) ← dutch(x)
beneluxian(x) ← belgian(x)	
luxembourgian(x) ∨ dutch(x) ∨ belgian ← beneluxian(x)	

Tabelle 3.3: Citizen-KB (disjunctive datalog)

Screech ändert die Ontologie in dem er die Disjunktionen der disjunktiven Regel luxembourgian(x) ∨ dutch(x) ∨ belgian ← beneluxian(x) als Konjunktionen behandelt. Das bedeutet, dass diese Regel durch die folgenden drei Regeln ersetzt wird:

luxembourgian(x) ← beneluxian(x)

dutch(x) ← beneluxian(x)

belgian(x) ← beneluxian(x)

european(x) ← serbian(x)	european(x) ← croatian(x)
european(x) ← eucitizen(x)	
eucitizen(x) ← german(x)	eucitizen(x) ← french(x)
eucitizen(x) ← beneluxian(x)	
beneluxian(x) ← luxembourgian(x)	beneluxian(x) ← dutch(x)
beneluxian(x) ← belgian(x)	
luxembourgian(x) ← beneluxian(x)	dutch(x) ← beneluxian(x)
belgian(x) ← beneluxian(x)	

Tabelle 3.4: Citizen-KB (non-disjunctive datalog)

Das resultierende nichtdisjunktive Splittprogramm ist in der Tabelle 3.4 aufgelistet. Die Änderung beeinfluss die Zuverlässigkeit der Schlussfolgerung. Aus der Information, dass z.B. Saartje ein Bürger der Beneluxstaaten ist, folgte (vorher) dass er entweder ein Belgier, ein Niederländer oder ein Luxemburger ist. Nach dem Approximeren der Regeln folgt jetzt daraus, dass Saartje ein Belgier, ein Niederländer und ein Luxemburger ist:

belgian(saartje) ✔

dutch(saartje) ✘

luxembourgian(saartje) ✘

Dabei ist Saartje nur ein Belgier. Die Schlussfolgerung ist zwar vollstäng, aber nicht mehr zuverlässig: Daher gibt es neben dem richtigen Ergebnis auch falsche Ergebnisse. Die komplexität der Schlussfolgerung hat damit zu Kosten der Zuverlässigkeit abgenommen (von [CoNP] auf [P]). Mit der Komplexität hat aber auch die Ausdrucksstärke der Modellierungssprache abgenommen. Konnte man vorher noch ausdrücken, dass ein Bürger der Beneluxstaaten entweder ein Belgier oder ein Niederländer oder ein Luxemburger ist, ist dies nicht mehr möglich.

Screech-All	Screech-None	Screech-One
Alle Disjunktionen werden als Konjunktionen behandelt	Alle Disjunktionen werden ignoriert	Eine der Disjunktionen wird als eine Konjunktion behandelt.
vollständig unzuverlässig	unvollständig zuverlässig	unvollständig unzuverlässig

Tabelle 3.5: Screech-Varianten

Die Tabelle 3.5 gibt einen Übersicht über die verschiedenen Screech-Algorithmen und ihre Eigenschaften. Was wir bis jetzt betrachtet haben ist das ursprüngliche Screech gewesen, welches neuerdings Screech-All genannt wird. In der Weiterentwicklung von Screech unterscheidet man nämlich zwischen Screech-All, Screech-None und Screech-One. Nachdem ich Screech-All ausführlich beschrieben habe, möchte ich jetzt kurz die Unterschiede erläutern. In Screech-None werden alle disjunktiven Regeln einfach entfernt. Das würde für unser Beispiel bedeuten, dass wir die disjunktive Regel luxembourgian(x) $\vee$ dutch(x) $\vee$ belgian $\leftarrow$ beneluxian(x) entfernen würden. Dadurch wäre die Schlussfolgerung zuverlässig, aber nicht mehr vollständig: Es würden zwar einige Ergebnisse fehlen, dafür wären aber alle erhaltenen Ergebnisse richtig. Die letzte Screech-Variante ist Screech-One. In Screech-One wird jede disjunktive Regel durch genau einen der Splittregeln ersetzt. Für unser Beispiel würde das bedeuten, dass wir die disjunktive Regel luxembourgian(x) $\vee$ dutch(x) $\vee$ belgian $\leftarrow$ beneluxian(x) entweder durch luxembourgian(x) $\leftarrow$ beneluxian(x), dutch(x) $\leftarrow$ beneluxian(x) oder belgian(x) $\leftarrow$ beneluxian(x) ersetzen würden. Die Wahl der Splittregel kann entweder zufällig oder durch statistischem Wissen erfolgen. In unserem Beispiel könnte man das Wissen benutzen, dass es mehr Niederländer als Belgier oder Luxemburger gibt und so die disjunktive Regel durch dutch(x) $\leftarrow$ beneluxian(x) ersetzen. Screech-One ist dabei weder zuverlässig, noch vollständig.

Die parallele Ausführung der Algorithmen ist ebenfalls von Bedeutung. Durch das parallele Ausführen von von Screech-All (vollständig) und Screch-None (zuverlässig) könnte man in einigen Fällen schneller zum exakten Ergebnis kommen als mit dem Kaon2 Reasoner: Stimmen nämlich die Anzahl der Elemente der berechneten Klasseninstanzen unter Screech-All und Screech-None überein, so folgt daraus, dass diese Instanzen richtig berechnet wurden sind. Und für den Fall, dass die berechneten Klasseninstanzen nicht übereinstimmen, kann man die Resultate zum Nachfiltern benutzten: Sei I_s die Menge der Instanzen die zuverlässig sind und I_c die Menge der Insanzen die vollständig sind, so bräuchte man nur die Differenzmenge $I_c \backslash I_s$ in einem weiteren exakten Reasoning-Schritt zu berücksichtigen.

3.2.2 Tests

In diesem Unterkapitel möchte ich kurz einige Test bezüglich der verschiedenen Algorithmen vorstellen. Die Tests stammen aus dem Forschungsbericht von Screech und werden von mir hier nur in gekürzter Form dargestellt, um einen kleinen praktischen Vergleich der Algorithmen zu liefern. Für die ausführlichen Tests referenziere ich auf den Forschungsbericht [TRKH08].

Galen

Galen[1] besteht nur aus einer TBox und verfügt über 175 Klassen. Da Galen eine reine TBox Ontologie ist, werden die Klassen zufällig mit 500 Individuen bestückt. Galen hat 673 Axiome und durch die Bestückung kommen noch 500 weitere dazu. Nach der Tansformation in das disjunktive Datalog-Programm gibt es ca. 1800 Regeln, wovon ca. 60 Regeln mit Disjunktionen sind. Die exakte Anzahl der Regeln unterscheidet sich leicht bei den verschiedenen Durchläufen, da die Kaon2 Translationsalgorithmen nichtdeterministisch sind. Nach dem Approximieren der disjunktiven Regeln entstehen 113 Splittregeln. Außerdem sind in den 1800 Regeln 149 Integritätsbedingung enthalten, die entfernt werden. Integritätsbedingungen sind Konsistenzbedingung für Datenbanken und stellen sicher, dass die Integrität eingehalten wird.

In 137 von 175 Klassen (78%) stimmen die Anzahl der Elemente der berechneten Klasseninstanzen unter Screech-All und Screech-None überein. Daraus folgt (ohne die disjunktiven Regeln zu benutzen), dass diese Instanzen richtig berechnet wurden sind.

Die Tabelle 3.6 zeigt das Resultat der Instanzenabfrage über die genannten Klassen der Galen Ontologie und stellt einen Vergleich der Screech-Varianten dar. Dabei gibt die Spalte „fehlen“ an, wie viele Klasseninstanzen nicht gefunden wurden sind. Die Spalte „korrekt“ gibt dagegen an, wie viele Instanzen korrekterweise gefunden wurden und „mehr“ zeigt an, wie viele Instanzen fälschlicherweise berechnet wurden sind. Die Spalten „Zeit“ und „Kaon2“ liefern einen Vergleich der Laufzeiten zwischen den jeweiligen Screech-Varianten und Kaon2.

GALEN					
Variante	**fehlen**	**korrekt**	**mehr**	**Zeit**	**Kaon2**
Screech-All	0	5187	465	513546ms	1562898ms
Screech-One	5	5182	134	569658ms	1562898ms
Screech-None	10	5177	0	366711ms	1562898ms

Tabelle 3.6: Instanzenabfrage über die genannten Klassen von Galen

1 http://www.cs.man.ac.uk/~rector/ontologies/simple-top-bio/

Dolce

Dolce[1] ist ebenfalls eine reine Tbox Ontologie. Da die komplette Ontologie die Fähigkeiten aktueller Reasoner übersteigt, wird nur eine Fraktion der Ontologie mit 1552 Axiomen benutzt. Dabei werden die 123 Klassen von Dolce zufällig mit 494 Individuen bestückt. Nach der Tansformation ins disjunktive Datalog gibt es ca. 1780 Regeln, wovon ca. 72 Regeln mit Disjunktionen sind. Desweiteren sind ca. 200 Integritätsbedingung vorhanden, die entfernt werden. Es entstehen nach dem Approximieren der disjunktiven Regeln 176 Splittregeln.

In 93 von 123 Klassen (76%) stimmen die Anzahl der Elemente der berechneten Klasseninstanzen unter Screech-All und Screech-None überein. Daraus folgt wieder (ohne die disjunktiven Regeln zu benutzen), dass diese Instanzen richtig berechnet wurden sind

In der Tabelle 3.7 sehen wir das Resultat der Instanzenabfrage über die genannten Klassen. Aus der Tatsache, dass Screech-None keine Fehler macht erkennen wir, dass das disjunktive Wissen keinen Beitrag zum Ergebnis leistet.

DOLCE					
Variante	**fehlen**	**korrekt**	**mehr**	**Zeit**	**Kaon2**
Screech-All	0	3697	2256	472941ms	516064ms
Screech-One	0	3697	512	425748ms	516064ms
Screech-None	0	3697	0	397260ms	516064ms

Tabelle 3.7: Instanzenabfrage über die genannten Klassen von Dolce

Wine

Wine[2] ist eine Ontologie mit einer ABox und einer TBox mit 140 Klassen. Dabei enthält Wine auch Nominals, welche auf eine approximierende Weise entfernt werden. Die daraus resultierende Ontologie hat 20762 Axiome, wobei die korrespondierende ABox 6601 Axiome hat. Es entstehen ca. 550 Regeln nach der Transformation in das disjunktive Datalog-Programm. Von denen sind 24 Regeln mit Disjunktionen und werden durch 48 Splittregeln approximiert. Ausserdem werden 3 Integritätsbedingung entfernt.

In 116 von 140 Klassen (83%) stimmen die Anzahl der Elemente der berechneten Klasseninstanzen unter Screech-All und Screech-None überein. Daraus läßt sich wieder schließen (ohne die disjunktiven Regeln zu benutzen), dass diese Instanzen richtig berechnet wurden sind.

Die Tabelle 3.8 auf der Seite 21 repräsentiert das Resultat der Instanzenabfrage über die genannten Klassen von Wine.

1 http://www.loa-cnr.it/DOLCE.html
2 http://www.schemaweb.info/schema/SchemaDetails.aspx?id=62

WINE					
Variante	**fehlen**	**korrekt**	**mehr**	**Zeit**	**Kaon2**
Screech-All	0	30627	1353	588562ms	707476ms
Screech-One	0	30627	615	494456ms	707476ms
Screech-None	697	29930	0	504914ms	707476ms

Tabelle 3.8: Instanzenabfrage über die genannten Klassen von Wine

Semintec

Semintec[1] besteht aus einer ABox mit 130702 Axiomen und einer TBox mit 59 Klassen und 35882 Axiomen. Nach der Transformation in das disjunktive Datalog-Programm entstehen 217 Regeln (alle ohne Disjunktionen). Davon sind ca. 113 Regeln Integritätsbedingung, die entfernt werden. Das Resultat der Instanzenabfrage kann man aus der Tabelle 3.9 entnehmen. Es stimmen alle drei Screech-Varianten überein, da bei der Transformation keine disjunktiven Regeln entstanden sind. Dabei wirkt sich das Entfernen der Integritätsbedingung positiv auf das Laufzeitverhalten aus.

SEMINTEC					
Variante	**fehlen**	**korrekt**	**mehr**	**Zeit**	**Kaon2**
Screech-All	0	51184	0	41796ms	86045ms
Screech-One	0	51184	0	41796ms	86045ms
Screech-None	0	51184	0	41796ms	86045ms

Tabelle 3.9: Instanzenabfrage über die genannten Klassen von Semintec

1 http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm

KAPITEL 4

Experiment

In diesem Kapitel werde ich die die Funktionsweise von Screech, mittels eines selbstimplementierten Java-Programms, experimentell demonstrieren. Um die Demonstration übersichtlich und kurz zu halten, werde ich dabei ein Fragment der Wissensbasis (Citizen-KB) aus dem Kapitel 3.2.1 verwenden. Das Fragment ist in der Tabelle 4.1 dargestellt und enthält eine ABox mit einer Instanz- und eine TBox mit einer Konzeptdefinition.

ABox:	belgian(saartje)
TBox:	beneluxian $\equiv$ luxembourgian $\sqcup$ dutch $\sqcup$ belgian

Tabelle 4.1: Fragtion aus der Wissensbasis Citizen-KB

Das Listing 4.1 zeigt die Ontologie in der owlx[1] Repräsentation, wobei owlx eine XML Präsentationssyntax für OWL ist. Ich habe owlx benutzt, da es von Kaon2 und von somit von Screech unterstützt wird und leicht zu lesen ist.

Listing 4.1: citizen.owl

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!DOCTYPE Ontology [
3      <!ENTITY ex "http://logic.guerkan.de/screech/citizen#">
4  ]>
5  <owlx:Ontology owlx:name="&ex;"
6    xmlns:owlx="http://www.w3.org/2003/05/owl-xml#">
7
8    <owlx:Class owlx:name="#beneluxian" owlx:complete="true">
9      <owlx:UnionOf>
10       <owlx:Class owlx:name="#luxembourgian"/>
11       <owlx:Class owlx:name="#dutch"/>
12       <owlx:Class owlx:name="#belgian"/>
13     </owlx:UnionOf>
14   </owlx:Class>
15
```

1 http://www.w3.org/TR/2003/NOTE-owl-xmlsyntax-20030611/

```
16    <owlx:Individual owlx:name="#saartje">
17      <owlx:type owlx:name="#belgian" />
18    </owlx:Individual>
19
20  </owlx:Ontology>
```

Ich werde zuerst das Screech aus der OWL Tools Programmsammlung erläutern und beschreiben auf welches Problem ich dabei gestoßen bin. Im Anschluss werde ich im Unterkapitel 4.2 meine eigene Implementierung vorstellen.

Es ist wichtig, bei dem OWL Tools Screech und bei meiner eigenen Implementierung, die Programme zuerst von der CD-ROM auf die Festplatte zu kopieren, da die Ergebnisse der Anfragen direkt in dem Verzeichnis des jeweiligen Programms gespeichert werden.

4.1 OWL Tools Screech

Das Screech aus dem OWL Tools Kommandozeilenprogramm implementiert den Screech-All Algorithmus und bietet einige Transformationsoptionen zur Auswahl. Die Übersicht der Optionen bzw. Optionsblöcken ist in der Abbildung 4.1 dargestellt. Die von einer Leerzeile getrennten Blöcke sind Optionsblöcke aus denen jeweils ein Optionsparameter gewählt werden kann.

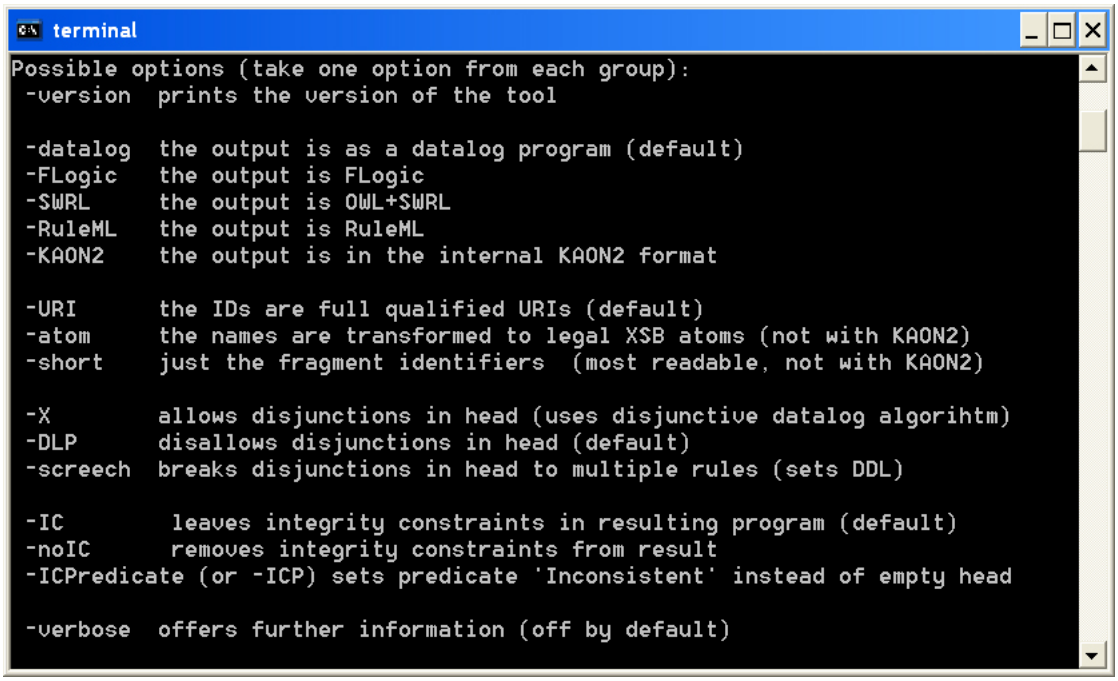

Abbildung 4.1: Kaon2 OWL Tools Screech

Unteranderem kann man dabei das Ausgabeformat auswählen und festlegen, ob man das disjunktive Datalog-Programm (-X) oder das nichtdisjunktive Splittprogramm (-screech) erhalten möchte. Desweiteren kann auch festgelegt werden, ob z.B. verkürzte Bezeichner

verwendet werden sollen, die zwar einfacher zu lesen sind, aber nicht von Kaon2 unterstützt werden und ob Integritätsbedingung bei der Transformation eliminiert werden sollen oder nicht.

Screech wird beispielsweise im Verzeichnis „c:\owltools-2006-06-07\owltools>“ über die Kommandozeile mit dem folgenden Befehl ausgeführt: „owl dlp screech“ gefolgt von einer Ontologie und höchsten einem Optionsparameter aus jedem Optionsblock (siehe Abbildung 4.1 auf Seite 23). Die Eingabeparameter müssen jeweils mit einem Leerzeichen abgegrenzt werden, wobei die Ontologie inklusive der Endung eingegeben werden muss. Wählt man keine Optionen aus, so werden die Standartwerte genommen. Möchte man die Ausgabe in einer Datei speichern, so muss man nach dem Ontologienamen einen weiteren Dateinamen (mit Endung) für die Ausgabe eingeben (vor den Optionsparametern).

In unserem konkreten Beispiel würden wir mit dem Befehl „owl dlp citizen.owl -screech“ die Ausgabe auf der Abbildung 4.2 erhalten, wobei die Notation „:-“ als „←“ gelesen wird.

```
beneluxian(X) :- belgian(X).
belgian(saartje).
beneluxian(X) :- dutch(X).
belgian(X) :- beneluxian(X).
dutch(X) :- beneluxian(X).
luxembourgian(X) :- beneluxian(X).
beneluxian(X) :- luxembourgian(X).
```

Abbildung 4.2: Ausgabe als Datalog (Splittprogramm)

Das Splittprogramm wurde mit dem Screech-All Algorithmus erzeugt und entspricht unseren Erwartungen (siehe 3.2.1). Um über die Kaon2 API auf die Ausgabe zugreifen und Instanzenabfragen durchführen zu können, braucht man ein Format, welches von der API unterstützt wird. Deshalb habe ich im nächsten Schritt „-SWRL“ als Optionsparameter gesetzt, da SWRL[1] Ontologien von der Kaon2 API unterstützt werden. Weil SWRL eine umfangreiche Regelsprache ist und weil sie an sich keine primäre Rolle in meiner Arbeit einnimmt, möchte ich für Einzelheiten auf die Webseite von W3C verweisen (siehe Fußnote). Unteranderem kann SWRL die Datalog-Regeln im XML Syntax repräsentieren. Dabei wird z.B. im SWRL Kopg-Tag der Kopf der Datalog-Regel und im SWRL Rumpf-Tag der Rumpf der Datalog-Regel beschrieben. Als ein Beispiel möchte ich in dem Listing 4.2 einen Auszug aus der Ausgabe präsentieren, welches das Befehl „owl dlp citizen.owl ausgabe.owl -SWRL -screech“ erzeugt.

Listing 4.2: Auszug aus ausgabe.owl (mit -SWRL)

```
1 <swrl:Imp>
2     <swrl:disjunctive rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">true</
          swrl:disjunctive>
3     <swrl:head rdf:parseType="Collection">
4         <swrl:ClassAtom>
5             <swrl:classPredicate rdf:resource="&a;belgian"/>
6             <swrl:argument1 rdf:resource="#X"/>
```

1 http://www.w3.org/Submission/SWRL/

```
7           </swrl:ClassAtom>
8           <swrl:ClassAtom>
9               <swrl:classPredicate  rdf:resource="&a;luxembourgian"/>
10              <swrl:argument1 rdf:resource="#X"/>
11          </swrl:ClassAtom>
12          <swrl:ClassAtom>
13              <swrl:classPredicate  rdf:resource="&a;dutch"/>
14              <swrl:argument1 rdf:resource="#X"/>
15          </swrl:ClassAtom>
16      </swrl:head>
17      <swrl:body rdf:parseType="Collection">
18          <swrl:ClassAtom>
19              <swrl:classPredicate  rdf:resource="&a;beneluxian"/>
20              <swrl:argument1 rdf:resource="#X"/>
21          </swrl:ClassAtom>
22      </swrl:body>
23  </swrl:Imp>
24
25  <owl:Class rdf:ID="belgian"/>
26
27  <owl:Class rdf:ID="beneluxian"/>
28
29  <owl:Class rdf:ID="dutch"/>
30
31  <owl:Class rdf:ID="luxembourgian"/>
```

An dem „swrl:disjunctive“-Tag und den im Kopf-Tag (swrl:head) aufgelisteten Klassen sehen wir, dass der Kopf der Regel Disjunktionen hat. Dieser Auszug entspricht der folgenden Datalog-Regel:

$$\text{luxembourgian}(x) \vee \text{dutch}(x) \vee \text{belgian} \leftarrow \text{beneluxian}(x)$$

Daher ist diese Ausgabe falsch, denn sie sollte eigentlich nur die SWRL Version des, in der Abbildung 4.2 auf Seite 24 gezeigten, Splittprogramms sein und dürfte keine Disjunktionen erhalten. Dadurch ist die SWRL Ausgabe von Screech fehlerhaft ist und es macht keinen Sinn mit diesen Daten weiterzuarbeiten. Deshalb habe ich mein eigenes Java Programm geschrieben und diesen Fehler behoben, wobei ich unteranderem den Screech-All Algorithmus aus dem OWL Tools Quellcode verwendet und in meine Implementierung eingebaut habe.

4.2 Eigene Implementierung

Ich habe mein Programm ebenfalls in Java geschrieben. Zuerst liest es die Ontologie und wandelt es intern in SWRL um. Danach wendet es den Screech-All Algorithmus an und führt schliesslich an Hand des Splittprogramms die Instanzenabfrage aller genannten Klassen durch. Das Ergebnis der Anfrage wird einerseits auf der Kommandozeile angezeigt und andererseits in einer XML Datei abgespeichert. Alle diese Schritte werden automatisch nacheinander ausgeführt, wenn man das Programm startet. Das Programm startet man über die Kommandozeile mit dem folgenden Befehl: „screech“ gefolgt von der Ontologie (mit Endung), die man laden möchte. In unserem komkreten Beispiel würde wir das

Programm mit dem Befehl „screech citizen.owl“ ausführen. Auf der Abbildung 4.3 sehen wir die Ausgabe auf der Kommandozeile.

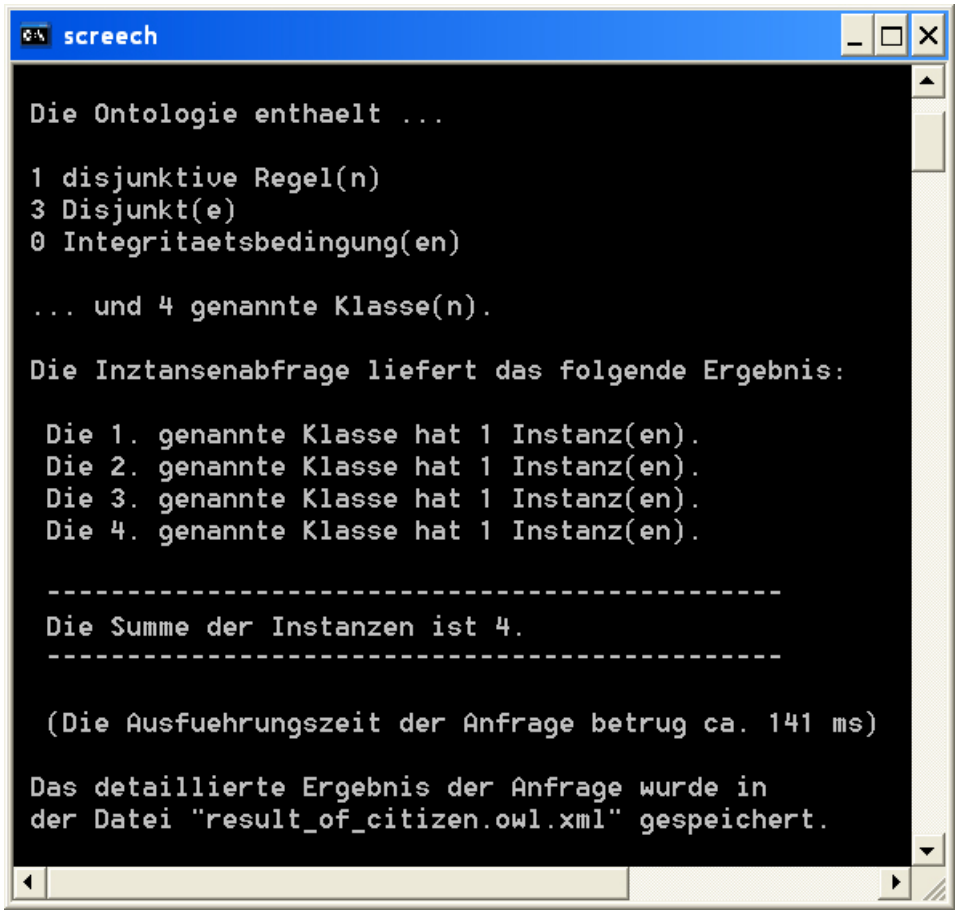

Abbildung 4.3: Ausgabe auf der Kommandozeile

Es wird auf der Kommandozeile angezeigt, wieviele disjunktive Regeln, Disjunkte und Integritätsbedingung die geladene Ontologie hat. Desweiteren wird angezeigt, wieviele genannte Klassen die Ontologie enthält und wieviele Instanzen diese Klassen jeweils haben. Als letztes wird die Summe die Instanzen aller genannten Klassen und die Bearbeitungszeit der Anfrage ausgegeben. In dem Listing 4.3 ist die XML Datei dargestellt, in der das Ergebnis gespeichert wird.

Listing 4.3: result_of_citizen.owl.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <genannte_Klassen Anzahl="4">
3   <Klasse Name="#beneluxian" Anzahl_Instanzen="1">
4     <Instanz Name="#saartje" />
5   </Klasse>
6   <Klasse Name="#luxembourgian" Anzahl_Instanzen="1">
7     <Instanz Name="#saartje" />
8   </Klasse>
```

```
9    <Klasse Name="#dutch" Anzahl_Instanzen="1">
10     <Instanz Name="#saartje" />
11   </Klasse>
12   <Klasse Name="#belgian" Anzahl_Instanzen="1">
13     <Instanz Name="#saartje" />
14   </Klasse>
15 </genannte_Klassen>
```

In der XML Datei wird nochmals angezeigt, wieviele genannte Klassen die Ontologie hat und wieviele Instanzen diese Klassen haben. Im gegensatz zur Kommandozeile wird hierbei auch ausgegeben, wie die Klassen und die Instanzen heißen.

Dieses Ergebnis entspricht unseren Erwartungen (siehe 3.2.1) und demonstriert, dass der Screech-All Algorithmus richtig angewendet wurde. Im Kapitel 3.2.1 hatten wir festgestellt, dass man nach dem Approximeren der Regeln mit dem Screech-All Algorithmus fälschlicherweise folgert, dass Saartje ein Belgier, ein Niederländer und ein Luxemburger ist, was hierbei ebenfalls der Fall ist.

Wir hatten in unserer Wissenbasis aus der Tabelle 4.1 auf Seite 22 die Information, dass Saartje ein Belgier ist und in dem Splittprogramm, welches in der Abbildung 4.2 auf Seite 24 dargestellt ist, gibt es unteranderem die folgenden Splittregeln:

$$\text{beneluxian(x)} \leftarrow \text{belgian(x)}$$

$$\text{luxembourgian(x)} \leftarrow \text{beneluxian(x)}$$

$$\text{dutch(x)} \leftarrow \text{beneluxian(x)}$$

$$\text{belgian(x)} \leftarrow \text{beneluxian(x)}$$

Aus dem Wissen, dass Saartje ein Belgier ist, folgt somit, dass er ein Bürger der Beneluxstaaten ist. Daraus folgt dann wiederum, dass er ein Belgier, ein Niederländer und ein Luxemburger ist. Aus diesem Grund sehen wir in dem Listing 4.3 auf Seite 26, dass alle genannten Klassen eine Instanz „Saartje“ haben.

Ich glaube, dass die Funktionsweise von Screech und das Beispiel aus dem Kapitel 3.2.1 durch dieses praktische Experiment deutlicher geworden sind.

KAPITEL 5

Zusammenfassung und Ausblick

5.1 Zusammenfassung

Im Rahmen dieser Projektarbeit wurden, am Beispiel des approximativen Reasoners Screech, mögliche Approximierungsansätze analysiert. Dazu wurden als ein wichtiges Anwendungsgebiet das Semantische Web vorgestellt und die, ihr zugrunde liegenden, Modellierungsparadigmen beschrieben. Darauf aufbauend wurden dann die beiden Reaosoner Kaon2 und Screech vorgestellt und die verschiedenen Screech-Algorithmen erläutert. Dabei wurde verdeutlicht, dass die hohe Ausdrucksstärke der Modellierungssprache zu einer hohen Komplexität bei der Schlussfolgerung führt und dass man auf Kosten der Zuverlässigkeit und/oder der Vollständigkeit die Komplexität sinken bzw. die Schlussfolgerungsgeschwindigkeit steigern kann. Abschliessend wurde die Funktionsweise von Screech, mittels eines selbstimplementierten Java-Programms, experimentell demonstriert.

Zusammenfassend möchte ich an dieser Stelle kurz meine Erfahrungen schildern und erwähnen, dass ich einiges an Zeit für die Benutzung der Reasonern und der Programmierschnittstelle gebraucht habe, da es keine ausführlichen Anleitungen gibt. Es existieren lediglich einige Beispiele, aus denen man dann zu lernen versucht. Es hat mich ebenfalls einiges an Verzweiflung und Zeit gekostet, bis ich dahinter gekommen bin, dass es einen Fehler bei der SWRL Ausgabe gibt. Ich hatte mir nämlich immer die leicht zu lesende Datalog ausgabe angeguckt und dann die Anfragen mit der SWRL Ausgabe durchgeführt. Aber dennoch, im Großen und Ganzen, habe ich vieles dazu gelernt und es hat mir Freude bereitet ein aktuelles, relevantes und anspruchvolles Thema bearbeitet zu haben.

5.2 Ausblick

Es wurde in Kapitel 3.2.1 verdeutlicht, dass die Kombination der verschiedenen Screech-Algorithmen von Bedeutung ist. Ich denke, dass sich die Screech-Algorithmen nicht nur untereinander gut kombinieren lassen, sondern auch mit anderen Algorithmen sinnvoll kombinierbar sind. So wird z.B. in dem Forschungsbericht [KMWW08] unteranderem ein Approximierungsansatz beschrieben, in dem SHI (Beschreibungslogik) TBoxen durch DL-Lite TBoxen approximiert werden. Die Approximation ist zwar unzuverlässig, aber dafür vollständig. Es wird dabei eine SHI TBox durch mehrere DL-Lite TBoxen approximiert. Daher gibt es mehrere Approximationen einer TBox, die zwar alle vollständig, aber

unterschiedlich gut bzw. genau sind. Der Screech-All Algorithmus, welches ebenfalls vollständig ist, könnte hierbei mit diesem Ansatz kombiniert und dazu benutzt werden, um die beste DL-Lite TBox Approximation zu identifizieren.

Die Abbildung 5.1 zeigt die Architektur dieser Kombination. Im ersten schritt werden die DL-Lite Approximationen der SHI TBox erzeugt. Danach folgt die Transformation ins disjunktive Datalog und die anschließende Approximation durch den Screech-All Algorithmus. Die dabei entstehenden Splittprogramme werden jeweils mit der ABox zusammen in die Reasonin Engine von Kaon2 „gefüttert“. Für jede dieser Approximationen wird eine komplette Inztansenabfrage durchgeführt und eine Antwort erzeugt. An Hand dieser Antworten kann die beste Approximation auf folgende Weise identifiziert werden. Da die Schreech-All Approximation vollständig ist, ist die Antwort mit der kleinsten Inztansenanzahl über alle Klassen (allgemein betrachtet) die beste Lösung. Spezieller kann man auch eine bestimmte Klasse in allen Antworten betrachten und überprüfen welche die wenigsten Instanzen hat. So kann man feststellen welche Approximation die beste bezüglich einer bestimmten Klasse ist.

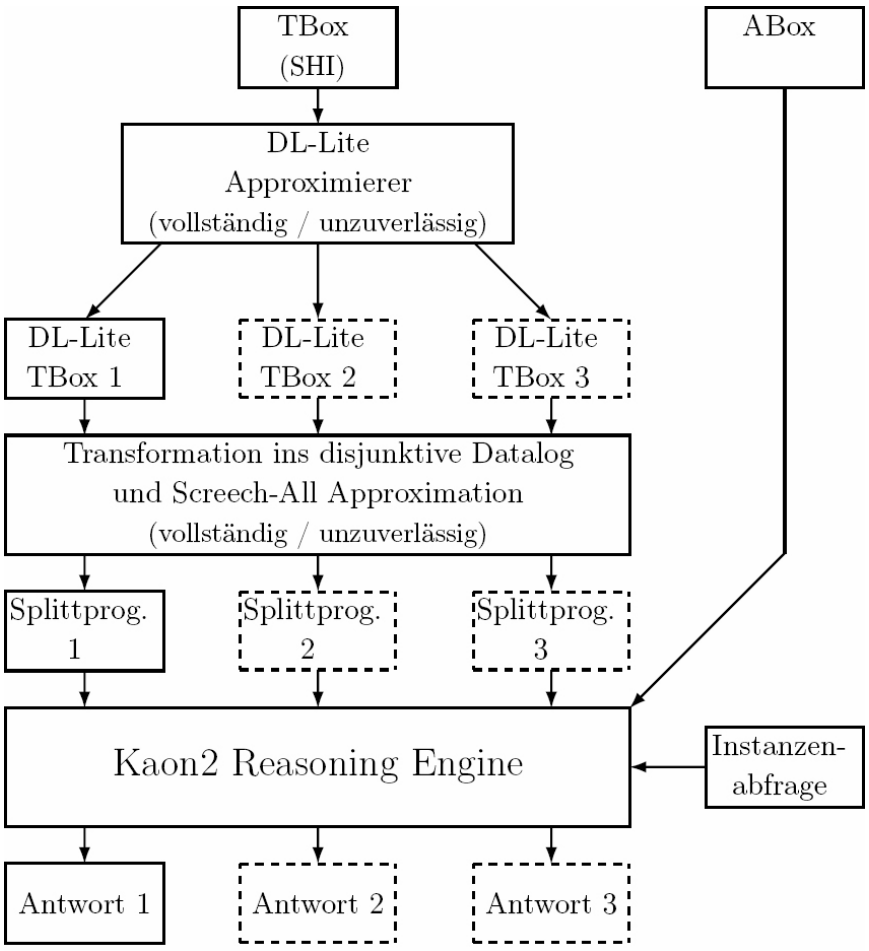

Abbildung 5.1: Screech in Kombination mit dem DL-Lite Approximierer

Mit diesem Beispiel wollte ich verdeutlichen, dass sich Screech auch mit anderen Approximationsansätzen kombinieren ließe und somit auch für zukünftige Forschungen in diesem Feld von Bedeutung wäre. Für die Eigenschaften der Beschreibungslogik SHI, die ausführliche Beschreibung der Approximation der SHI TBoxen und die genaue Funktionsweise des gesamten Ansatzes referenziere ich den Interessenten auf den Forschungsbericht [KMWW08].

Literaturverzeichnis

[BCM03] Baader, Franz ; Calvanese, Diego ; McGuinness, Deborah: *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003. – 574 S. – ISBN 0521781760 (Zitiert auf Seite 4)

[BHT02] Bechhofer, S. ; Horrocks, I. ; Turi, D.: Instance store - database support for reasoning over individuals / http://instancestore.man.ac.uk/instancestore.pdf. 2002. – Forschungsbericht (Zitiert auf Seite 12)

[BLHL05] Berners-Lee, Tim ; Hendler, James ; Lassila, Ora: The semantic web. In: *Scientific American* (2005) (Zitiert auf Seite 3)

[BLPF05] Bruijn, Jos de ; Lara, Rubén ; Polleres, Axel ; Fensel, Dieter: OWL DL vs. OWL Flight: Conceptual Modeling and Reasoning for the Semantic Web / International World Wide Web Conference Committee (IW3C2). 2005. – Forschungsbericht (Zitiert auf Seiten 4, 8 und 9)

[BPLF04] Bruijn, J. de ; Polleres, A. ; Lara, R. ; Fensel, D.: OWL Flight, Deliverable D20.3v0.1, WSML / http://www.wsmo.org/TR/d20/d20.3/v0.1/. 2004. – Forschungsbericht (Zitiert auf Seite 12)

[HM03] Haarslev, V. ; Möller, R.: Incremental query answering for implementing document retrieval services / In Proceedings of the International Workshop on Description Logics (DL-2003), pages 85-94, Rome, Italy. 2003. – Forschungsbericht (Zitiert auf Seite 12)

[HM04] Haarslev, V. ; Möller, R.: Optimization techniques for retrieving resources described in OWL/RDF documents: First results / In Ninth International Conference on the Principles of Knowledge Representation and Reasoning (KR2004), pages 163173, Whistler, BC, Canada. 2004. – Forschungsbericht (Zitiert auf Seite 12)

[HMS04] Hustadt, Ullrich ; Motik, Boris ; Sattler, Ulrike: Reducing $SHIQ^-$ description logic to disjunktive datalog programs / In Principles of Knowledge Representation and Reasoning: Proceedings of the 9th Int. Conf. (KR2004). 2004. – Forschungsbericht (Zitiert auf Seiten 6 und 12)

[HV05] Hitzler, Pascal ; Vrandečić, Denny: Resolution-Based Approximate Reasoning for OWL DL / AIFB, University of Karlsruhe, Germany. 2005. – Forschungsbericht (Zitiert auf Seiten 1 und 15)

[KMWW08] Kaplunova, Alissa ; Möller, Ralf ; Wandelt, Sebastian ; Wessel, Michael: Approximation and ABox Segmentation / Institute for Software Systems (STS), Hamburg University of Technology. Germany, 2008. – Forschungsbericht. – Siehe http://www.sts.tu-harburg.de/tech-reports/papers.html. (Zitiert auf Seiten 28 und 30)

[Mot06] Motik, B.: *Reasoning in Description Logics using Resolution and Deductive Databases.*, PhD Thesis, University of Karlsruhe, Karlsruhe, Germany, Diss., 2006 (Zitiert auf Seite 12)

[PH06] PatelSchneider ; Horrocks, Ian: Position Paper: A Comparison of Two Modelling Paradigms in the Semantic Web / Bell Labs Research Lucent Technologies und School of Computer Science University of Manchester. 2006. – Forschungsbericht (Zitiert auf Seiten 3, 4 und 6)

[TRKH08] Tserendorj, Tuvshintur ; Rudolph, Sebastian ; Krötzsch, Markus ; Hitzler, Pascal: Approximate OWL-Reasoning with SCREECH / FZI Karlsruhe, Germany und AIFB, University of Karlsruhe, Germany. 2008. – Forschungsbericht (Zitiert auf Seiten 13, 14, 15 und 19)

[Wik] Wikipedia: *Beschreibungslogik.* – http://de.wikipedia.org/wiki/Beschreibungslogik Abruf: 28/01/2009 (Zitiert auf Seite 4)

Abbildungsverzeichnis

3.1 Kaon2 GUI 11
3.2 Kaon2 Architektur 13
3.3 Kaon2 OWL Tools 14
3.4 Screech Architektur 15

4.1 Kaon2 OWL Tools Screech 23
4.2 Ausgabe als Datalog (Splittprogramm) 24
4.3 Ausgabe auf der Kommandozeile 26

5.1 Screech in Kombination mit dem DL-Lite Approximierer 29

A.1 Verzeichnisstruktur der CD-ROM 37

Tabellenverzeichnis

3.1 Citizen-KB (TBox) 15
3.2 Citizen-KB (ABox) 16
3.3 Citizen-KB (disjunctive datalog) 16
3.4 Citizen-KB (non-disjunctive datalog) 17
3.5 Screech-Varianten 18
3.6 Instanzenabfrage über die genannten Klassen von Galen 19
3.7 Instanzenabfrage über die genannten Klassen von Dolce 20
3.8 Instanzenabfrage über die genannten Klassen von Wine 21
3.9 Instanzenabfrage über die genannten Klassen von Semintec 21

4.1 Fragtion aus der Wissensbasis Citizen-KB 22

Listings

4.1 citizen.owl 22
4.2 Auszug aus ausgabe.owl (mit -SWRL) 24
4.3 result_of_citizen.owl.xml 26

Abkürzungsverzeichnis

ABox	Assertional Box
API	Application Programming Interface
bzw.	beziehungsweise
ca.	circa
DIG	DL Implementation Group
GUI	Graphical User Interface
OWL	Web Ontology Language
PDF	Portable Document Format
TBox	Terminology Box
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XML	Extensible Markup Language
z.B.	zum Beispiel

ANHANG A

CD-ROM

A.1 Inhalt

Dieser Projektarbeit liegt eine CD-ROM bei. Auf ihr befindet sich die digitale (PDF) Version dieses Dokuments. Desweiteren sind auf der CD-ROM die, in dieser Projektarbeit verwendeten, Dateien und Programme enthalten. Die Verzeichnisstruktur der CD-ROM ist in der Abbildung A.1 dargestellt. Der Übersichtlichkeitshalber habe ich auf der Abbildung nicht alle Ordner geöffnet dargestellt.

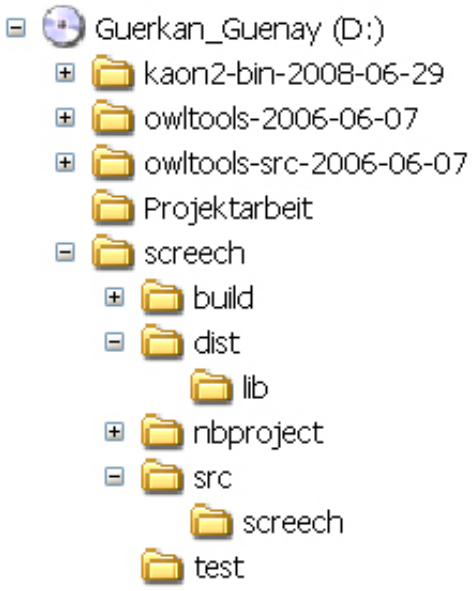

Abbildung A.1: Verzeichnisstruktur der CD-ROM

Die digitale Version der Projektarbeit befindet sich in dem Ordner „Projektarbeit“

In dem Unterordner „dist“ (\screech\dist) befindet sich meine Implementierung. Hier befinden sich auch die citizen Ontologie und das von dem Reasoner gelieferte (XML) Ergebnis. Die benötigten Java Archive (.jar) befinden sich dagegen in dem lib Ordner (\screech\dist\lib), wobei der Quellcode in dem „screech“ Unterordner von „src“ ist

(\screech\src\screech).

Die aktuellen Versionen von Kaon2 und der OWL Tools Programmsammlung befinden sich in den Ordnern „kaon2-bin-2008-06-29“ und „owltools-2006-06-07“. Vollständiskeitshalber habe ich ebenfalls den Quellcode von OWL Tools („owltools-src-2006-06-07“) auf die CD-ROM gebrannt.