Lernen von Multimedia-Interpretationsregeln

Diplomarbeit

Gerret Brandt

Püfer
Prof. Dr. Ralf Möller

Technische Universität Hamburg-Harburg
Institut für Softwaresysteme

Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit selbstständig, ohne fremde Hilfe und nur unter Benutzung der angegebenen Quellen angefertigt zu haben. Die Arbeit ist in gleicher oder ähnlicher Form noch nicht als Prüfungsarbeit eingereicht worden. Alle Aus führungen der Arbeit, die wörtlich oder sinngemäß übernommen wurden, sind als solche kenntlich gemacht.

Hamburg, 17 August 2009
Gerret Brandt

Inhaltsverzeichnis

Kapitel 1

Einleitung

Multimedia-Dokumente - seien es Bilder, Texte, Audio-Clips oder Filme - enthalten eine Vielzahl von impliziten Informationen, die automatisierten Information-Retrieval-Systemen zugänglich gemacht werden sollen. In intelligenten Retrieval-Systemen werden semantische Annotationen zu den Multimedia-Dokumenten in einer Wissensbasis abgelegt. Mit den Methoden der Beschreibungslogik können explizite und implizite Informationen aus diesen Wissensbasen abgerufen werden. Deswegen wird für ein effektives Information-Retrieval die Extraktion von Informationen aus Multimedia-Dokumente immer wichtiger.

Um entsprechenden Annotationen aus Mutltmedia-Dokumenten zu extrahieren, wird in einem ersten Schritt eine sogenannte Low-Level-Analyse durchgeführt, in der einfache Objekte und ihre Beziehungen untereinander erkannt werden. So sind beispielsweise aus dem Bild in Abbildung 1.1 die Objekte in Abbildung 1.2 extrahiert worden.

Figure 1.1: Stabhochsprung

Die Low-Level-Analyse ist allerdings nicht in der Lage, abstrakte Entitäten, die low-level Objekte zu einem high-level Objekt zusammenfassen, zu erkennen. So

$$
\begin{aligned}
person_1 &: \texttt{Person} \\
bar_1 &: \texttt{HorizontalBar} \\
pillar_1 &: \texttt{Pillar} \\
(athlete_1, bar_1) &: \texttt{isOverlapping} \\
(pillar_1, bar_1) &: \texttt{isNear}
\end{aligned}
$$

Figure 1.2: Analyse-ABox

wurde zum Beispiel durch die Analyse nicht erkannt, dass in Abbildung 1.1 ein Stabhochsprung-Ereignis dargestellt ist, oder dass die abgebildete Person ein Athlet bzw. Stabhochspringer ist. Stattdessen wird in einem zweiten Schritt, der Interpretation, versucht, aus den durch eine Analyse gewonnen Annotationen high-level Objekte zu identifizieren.

Im Rahmen dieser Arbeit wird die Interpretation durch Abduktion betrachtet. Abduktion ist ein logisches Schlussverfahren, dass von Beobachtungen auf mögliche Erklärungen schliesst. In Abbildung 1.1 kann man beispielsweise aus der Beobachtung zweier Pfeiler, einer horizontalen Stange und einer Person, die sich über der horizontal Stange befindet, auf ein Hochsprung-Ereignis als mögliche Erklärung schliessen.

Konkret wird Interpretation durch die rückwärtsverkettete Anwendung von Regeln durchgeführt. Das Erstellen dieser Regeln ist ein nicht-triviales Problem. Zum einen muss festgelegt werden, welche Beobachtungen auf high-level Konzepte schliessen lassen. Je komplizierter ein high-level Konzept ist und je mehr Varianten es gibt, desto mehr Regeln werden benötigt, um es zu beschreiben. Auch muss darauf geachtet werden, dass Regeln keine Konzepte hypothetisieren, wenn diese Konzepte im betrachteten Multimedia-Dokument tatsächlich gar nicht vorhanden sind.

In dieser Arbeit sollen nun automatische Lernverfahren für Abduktionsregeln untersucht werden. Dazu wird in Kapitel 2 eine kurze Einführung in die Beschreibungslogik und DL-sichere Regeln gegeben. Anschliessend wird in Kapitel 3 die Abduktion genauer erläutert und in Kapitel 4 erklärt, was unter Interpretation zu verstehen ist. Die für diese Arbeit wichtigen Konzepte des Maschinellen Lernens werden in Kapitel 5 beschrieben. Schliesslich werden zwei Algorithmen für das Lernen von Interpretationsregeln in Kapitel 6 vorgestellt und anschliessend die Ergebnisse dieser Arbeit in Kapitel 7 zusammengefasst.

$$
\begin{aligned}
polevault_1 &: \texttt{PoleVault} \\
person_1 &: \texttt{PoleVaulter} \\
bar_1 &: \texttt{HorizontalBar} \\
pillar_1 &: \texttt{Pillar} \\
(polevault_1, person_1) &: \texttt{hasParticipant} \\
(polevault_1, bar_1) &: \texttt{hasPart} \\
(polevault_1, pillar_1) &: \texttt{hasPart} \\
(athlete_1, bar_1) &: \texttt{isOverlapping} \\
(pillar_1, bar_1) &: \texttt{isNear}
\end{aligned}
$$

Figure 1.3: Analyse-ABox

Kapitel 2

Beschreibungslogiken

Die Informationen in diesem Kapitel wurden wurden [2] und [5] entnommen.

Beschreibungslogiken (engl. Descripion Logics) sind eine Familie von Sprachen mit einer formalen logik-basierten Semantik und dienen der Wissensrepräsentation. Mit Hilfe von Beschreibungslogiken kann man das Wissen einer Anwendungsdomäne beschreiben, indem man die Terminologie der Domäne mittels Konzepten und Rollen definiert und den Zustand der Welt mittels Konzept- und Rollenzusicherung auf Individuen beschreibt. Hierbei sind Konzept und Rollen einstellige bzw. zweistellige Prädikate und Individuen Konstanten.

Beschreibungslogiken sind ein Fragment der Prädikatenlogik erster Stufe und im Gegensatz zu diesen entscheidbar. Diese Eigenschaft ist die Basis für Inferenzdienste, mit deren Hilfe aus explizitem Wissen in der Wissensbasis implizites Wissen gefolgert werden kann. Beispiele für Inferenzdienste sind die Ermitltlung von Subsumptionsbeziehungen zwischen Konzepten oder Rollen und von Instanzbeziehungen zwischen Konzepten und Individuen.

Eine beschreibungslogische Wissensbasis wird in eine sog. TBox und eine ABox aufgeteilt. In der TBox wird das terminologische wissen mittels Axiomen über Konzepte und Rollen definiert. Die Abox beschreibt den Zustand Welt mittels Konzept- und Rollenzusicherungen auf Individuen.

2.1 Konstruktoren für Konzepte und Rollen

Elementare Beschreibungen einer Beschreibungslogik sind atomare Konzepte und atomare Rollen. Aus diesen lassen sich induktiv mittels Konstruktoren komplexe Beschreibungen erzeugen. In Abbildung 2.1 sind als Beispiel die Konzeptkonstruktoren der Sprache $\mathcal{AL}$ aufgeführt.

Beschreibungslogiken unterscheiden sich in der Menge und Art der Konstruktoren, die sie zur Verfügung stellen. Die Konstruktoren einer Beschreibungslogik bestimmen ihre Ausdrucksmächtigkeit.

$$
\begin{array}{rcll}
\mathtt{C}, \mathtt{D} & \longrightarrow & \mathtt{A} & | \quad \text{(Atomares Konzept)} \\
 & & \top & | \quad \text{(Universelles Konzept)} \\
 & & \bot & | \quad \text{(Unerfüllbares Konzept)} \\
 & & \neg\mathtt{A} & | \quad \text{(Universelles Konzept)} \\
 & & \mathtt{C} \sqcap \mathtt{D} & | \quad \text{(Durchschnitt)}
\end{array}
$$

Figure 2.1: Konstruktoren der Sprache $\mathcal{AL}$

In Abbildung 2.1 bezeichnet $\top$ die Menge aller Konzepte und $\bot$ das unerfüllbare Konzept. $\mathtt{C}$ und $\mathtt{D}$ sind komplexe Konzepte, die mithilfe der Konstruktoren, atomaren Konzepten, $\top$ und $\bot$ konstuiert werden können. Konstruktoren für Rollen werden analog definiert.

Um eine formale Semantik zu definieren wird eine sogenannte Interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ betrachtet, die aus einer nicht-leeren Menge $\Delta^{\mathcal{I}}$ und einer Interpretationsfunktion $\cdot^{\mathcal{I}}$ besteht. Die Interpretationsfunktion bildet jedes atomare Konzept $\mathtt{A}$ auf eine Menge $\mathtt{A}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ ab und jede atomare Rolle $\mathtt{R}$ auf eine zweistellige Relation $\mathtt{R}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Die Interpretationsfuktion wird durch die induktive Definitionen in Abbildung 2.2 auf komplexe Beschreibungen erweitert.

$$
\begin{array}{rcl}
\top^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} & = & \emptyset \\
\neg\mathtt{A}^{\mathcal{I}} & = & \Delta \backslash \mathtt{A}^{\mathcal{I}} \\
(\mathtt{C} \sqcap \mathtt{D})^{\mathcal{I}} & = & \mathtt{C}^{\mathcal{I}} \cap \mathtt{D}^{\mathcal{I}} \\
(\forall \mathtt{R}.\mathtt{C})^{\mathcal{I}} & = & \{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in \mathtt{R}^{\mathcal{I}} \rightarrow b \in \mathtt{C}^{\mathcal{I}}\} \\
(\exists \mathtt{R}.\top)^{\mathcal{I}} & = & \{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in \mathtt{R}^{\mathcal{I}}\}
\end{array}
$$

Figure 2.2: Interpretationsfunktion

Andere Beschreibungslogiken der $\mathcal{AL}$-Familie lassen sich erzeugen, indem weitere Konstruktoren zur $\mathcal{AL}$ hinzugefügt werden. Beispielhaft sind in Abbildung 2.3 vier Konstruktoren beschrieben.

Name	**Syntax**	**Semantik**
Vereinigung ($\mathcal{U}$)	$\mathtt{C} \sqcup \mathtt{D}$	$(\mathtt{C} \sqcup \mathtt{D})^{\mathcal{I}} = \mathtt{C}^{\mathcal{I}} \cup \mathtt{D}^{\mathcal{I}}$
Vollst. Existenzrestrikt. ($\mathcal{E}$)	$\exists \mathtt{R}.\mathtt{C}$	$(\exists \mathtt{R}.\mathtt{C})^{\mathcal{I}} =$ $\{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in \mathtt{R}^{\mathcal{I}} \wedge b \in \mathtt{C}^{\mathcal{I}}\}$
Anzahlrestriktion ($\mathcal{N}$)	$\leq n\mathtt{R}$	$(\leq n\mathtt{R})^{\mathcal{I}} =$ $\{a \in \Delta^{\mathcal{I}} \mid \lvert\{b \mid (a,b) \in \mathtt{R}^{\mathcal{I}}\}\rvert \leq n\}$
Anzahlrestriktion ($\mathcal{N}$)	$\geq n\mathtt{R}$	$(\geq n\mathtt{R})^{\mathcal{I}} =$ $\{a \in \Delta^{\mathcal{I}} \mid \lvert\{b \mid (a,b) \in \mathtt{R}^{\mathcal{I}}\}\rvert \geq n\}$
Komplexe Negation ($\mathcal{C}$)	$\neg\mathtt{C}$	$(\neg\mathtt{C})^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash \mathtt{C}^{\mathcal{I}}$

Figure 2.3: Konstruktoren um $\mathcal{AL}$ zu erweitern

Aufgrund der semantischen Definition können Vereinigung ($\mathcal{U}$) und Vollständi-

ge Existenzrestriktion $\mathcal{E}$ durch die Komplexe Negation ($\mathcal{C}$)ausgedrückt werden und umgekehrt. Die durch diese zusätzlichen Konstruktoren erzeugbaren Sprachen bezeichnet man als $\mathcal{ALE}$, $\mathcal{ALU}$, $\mathcal{ALN}$, $\mathcal{ALCN}$, usw.

2.1.1 TBox

Die TBox einer beschreibungslogischen Wissensbasis ist eine Definition der Terminologie einer Anwendungsdomäne und beschreibt somit das intensionale Wissen über eine Domäne. Zu diesem Zweck ist eine TBox aus zwei Typen von Axiomen zur Beschreibung von Konzepten und Rollen aufgebaut.

	Syntax	**Semantik**
Äquivalenzaxiom	$\mathtt{C} \equiv \mathtt{D}$	$(\mathtt{C} \equiv \mathtt{D})^{\mathcal{I}} \doteq (\mathtt{C}^{\mathcal{I}} = \mathtt{D}^{\mathcal{I}})$
	$\mathtt{R} \equiv \mathtt{S}$	$(\mathtt{R} \equiv \mathtt{S})^{\mathcal{I}} \doteq (\mathtt{R}^{\mathcal{I}} = \mathtt{S}^{\mathcal{I}})$
Inklusionsaxiom	$\mathtt{C} \sqsubseteq \mathtt{D}$	$(\mathtt{C} \sqsubseteq \mathtt{D})^{\mathcal{I}} \doteq (\mathtt{C}^{\mathcal{I}} \sqsubseteq \mathtt{D}^{\mathcal{I}})$
	$\mathtt{R} \sqsubseteq \mathtt{S}$	$(\mathtt{R} \sqsubseteq \mathtt{S})^{\mathcal{I}} \doteq (\mathtt{R}^{\mathcal{I}} \subseteq \mathtt{S}^{\mathcal{I}})$

Hierbei sind C, D komplexe Konzepte und R, S komplexe Rollen.

Äquivalenzaxiome mit nur einem atomaren Konzept bzw. einer atomaren Rolle auf der linken Seite werden auch als Definitionen bezeichnet und dienen der Einführung symbolischer Namen für komplexe Konzepte bzw. Rollen:

$$\mathtt{Parent} \equiv \mathtt{Father} \sqcup \mathtt{Mother}$$

Inklusionsaxiom deren linke linke Seite nur aus einem atomaren Konzept bzw. einer atomaren Rolle besteht, werden auch als Spezialisierungen bezeichnet. Spezialisierungen sind hilfreich, wenn die exakte Beschreibung eines Konzepts oder Rolle nicht bekannt ist:

$$\mathtt{Athlet} \sqsubseteq \mathtt{Mensch}$$

Disjunktionen können ebenfalls mit Inklusionsaxiomen beschrieben werden:

$$\mathtt{Mann} \sqcap \mathtt{Frau} \sqsubseteq \bot$$

2.1.2 Abox

In der ABox wird der Zustand der Welt mit Hilfe von Zusicherungen auf Individuen beschrieben. Die Individuen werde durch die ABox eingeführt, indem für jedes Individuum in der ABox ein Name vergeben wird. Die Individuen werden durch Konzept- und Rollenzusicherungen, der Form $\mathtt{C}$(a) bzw. $\mathtt{R}$(a,b) beschrieben, wobei a und b Individuenkonstanten, $\mathtt{R}$ eine Rolle und $\mathtt{C}$ ein Konzept ist.

Eine Semantik für die ABox ist durch eine Erweiterung der Interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ gegeben, indem die Interpretationsfunktion zusätzlich jeder Individuenkonstante auf ein Element $a^{\mathcal{I}} \subseteq \Delta^{I}$ abbildet. Da angenommen wird, dass

unterschiedliche Individuenkonstanten unterschiedliche Individuen bezeichnen (Unique Name Assumption) gilt für die Interpretationsfunktion $a^{\mathcal{I}} \neq b^{\mathcal{I}}$.

Zusätzlich wird eine sog. offenen Welt angenommen (Open World Assumption), indem davon ausgegangen wird, dass das Wissen in der Wissensbasis nicht vollständig ist. Wenn also eine ABox nur die Zusicherungen $\texttt{hatKind}(mary, john)$, $\texttt{hatKind}(mary, john)$, $\texttt{Junge}(john)$ und $\texttt{Junge}(mary, james)$ enthält, kann daraus nicht geschlossen werden, dass Mary nur zwei oder nur männliche Kinder hat. Um diesen Schluß machen zu können, müsste der ABox zusätzlich die Zusicherung $\leq 2\texttt{hatKind}(mary)$ hinzugefügt werden.

2.2 Regeln

Eine beschreibungslogische Wissensbasis kann - unter Erhaltung der logischen Interpretierbarkeit - durch Regeln erweitert werden. Diese Regeln dienen als Mechanismus um Zusicherungen zu einer ABox hinzuzufügen. Ein einfaches Beispiel für eine Regel ist

$$\texttt{C}(X) \leftarrow \texttt{D}(X), \texttt{E}(X).$$

Hierbei sind `C`, `D` und `E` Konzepte und X ist eine Individuen-Variable. Regeln haben maximal ein Atom im Regelkopf, sind nicht-rekursiv und alle Variablen, die im Regelkopf vorkommen, müssen auch im Regelkörper vorkommen.

Die Bedeutung der obigen Regel ist Folgende: Wenn von einem Individuum bewiesen werden kann, dass es eine Instanz von `D` und `E` ist, dann kann aus der Regel abgeleitet werden, dass dieses Individuum auch eine Instanz von `C` ist.

Die operationelle Semantik einer Regel kann wie folgt beschrieben werden: Wir gehen von einer Wissensbasis $\mathcal{K}(\mathcal{T}, \mathcal{A})$ mit einer TBox $\mathcal{T}$ und einer ABox $\mathcal{A}$ sowie einer endliche Regelmenge $\mathcal{R}$ aus. Aus der Wissenbasis $\mathcal{K}$ wird eine Folge von Wissensbasen $\mathcal{K}^0, \mathcal{K}^1, \mathcal{K}^2, ...$ konstruiert wobei $\mathcal{K}^0 = \mathcal{K}$ ist und $\mathcal{K}^{i+1}$ aus $\mathcal{K}^i$ erzeugt wird, indem eine neue Zusicherung $\texttt{D}(a)$ hinzugefügt wird, wenn R eine Regel $\texttt{D}(X) \leftarrow \texttt{C}(X)$ enthält und $\mathcal{K}^i \models \texttt{C}(a)$ gilt, aber $\mathcal{K}^i$ die Zusicherung $\texttt{D}(a)$ noch nicht enthält.

Diese Prozedur wird nach eine endlichen Anzahl von Schritten anhalten, weil die Wissenbasis $\mathcal{K}$ nur endlich viele Individuen und $\mathcal{R}$ nur endlich viele Regeln enthält.

Mithilfe von Regeln lassen sich Sachverhalte beschreiben, die sich mit den Axiomen, Konstruktoren und Inferenzdiensten der Beschreibungslogik alleine nicht ausdrücken lassen. Beispielsweise kann aus den Zusicherungen $\texttt{Bruder}(a, b)$ und $\texttt{Vater}(b, c)$ nicht auf eine Onkel-Beziehung $\texttt{Onkel}(a, c)$ geschlossen werden. Mit der Regel

$$\texttt{Onkel}(X, Y) \leftarrow \texttt{Bruder}(X, Z), \texttt{Vater}(Z, Y)$$

lässt sich die Onkel-Beziehung für die entsprechenden Individuen hinzufügen.

2.2.1 Inferenzdienste

Ein Wissensrepräsentationssystem speichert nicht nur Informationen über die Terminologie und den Zustand der Welt in einer TBox bzw. ABox, sondern kann auch über die Informationen in der Wissenbasis schlußfolgern. Im folgenden werden Standard-Inferenzdienste kurz erläutert.

Consistency Check überprüft die Konsitenz von Konzepten `C`, ABoxen $\mathcal{A}$ und Wissensbasen $\Sigma = (\mathcal{A}, \mathcal{T})$. Ein Konzept `C` wird als konsistent (bezüglich einer TBox $\mathcal{T}$) bezeichnet, wenn ein Modell für `C` existiert (das ebenso ein Modell von $\mathcal{T}$ ist). Eine ABox $\mathcal{A}$ ist konsistent (bezüglich einer TBox $\mathcal{T}$, wenn $\mathcal{A}$ ein Modell hat (das ebenfalls ein Modell von $\mathcal{T}$ ist). Eine Wissensbasis $\Sigma = (\mathcal{A}, \mathcal{T})$ ist konsistent, wenn ein Modell für $\mathcal{A}$ existiert, dass ebenfalls ein Modell von $\mathcal{T}$ ist.

Subsumption Check Ein grundlegender Inferenzdienst ist die Ermittlung der Subsumptionsbeziehungen zwischen allen Konzepten um eine Subsumptionshierarchie aufzubauen bzw. die TBox zu klassifizieren.

TBox Coherence Check Mit dem TBox Coherence Check wird die Konsistenz aller Konzepte einer TBox geprüft.

Instance-Checking Mit einem Instance Check wird überprüft, ob ein Individuum i eine Instanz eines Konzeptes `C` ist.

Direct Type Inference Mittels der Direct Type Inference werden die spezifischsten Konzepte eines Individuums ermittelt.

ABox Entailment Eine ABox $\mathcal{A}$' ist eine logische Konsequenz aus einer TBox $\mathcal{T}$ und einer ABox $\mathcal{A}$, wenn alle Modelle von $\mathcal{T}$ und $\mathcal{A}$ ebenfalls Modelle von $\mathcal{A}$' sind: $\mathcal{T} \cup \mathcal{A} \vDash \mathcal{A}'$.

Retrieval Inference findet alle Individuen einer ABox $\mathcal{A}$, die Instanzen eines bestimmten Konzeptes `C` sind.

Kapitel 3

Abduktion

Die Informationen in diesem Kapitel wurdenn [2] und [3] entnommen.

Die Abduktion ist ein logisches Schlußverfahren, dass von Beobachtungen auf Erklärungen bzw. Hypothesen schliesst. Eingeführt wurde dieses Verfahren von Charles Sanders Pierce (1839-1914).

Im Unterschied zu den Schlußverfahren Deduktion und Induktion ist Abduktion durch die Aufstellung von Hypothesen kenntniserweiternd. Das bedeutet dass mittels Abduktion neues Wissen hinzugelernt werden kann. Allerdings kann der Grad der Gültigkeit dieses hypothetisierten Wissens in der Regel nicht gesichert bestimmt werden.

Zum Beispiel könnte man bei einem Auto, dessen Motor nicht startet, darauf schließen, dass die Batterie defekt ist. Hier ist die Beobachtung "Motor startet nicht" und die Hypothese ist "Batterie defekt". Allerdings können ebenso defekte Kabel die Ursache für einen nicht startenden Motor sein (bei intakter Batterie). Das bedeutet, dass eine gefunden Erklärung/Hypothese nicht immer zwingend die korrekte Hypothese ist. Im allgemeinen wird es für eine Beobachtung mehrere mögliche Hypothesen geben.

Als Inferenzdienst gehört Abduktion zu den Nicht-Standard-Inferenzdiensten und versucht aus einer gegebenen Menge von Zusicherungen Γ eine minimale Menge von Erklärungen Δ zu konstruieren, so daß Δ konsistent bezüglich der Wissensbasis $\Sigma = (\mathcal{T}, \mathcal{A})$:

1. $\Sigma \cup \Delta \models \Gamma$

2. $\Sigma \cup \Delta \not\models \bot$

3. Es existiert keine andere Hypothese Δ' in der Menge der Lösungen, die nicht äquivalent zu Δ ist und für die gilt $\Delta' \models \Delta$

Die erste Bedingung fordert die Konsistenz der Wissensbasis nach Hinzufügen der Hypothese. Die Zweite, dass Γ eine logische Konsequenz von $\Sigma \cup \Delta$ ist und damit Δ eine Erklärung für Γ ist. Die Dritte Bedingung schliesslich fordert die Minimalität der Hypothesenmenge.

Abduktion gehört zu den nicht-monotonen Inferenzdiensten, weil man die Korrektheit einer Abduktionsypothese im allgemeinen nicht bestimmen kann. Beispielsweise kann nach Hinzufügen von Zusicherungen auf Basis von Abduktionshypothesen eine Wisssensbasis inkonsistent werden, wenn später weitere (korrekte) Informationen hinzugefügt werden, die inkonsistent zu den Abduktionshypothesen sind.

3.1 Abductive Query Answering Service

Konkret wird Abduktion in einem beschreibungslogischen Wissensrepräsentationssystem durch eine Rückwärtsverkettung von Regeln der Wissensbasis ausgeführt. Dazu wird zuerst eine Abfrage konstruiert und dann mittels des Abductive Query Answering Services eine Erklärung ermittelt.

$$\begin{aligned} bar_1 &: \texttt{HorizontalBar} \\ athlete_1 &: \texttt{Athlete} \\ (athlete_1, bar_1) &: \texttt{near} \end{aligned}$$

Figure 3.1: ABox

$$\texttt{near}(X, Y) \leftarrow \texttt{HighJump}(Z), \texttt{hasPart}(Z, X), \texttt{hasPart}(Z, Y), \texttt{Athlete}(X), \texttt{HorizontalBar}(Y)$$

Figure 3.2: Abduktionsregel

$$Q := \{() \mid \texttt{near}(athlete_1, bar_1)\}$$

Figure 3.3: Abduktionsabfrage

Als ein Beispiel nehmen wir die Abox Γ in Abbildung 3.1 und möchten durch Abduktion eine Erklärung für die Rollenzusicherung $\texttt{near}(athlete_1, bar_1)$ ermitteln. Dazu wird die Abfrage in Abbildung 3.3 erzeugt und ausgeführt. Für alle Atome in der Abfrage (in diesem Fall hat die Abfrage nur ein Atom) werden die Regeln gesucht, deren Kopf dem Abfrage-Atom entsprechen. Das Atom in der Abfrage wird durch den Regelkörper ersetzt und die Variablen der Regel werden auf die Individuenkonstanten in der Abfrage abgebildet. Dies wird solange durchgeführt, bis kein Atom mehr einem Regelkopf entspricht. Da die Regeln nicht-rekursiv sind, ist garantiert, dass dieser Ersetzungsvorgang nach endlich vielen Schritten beendet wird.

Anschliessend wird die Abfrage mittels des Abductive Query Answering Services ausgeführt, indem für jede noch ungebundene Variable eine konsistente Abbildung auf die Menge der Individuen in der ABox gesucht wird. Wenn für eine Zusicherung keine Variablenbindung in der ABox gefunden werden kann, wird versucht eine Erklärung zu finden, indem entweder ein neues Individuum hypothetisiert oder die Zusicherung auf ein bestehendes Individuum angewendet wird. Wenn für ein Abfrage-Atom keine Regel gefunden werden kann, wird als Erklärung die triviale Erklärung ausgegeben - nämlich das Abfrage-Atom selber. Im allgemeine gibt es mehrere Erklärungen, d.h., mehrere unterschiedliche Abbildungen von Individuen auf Variablen und/oder unterschiedliche Mengen von Hypothesen.

Eine formale Beschreibung des Abductive Query Answering Service ist in [3] gegeben:

Abductive Query Answering Service Gegeben sei eine eine ABox $\mathcal{A}$ und eine konjunktive Abuktionsabfrage q der Form $ans(x) \leftarrow a_1 \wedge ... \wedge a_n$. Die Antwort von q auf $\mathcal{A}$ des Abductive Query Answering Services ist ein Menge von Abbildungsfunktionen $\mathcal{M} = \{\Phi_1...\}$ mit $\Phi_i : vars(q) \mapsto inds(\mathcal{A}) \cup \{c_1, ..., c_m\}$, wobei $\{c_1, ..., c_m\} \cap inds(\mathcal{A}) = \emptyset, m = |vars(q)|$, so dass für jedes Φ_i die folgende Aussagen gelten

1. $\mathcal{A} \cup (hypothesizedAssertions(\beta(\Phi_i(q)))) \models \beta(\Phi_i))$, und
2. $\mathcal{A} \cup (hypothesizedAssertions(\beta(\Phi_i(q))))$ ist konsistent.

hypothesizedAssertion ist folgendermaßen definiert:

$hypothesizedAssertions(\beta(\Phi_i(q))) =_{def} \{a \mid a \in \beta(\Phi_i(q)), \mathcal{A} \not\models a\}$

β **- Funktion** transformiert eine Abfrage $q = a_1 \wedge ... \wedge a_n$, für die gilt $vars(\Phi_i(q)) = \emptyset$, in eine ABox:

- $\beta(q) =_{def} \{\beta(a_i) \mid i \in 1...n\}$
- $\beta(\mathtt{C}(i)) =_{def} i : \mathtt{C}$
- $\beta(R(i,j)) =_{def} (i,j) : \mathtt{R}$

3.2 Scoring-Funktion

Da der Abductive Query Answering Services eine Menge von Abbildungsfunktionen $\mathcal{M} = \{\Phi_1, ...\}$ und somit eine Menge von Hypothesen $\mathcal{H} = \{H_1, ...\} = \{\beta(\Phi_1(q)), ...\}$ erzeugt, muss eine bevorzugte Hypothese ausgewählt werden.

Dazu werden mit Hilfe der Funktionen in Abbildung 3.4 folgende Scoring-Funktionen definiert [3] [7]:

(1) $|entailedAssertions(H_i)| - |hypothesizedAssertions(H_i)|$ [3]
(2) $|oldInds(H_i)| - |newInds(H_i)|$ [7]

Beide Scoring-Funktionen liefern um so höhere Werte, je weniger hypothetisiert wird und je mehr vorhandene Zusicherungen bzw. Individuen verwendet werden Dies entspricht den Kriterien der größtmöglichen Einfachheit (Simplicity) und maximalen Übereinstimmung (Consilience) [6].

In dieser Arbeit kommt als Scoring-Funktion die Funktion (1) zur Anwendung, denn hypothetisierte Individuen kommen immer in hypothetisierten Zusicherungen vor, aber hypothetisierte Zusicherungen enthalten nicht zwingend hypothetisierte Individuen. D.h. die Scoring-Funktion (1) berücksichtigt mehr Unterschiede zwischen einer ABox $\mathcal{A}$ und einer ABox $\mathcal{A}'$ die aus $\mathcal{A}$ nach der Anwendung des Abduction Query Answering Service hervorgegangen ist.

$hypothesizedAssertions(H_i)$	$=_{def}$	$\{a \mid a \in H_i, \mathcal{A} \nvDash a\}$
$entailedAssertions(H_i)$	$=_{def}$	$\{a \mid a \in H_i, A \vDash a\}$
$oldInds(H_i)$	$=_{def}$	$indsH_i \cap inds(\mathcal{A})$
$newInds(H_i)$	$=_{def}$	$inds(H_i) \cap \{c_1, ..., c_m\}, m = \lvert vars(q)\rvert$

Figure 3.4: Scoring-Funktionen

Beispiel: Gegeben sind die Abox in Abbildung 3.5 und zwei Regeln in Abbildung 3.6. Beide Regeln hypothetisieren jeweils ein neues Individuum (siehe Abbildungen 3.8 und 3.8) und zwei Rollenzusicherungen, aber Regel (2) zusätzlich noch eine Konzeptzusicherung. Das bedeutet, dass der Score-Wert für Regel 2 niedriger ist, da mehr hypothetisiert wird:

$$\begin{array}{lcll} Score_{Regel1} & = & entailedAsserts - hypothesizedAsserts & = & 2-3=-1 \\ Score_{Regel2} & = & entailedAsserts - hypothesizedAsserts & = & 1-4=-3 \end{array}$$

Regel 2 fügt die Zusicherung (bar_1 : `Javelin`) zur ABox hinzu, die schon die Zusicherung (bar_1 : `HorizontalBar`) enthält. Sollte die TBox ein Disjointnes-Axiom der Form

$$\texttt{HorizontalBar} \sqcap \texttt{Javelin} \sqsubseteq \bot$$

enthalten, wäre die ABox in Abbildung 3.8 inkonsistent, d.h. die Bedingung $\Sigma \cup \Delta \nvDash \bot$ wäre verletzt, und die Regel hätte nicht angewendet werden dürfen.

$$\begin{array}{rcl} bar_1 & : & \texttt{HorizontalBar} \\ athlete_1 & : & \texttt{Athlete} \\ (athlete_1, bar_1) & : & \texttt{near} \end{array}$$

Figure 3.5: ABox 2

$$
\begin{array}{lll}
(1) & \texttt{near}(X,Y) & \leftarrow \quad \texttt{HighJump}(Z), \texttt{hasParticipant}(Z,X), \texttt{hasPart}(Z,Y), \\
 & & \quad\ \texttt{Athlete}(X), \texttt{HorizontalBar}(Y) \\
(2) & \texttt{near}(X,Y) & \leftarrow \quad \texttt{JavelinThrow}(Z), \texttt{hasParticipant}(Z,X), \\
 & & \quad\ \texttt{hasPart}(Z,Y), \texttt{Athlete}(X), \texttt{Javelin}(Y)
\end{array}
$$

Figure 3.6: Regeln

$$
\begin{array}{rcl}
highjump_1 & : & \texttt{HighJump} \\
bar_1 & : & \texttt{HorizontalBar} \\
athlete_1 & : & \texttt{Athlete} \\
(highjump_1, athlete_1) & : & \texttt{hasParticipant} \\
(highjump_1, bar_1) & : & \texttt{hasPart} \\
(athlete_1, bar_1) & : & \texttt{near}
\end{array}
$$

Figure 3.7: ABox 2 nach Anwendung von Regel 1

$$
\begin{array}{rcl}
javelinthrow_1 & : & \texttt{JavelinThrow} \\
bar_1 & : & \texttt{Javelin} \\
bar_1 & : & \texttt{HorizontalBar} \\
athlete_1 & : & \texttt{Athlete} \\
(javelinthrow_1, athlete_1) & : & \texttt{hasParticipant} \\
(javelinthrow_1, bar_1) & : & \texttt{hasPart} \\
(athlete_1, bar_1) & : & \texttt{near}
\end{array}
$$

Figure 3.8: ABox 2 nach Anwendung von Regel 2

Kapitel 4

Interpretation

Interpretation bedeutet die Hypothetisierung übergeordneter (high-level) Konzepte bzw. Objekte in einer ABox. Es wird dabei angenommen, dass die ABox durch eine automatische Analyse eines Multimedia-Dokuments, wie zum Beispiel einer Textdatei, eines Bildes, eines Audio-Clips oder eines Filmes, gewonnen wurde. Durch eine automatische Analyse können nur primitive (low-level) Objekte erkannt werden, wie zum Beispiel Personen, ein Ball oder ein Tor. Die Interpretation der Gesamtmenge dieser Objekte - beispielsweise als Fußballspiel - kann durch eine einfache Analyse nicht geleistet werden.

Als konkretes Beispiel betrachten wir das Bild in Abbildung 4.1 mit der zugehörigen ABox in Abbildung 4.2 Diese ABox wurde durch eine automatisierte low-level Analyse des Bildes erzeugt und beschreibt die primitiven Objekte, die erkannt wurden. Aus der ABox wird deutlich, dass durch die Analyse das komplexere Konzept "Speerwurf" nicht extrahiert wurde. Dieses high-level Konzept tritt nicht als einfaches Objekt in Erscheinung, sondern aggregiert als übergeordnetes Objekt mehrere primitive Objekte. Tatsächlich ist die Aggregation von untergeordneten Konzepten ein wesentliches Merkmal von durch Interpretation gewonnen Konzepten.

Figure 4.1: Speerwurf

Eine Aggregation ist dadurch gekennzeichnet, das zwischen den beteiligten Objekten eine Teil-Ganzes-Beziehung besteht. Allgemein bilden die Elemente der

$$
\begin{array}{rcl}
person_1 & : & \texttt{Person} \\
javelin_1 & : & \texttt{Javelin} \\
(athlete_1, javelin_1) & : & \texttt{near}
\end{array}
$$

Figure 4.2: Speerwurf-ABox nach low-level Analyse)

$$
\begin{array}{rcl}
throw_1 & : & \texttt{JavelinThrow} \\
person_1 & : & \texttt{JavelinThrower} \\
javelin_1 & : & \texttt{Javelin} \\
(throw_1, person_1) & : & \texttt{hasParticipant} \\
(throw_1, javelin_1) & : & \texttt{hasPartr} \\
(athlete_1, javelin_1) & : & \texttt{near}
\end{array}
$$

Figure 4.3: Speerwurf-ABox nach Interpretation

Aggregation einen gerichteten azyklischen Graphen bezüglich der Teil-Ganzes-Beziehungen. Ein Objekt kann Bestandteil mehrere Aggregate sein und allgemein können Aggregate eine Kompositionshierarchie bilden [5]. So aggregiert zum Beispiel der Speerwurf in Abbildung 4.1 den Speer und den Speerwerfer (siehe Abbildung 4.3 mit der ABox nach der Interpretation) und der Speerwerfer wiederum aggregiert die anatomischen Einzelteile seines Körpers, wie zum Beispiel Kopf, Arme, Rumpf usw.

Ein Aggregat kann nicht alleine mit den Mitteln der Beschreibungslogik beschrieben werden, da die möglicherweise komplexen Rollenbeziehungen zwischen den Bestandteilen eines Aggregats mehrwertige Prädikate verlangen. Die Prädikatenlogik erste Stufe bietet bietet zwar mehrwertige Prädikate, aber die Beschreibungslogik ist nur ein Fragment der Prädikatenlogik und auf zweiwertige Prädikate (Rollen) beschränkt. Aus diesem Grund werden beschreibungslogische Wissensbasen oft um Regeln ergänzt, deren vorwärtsverkettete Ausführung deduktiv zusätzliche Konzept- oder Rollenzusicherungen realisiert. In Abbildung 4.4 ist die allgemeine Struktur einer Aggregation mit den beschränkten Mitteln der Beschreibungslogik beschrieben. Rollenbeziehungen zwischen den Aggregatsbestandteilen können nur als Constraints angedeutet werden [5].

$$
\begin{array}{rl}
\texttt{Aggregate} \equiv & \texttt{Parent}_1 \sqcap ... \sqcap \texttt{Parent}_1 \sqcap \\
& \exists_{\geq m_1} \texttt{hasPart.Part}_1 \sqcap \\
& ... \\
& \exists_{\geq m_k} \texttt{hasPart.Part}_k \sqcap \\
& \text{Constraints zwischen Part-Konzepten}
\end{array}
$$

Figure 4.4: Aggregation

DL-sichere Regeln haben die Eigenschaft, dass Variablen im Regelkopf auch zwingend im Regelkörper vorkommen müssen. Das bedeutet, das durch eine

vorwärtsverkettete Anwendung von Regeln, keine neuen Individuen erkannt bzw. realisiert werden können.

Aus diesem Grund werden für die Interpretation die Regeln rückwärtsverkettet ausgeführt. Rückwärtsverkettung bedeutet, dass in einer ABox eine Zusicherung durch einen Regelkörper ersetzt wird, wenn der Regelkopf auf die Zusicherung anwendbar ist. Dies wird solange wiederholt, bis keine Regel mehr angewendet werden kann. Der Vorgang endet nach endlich vielen Schritten, da die Regeln nicht-rekursiv sind und es nur eine endliche Anzahl von Regeln und Zusicherungen gibt.

Dabei können bei jeder Ersetzung einer Zusicherung durch den Regelkörper neue Zusicherungen und auch Individuen hypothetisiert werden. Diese Hypothetisierung von Individuen oder Zusicherungen auf Individuen kann als Abduktion aufgefasst werden. In Abbildung 4.6 ist eine ABox und in Abbildung 4.5 eine Regel abgebildet . Die rückwärstverkettete Anwendung der Regel führt zur erweiterten ABox in Abbildung 4.7 mit zusäztlichen Konzept- und Rollenzusicherungen und einem neuen Individuum.

$$\texttt{isNear}(X, Y) \quad \leftarrow \quad \texttt{HighJump}(Z), \texttt{isParticipant}(Z, X), \texttt{hasPart}(Z, Y), \texttt{Person}(X), \texttt{HorizontalBar}(Y)$$

Figure 4.5: Abduktionsregel

$$\begin{aligned} person_1 &: \texttt{Person} \\ bar_1 &: \texttt{HorizontalBar} \\ (person_1, bar_1) &: \texttt{isNear} \end{aligned}$$

Figure 4.6: Hochsprung-ABox vor Interpretation

$$\begin{aligned} highjump_1 &: \texttt{HighJump} \\ person_1 &: \texttt{Person} \\ bar_1 &: \texttt{HorizontalBar} \\ (highjump_1, person_1) &: \texttt{hasParticipant} \\ (highjump_1, bar_1) &: \texttt{isPart} \\ (person_1, bar_1) &: \texttt{isNear} \end{aligned}$$

Figure 4.7: Hochsprung-ABox nach Interpretation

Durch Interpretation gefundene Zusicherung können nicht nur zur ABox-Erweiterung genutzt werden, sondern können auch der Verbesserung der Analyse dienen. Das Ergebnis und die Folgerungen aus der Interpretation kann dem Analyseprozess übergeben werden, um diesen in einem weiteren Analysedurchlauf die Erkennung von zusätzlichen low-level Objekten zu ermöglichen. Beispielsweise gehört zu einem Hochsprung-Ereignis neben dem Hochspringer und einer

Querstange im allgemeinen auch eine Hochsprungmatte. Wenn diese im ersten Analysedurchlauf nicht erkannt wurde, kann die Information, dass es sich um ein Hochsprung-Ereignis handelt, vom Analyseprozess zur Optimierung genutzt werden, um das fehlende Objekt im zweiten Versuch zu erkennen.

Kapitel 5

Maschinelles Lernen

Maschinelles lernen ist die automatische Optimierung eines Modells oder Algorithmus mittels Beispieldaten oder aus Erfahrung. In den Beispieldaten bzw. Erfahrungswerten werden Gesetzmäßigkeiten erkannt, um nach der Lernphase neue Daten korrekt beurteilen zu können. Die Kernaufgabe besteht also darin, aufgrund eine Stichprobe bzw. Trainingsdaten Schlußfolgerungen zu ziehen [1].

Probleme des Maschinellen Lernens bestehen nicht nur aus dem von Finden von effizienten Algorithmen, um Optimierungsprobleme zu lösen, sondern die erlernten Verfahren müssen ebenfalls effizient anwendbar sein. Bein manchen Anwendungen kann die Effizienz des Lernalgorithmus genauso wichtig sein wie seine Vorhersagegenauigkeit.

Der Oberbegiff Maschinelles Lernen umfasst eine große Gruppe von Lernmethoden. Mann kann die viele dieser Methoden in folgende Gruppen einteilen:

Überwachtes Lernen (engl. supervised learning) Der Algorithmus lernt anhand von Trainingsdaten. Dabei ist das korrekte Ergebnis für alle Trainingsdaten vorgegeben. Ziel ist es, eine Hypothese zu finden, die für alle Trainingsdaten das korrekte Ergebnis liefert. Hierbei wird die Annahme gemacht, dass auch für andere Daten das korrekte Ergebnis möglichst gut von der Hypothese approximiert wird.

Unüberwachtes Lernen (engl. unsupervised learning) Es liegen nur Trainingsdaten vor. Typischerweise besteht das Problem darin, die Menge der Trainingsdaten in geeignete Untermengen zu unterteilen.

Verstärkendes Lernen (engl. reinforcement learning) Es liegen Trainingsdaten vor und der Algorithmus lernt anhand von Rückmeldungen über die Güte der gelieferten Ergebnisse. Die korrekten Ergebnisse liegen allerdings nicht explizit vor, sondern das Modell wird über ein System von Belohnung und Bestrafung optimiert.

(a) High Jump

(b) Pole Vault

Figure 5.1: Klassifizierung (Bilder)

$athlete_1$	:	`Athlete`
bar_1	:	`HorizontalBar`
$(athlete_1, bar_1)$	:	`isOverlapping`

(a) High Jump

$athlete_1$	:	`Athlete`
bar_1	:	`HorizontalBar`
$(athlete_1, bar_1)$	:	`isOverlapping`

(b) Pole Vault

Figure 5.2: Klassifizierung (ABoxen)

5.1 Automatische Klassifizierung

Ein Teilgebiet des Maschinellen Lernens ist ist die automatische Klassifizierung. Dabei teilt ein Klassifikator eine Menge von Objekten in verschiedene Klassen ein. Auch hier kann man die Verfahren wieder in Überwachtes Lernen, Unüberwachtes Lernen und Verstärkendes Lernen einteilen. Die Klassifizierung wird anhand von Objektmerkmalen durchgeführt. Das Klassifizierungsproblem besteht darin, die Art, Anzahl und Konfiguration der Merkmale zu ermitteln, die für eine Klassifizierung optimal sind [1].

Für eine erfolgreiche Klassifizierung ist eine Voraussetzung das Vorhandensein von Merkmalen oder Merkmalsmengen, deren Werte oder Konfiguration Klassen voneinander abgrenzen können. Wenn diese Merkmale durch eine Aanalyse nicht extrahiert werden können, kann eine Klassifizierung nicht erfolgreich sein. Als Beispiel sollen die Bilder in Abbildung 5.1 dienen, in denen ein Hochsprung- und ein Stabhochsprung-Ereignis gezeigt wird. Die ABoxen aus der Low-Level-Analyse der Bilder sind in Abbildung 5.2 dargestellt. Da in der Analysephase keine Unterschiede extrahiert werden konnten, muss eine korrekte Klassifizierung des Bildinhalts als Hochsprung bzw. Stabhochsprungereignis scheitern [1].

5.2 Evaluierung

Um die Güte von Klassifikatoren nach der Lernphase beurteilen zu können, werden die Klassifikatoren auf Testdaten angewendet und bewertet. Tatsächlich ist die Güte eines Klassifikators nicht nur vom Lernalgorithmus abhängig, sondern

auch von der Wahl der Trainingsdaten. Sind die Trainingsdaten nicht repräsentativ für die in tatsächlichen Anwendungen vorkommenden Daten, werden Klassifikatoren im allgemeinen fehlerhafte Klassifizierungen liefern.

Es gibt verschieden Maße um die Leistung von Klassifikatoren auf Mengen zu bewerten. Hier sollen nur zwei vorgestellt werden [4]:

Coverage ist die Anzahl der Elemente, für die eine Klassifizierung möglich ist bezogen auf die Gesamtmenge der Elemente. Dabei spielt keine Rolle, ob die Klassifizierung korrekt ist oder nicht.

Sei n_{covers} die Anzahl der Elemente einer Menge D, für die eine Klassifizierung möglich war. Dann ist

$$Coverage = \frac{n_{covers}}{|D|}.$$

Dabei ist $|D|$ die Kardinalität von D.

Accuracy Accuracy ist das Verhältinis aus korrekt klassifizieren Elementen zu allen klassifizierten Elementen.

Sei $n_{correct}$ die Menge der korrekt klassifizierten Elemente. Dann ist

$$Accuracy = \frac{n_{correct}}{n_{covers}}.$$

Kapitel 6

Lernen von Interpretationsregeln

Wenn man Abduktionsregeln für die Interpretation konstruieren möchte, dann ist als erstes zu beachten, dass ein high-level Konzept im allgemeine nicht durch eine einzige Regel, sondern nur durch eine Regelmenge erschöpfend beschrieben werden kann. Außerdem möchte man nicht nur für ein einziges Konzept eine Regelmenge aufstellen, sondern für beliebig viele.

Eine Regel ist auf eine ABox anwendbar, wenn für die Variablen im Regelkopf eine Abbildung auf die Individuen der ABox gefunden werden kann, so daß der Regelkopf wahr wird. Jede auf eine ABox anwendbare Regel kann Zusicherungen hypothetisieren und damit ein Interpretationsergebnis liefern. Das bedeutet, dass man Regelmengen für einzelne high-level Konzepte nicht isoliert betrachten kann, sondern man muss die Gesamtmenge der Regeln so konstruieren, dass durch eine geeignete Scoring-Funktion immer nur eine Regel ausgewählt wird, die die korrekten Zusicherungen und Individuen hypothetisiert.

Als Beispiel soll die Analyse-ABox in Abbildung 6.1 und die Regeln in Abbildung 6.2 dienen. Beide Regeln können auf die ABox angewendet werden und die resultierenden Aboxen sind in Abbildungen 6.3 und 6.4 dargestellt. Regel (b) hypothetisiert offensichtlich das falsche Ereignis Hochsprung (HighJump) und zusätzlich eine Querstange (HorizontalBar).

$$\begin{aligned} person_1 &: \texttt{Person} \\ javelin_1 &: \texttt{Javelin} \\ (person_1, javelin_1) &: \texttt{near} \end{aligned}$$

Figure 6.1: Analyse-Abox

In den hier vorgestellten Verfahren wird das Lernen von Interpretationsregeln als Klassifizierungsproblem aufgefasst. Die gesuchten Interpretationsregeln dienen als Klassifikator für eine Menge von Analyse-ABoxen.

$$
\begin{array}{lll}
\text{(a)} & \texttt{near}(X,Y) & \leftarrow \texttt{JavelinThrow}(Z), \texttt{hasParticipant}(Z,X), \\
 & & \texttt{hasPart}(Z,Y), \texttt{JavelinThrower}(X), \texttt{Javelin} \\
\text{(b)} & \texttt{near}(X,Y) & \leftarrow \texttt{HighJump}(Z), \texttt{hasParticipant}(Z,X), \texttt{hasPart}(Z,Y), \\
 & & \texttt{HighJumper}(X), \texttt{HorizontalBar}
\end{array}
$$

Figure 6.2: Regelmenge

$$
\begin{array}{rcl}
newInd_1 & : & \texttt{JavelinThrow} \\
person_1 & : & \texttt{JavelinThrower} \\
javelin_1 & : & \texttt{Javelin} \\
(newInd_1, person_1) & : & \texttt{hasParticipant} \\
(newInd_1, javelin_1) & : & \texttt{hasPart} \\
(person_1, javelin_1) & : & \texttt{near}
\end{array}
$$

Figure 6.3: ABox nach Anwendung von Regel (a)

$$
\begin{array}{rcl}
newInd_1 & : & \texttt{HighJump} \\
person_1 & : & \texttt{HighJumper} \\
javelin_1 & : & \texttt{Javelin} \\
newInd_2 & : & \texttt{HorizontalBar} \\
(newInd_1, person_1) & : & \texttt{hasParticipant} \\
(newInd_1,) & : & \texttt{hasPart} \\
(person_1, javelin_1) & : & \texttt{near}
\end{array}
$$

Figure 6.4: Abox nach Anwendung von Regel (b)

Um die Interpretationsregeln zu trainieren ist eine Trainingsmenge gegeben, die jeweils aus Paaren von Analyse-ABoxen $\mathcal{A}_A$ und Gold-Standard-ABoxen $\mathcal{A}_G$ besteht. Die Gold-Standard-ABoxen sind von Experten annotierte Analyse-ABoxen, die jeweils eine korrekte Interpretation darstellen.

Jeweils eine Zusicherung in den Gold-Standard-ABoxen wird als Hauptzusicherung identifiziert und anhand dieser Hauptzusicherung wird die ABox klassifiziert. Hauptzusicherungen entsprechen den Erklärungen, die man durch Abduktion für ausgewählte Beobachtungen erhalten möchte. In den Abbildungen 6.9 und 6.6 ist eine Analyse- bzw. eine entsprechende Gold-Standard-ABox dargestellt. Als Hauptzusicherung wird in diesem Fall die Konzeptzusicherung für die Instanz eines `JavelinThrow` gewählt und damit festgelegt, dass die ABoxen als Repräsentation eines Speerwurfs (engl. javelin throw) klassifiziert werden sollen.

$$
\begin{array}{rcl}
person_1 & : & \texttt{Person} \\
javelin_1 & : & \texttt{Javelin} \\
(person_1, javelin_1) & : & \texttt{near}
\end{array}
$$

Figure 6.5: Analyse-ABox

$$
\begin{array}{rcl}
newInd_1 & : & \texttt{JavelinThrow} \\
person_1 & : & \texttt{JavelinThrower} \\
javelin_1 & : & \texttt{Javelin} \\
(newInd_1, person_1) & : & \texttt{hasParticipant} \\
(newInd_1, javelin_1) & : & \texttt{hasPart} \\
(person_1, javelin_1) & : & \texttt{near}
\end{array}
$$

Figure 6.6: Gold-Standard-ABox

Ziel ist es nun eine Regelmenge zu finden, die für jede Analyse-ABox entweder die Hauptzusicherung der Gold-Standard-ABox hypothetisiert (korrekte Klassifikation) oder aber gar nichts hypothetisiert (keine Klassifikation). Eine Regelmenge kann eine ABox nicht klassifizieren, wenn keine Anwendung einer Regel zu einer eine konsistenten ABox führt.

Regelmengen, die ABoxen falsch klassifizieren, sind zu verwerfen, denn Interpretations-ABoxen sollten keine falsche Informationen/Zusicherungen enthalten. Eine Nicht-Klassifizierung ist akzeptabel, denn nach dem Prinzip der Open World Assumption bedeutet dass Fehlen einer Zusicherung nicht die Annahme der gegenteiligen Zusicherung, sondern einfach nur Nichtwissen.

Eine Analyse-Abox $\mathcal{A}_A$ wird vor der Interpretation in Bona-Fide-Zusicherungen[1] Γ_1, von denen angenommen wird, dass sie immer war sind, und Fiat-Zusicherungen[2] Γ_2, für die eine Erklärung Δ gesucht wird, aufgeteilt [7]. Wenn $\Sigma = (\mathcal{T}, \mathcal{A})$ ein Wissensbasis ist, dann gilt Folgendes:

$$\Sigma \cup \Gamma_1 \cup \Delta \models \Gamma_2$$

Die Aufteilung einer Analyse-Abox $\mathcal{A}_A$ in die Teilmengen Γ_1 und Γ_2 ist nicht Thema dieser Arbeit, stattdessen wird angenommen, dass die Fiat-Zusicherungen Γ_2 bereits identifiziert wurden.

Die vorgestellten Algorithmen berechnen nur Regelmengen mit einem vorher festgelegten Regelkopf. Der Regelkopf entspricht einer Beobachtung, für die man eine Erklärung sucht, also einer Fiat-Zusicherung aus Γ_2. Die ABoxen in der Trainingsmenge müssen jeweils die gewählte Beobachtung als Zusicherung enthalten, andernfalls ist ein Training nicht möglich. Wenn man Regelmengen für alle Zusicherungen in Γ_2, dann muss man die Algorithmen mehrfach ausführen.

6.1 Regelkandidaten

In beiden Algorithmen wird aus jeder Golden-Standard-ABox jeweils eine Menge von Regelkandidaten erzeugt aus denen eine geeignete Regelmenge für die

[1] bona fide (lat.): in gutem Glauben
[2] fiat (lat.): es werde ...

Interpretation konstruiert wird. Dabei werden aus Teilmengen der ABox-Zusicherungen folgendermaßen Regeln transformiert: Eine ABox oder Teilmenge einer ABox

$$\mathcal{A} = \{a_1, ...a_n\}$$

transformiert man in eine Abduktionsregel

$$\alpha(a_k) \leftarrow \alpha(a_1), ..., \alpha(a_{k-1}), \alpha(a_{k+1}), ..., \alpha(a_n)$$

indem auf die ABox-Zusicherungen $a_1, ..., a_n$ die Funkion α angewendet wird, die folgendermaßen definiert ist [3]:

$$\begin{array}{lll} \alpha(i:C) & =_{def} & C(x_i) \\ \alpha((i,j):R) & =_{def} & R(x_i, x_j) \end{array}$$

Die i, j sind ABox-Individuen und die x_i, x_j sind die Variablen der Regel. In Abbildung 6.7 ist eine Transformation einer ABox in eine Abduktionsregel dargestellt.

$$\begin{array}{rcl} person_1 & : & \texttt{Person} \\ javelin_1 & : & \texttt{Javelin} \\ (person_1, javelin_1) & : & \texttt{near} \end{array}$$

$$\texttt{near}(X, Y) \leftarrow \texttt{Person}(X), \texttt{Javelin}(Y)$$

Figure 6.7: Konstruktion einer Regel aus einer ABox

Der Regelkopf aller Regelkandidaten ist vorgegeben und entspricht jeweils der Beobachtung, für die man eine Erklärung erzeugen möchte. Die zu hypothetisierenden Erklärungen sind die Zusicherungen, die in ABox $\mathcal{A}_G$ aber nicht in ABox $\mathcal{A}_A$ vorkommen und werden auch als ABox-Differenz $\mathcal{A}_\Delta = \mathcal{A}_G - \mathcal{A}_A$ bezeichnet (siehe Kapitel 6.2 zur ABox-Differenz) . Diese Zusicherungen sind die korrekten Erklärungen für die Beobachtung im Regelkopf und werden deshalb in den Regelkörper aller Regelkandidaten aufgenommen. Das Regelatom, dass aus dem Hauptkonzept der Gold-Standard-ABox $\mathcal{A}_G$ konstruiert wurde, wird als Hauptatom bezeichnet.

Aus den übrigen Zusicherungen der Gold-Standard-ABox werden geeignete Teilmengen ausgewählt mit denen die verschiedenen Regelkandidaten ausgewählt werden. Dabei werden nur aus solchen Teilmengen Regelkandidaten erzeugt, deren resultierende Regeln die folgenden Bedingungen erfüllen:

1. Von jeder Variablen in der Regel muss ein Pfad zur Variablen des Hauptatoms konstruierbar sein. Dabei sind die Variablen die Knoten des Pfades und die Kanten werden aus den binären Rollenatomen konstruiert.

2. Für jede Variable existiert mindestens ein einwertiges Konzeptatom in der Regel.

Bedingung 1 entspricht der Annahme, dass das zu hypothetisierende Konzept (repräsentiert durch das Hauptkonzept bzw. Hauptatom) Aggregate anderer Konzepten sind, denn isolierte Konzepte können nicht teil eines Aggregats sein.

Bedingung 2 verhindert, dass bei der Abduktion Variablen durch beliebige Individuen ersetzt werden können.

6.2 ABox-Differenz

Eine ABox-Differenz-Funktion wird benötigt, um die Unterschiede zwischen einer Gold-Standard-ABox $\mathcal{A}_G$ und einer Analyse-ABox $\mathcal{A}_A$ zu berechnen.
Eine ABox-Differenz kann für zwei ABoxen konstruiert werden, die die gleiche TBox verwenden. Dabei wird nicht eine einfache Mengendifferenz der Zusicherungsmengen gebildet, sondern es werden die Axiome der TBox berücksichtigt, und dass in beiden ABoxen für identische Individuen jeweils unterschiedliche Namen verwendet werden können.

Die Definition der ABox-Differenz in [3] lautet wie folgt:

Definition: ABox-Differenz. Gegeben seien zwei ABoxen $\mathcal{A}$ und $\mathcal{A}'$, die dieselbe TBox nutzen. Die ABox-Differenz ist eine ABox $\Delta_{\mathcal{A},\mathcal{A}'}$ mit den folgenden Eigenschaften:

1. Es existiert eine Abbildung $\phi : inds(\mathcal{A}') \mapsto inds(\mathcal{A})$, so dass $\phi(\mathcal{A}' \cup \Delta_{\mathcal{A},\mathcal{A}'}) \models \mathcal{A}$
2. $\mathcal{A}' \cup \Delta_{\mathcal{A},\mathcal{A}} \not\models \bot$ (Konsistenz)
3. $\Delta_{\mathcal{A},\mathcal{A}}$ ist minimal (es gibt bezüglich $\subseteq$ keine kleinere ABox $\Delta'_{\mathcal{A},\mathcal{A}}$ mit denselben Eigenschaften)

Der ABox-Differenz Algorithmus wie er in RacerPro verwendet wird, ist wie folgt definiert [3]:

ABox-Differenz-Algorithmus Gegeben seien zwei ABoxen $\mathcal{A}$ und $\mathcal{A}$', die beide dieselbe TBox $\mathcal{T}$ nutzen. Die Differenz $\Delta_{\mathcal{A},\mathcal{A}'}$ wird folgendermaßen berechnet:

1. Transformiere $\mathcal{A} = \{a_1, ..., a_n\}$ in eine sogenannte grounded conjunctive query $q_{\mathcal{A}} = ans(x) \leftarrow \alpha(a_1) \wedge ... \wedge \alpha(a_n)$, mit $x = (x_{i_1}, ..., x_{i_m})$, $i_j \in inds(\mathcal{A})$
2. Rufe den Abductive Query Answering Service auf (siehe Kapitel 3.1) um die Menge der bevorzugten Hypothesen $\mathcal{H}$ zu berechnen.
3. $\Delta_{\mathcal{A},\mathcal{A}'} =_{def} \{hypothesizedAssertions(H) \mid H \in \mathcal{H}\}$
4. Gebe $\Delta_{\mathcal{A},\mathcal{A}'}$ aus

Dabei ist die Funktion α wie in Kapitel 6.1.

Beispiel: Die ABox-Differenz $\mathcal{A}_D = \mathcal{A}_G - \mathcal{A}_S$ aus der Gold-Standard-ABox $\mathcal{A}_G$ in Abbildung 6.8 und der Analyse-ABox $\mathcal{A}_A$ in Abbildung 6.9 ist in Abbildung 6.10 dargestellt. Die Differenz besteht aus vier Zusicherungen (`HighJump`, `HighJumper`, `hasParticipant`, `hasPart`) und einem Individuum ($highjump_1$).

$$\begin{aligned} highjump_1 &: \texttt{HighJump} \\ person_1 &: \texttt{HighJumper} \\ bar_1 &: \texttt{HorizontalBar} \\ (highjump_1, person_1) &: \texttt{hasParticipant} \\ (highjump_1, bar_1) &: \texttt{hasPart} \\ (person_1, bar_1) &: \texttt{near} \end{aligned}$$

Figure 6.8: Gold-Standard-ABox $\mathcal{A}_G$

$$\begin{aligned} person_1 &: \texttt{Person} \\ bar_1 &: \texttt{HorizontalBar} \\ (person_1, bar_1) &: \texttt{near} \end{aligned}$$

Figure 6.9: Analyse-ABox $\mathcal{A}_A$

$$\begin{aligned} highjump_1 &: \texttt{HighJump} \\ person_1 &: \texttt{HighJumper} \\ (highjump_1, person_1) &: \texttt{hasParticipant} \\ (highjump_1, bar_1) &: \texttt{hasPart} \end{aligned}$$

Figure 6.10: ABox-Differenz $\mathcal{A}_D = \mathcal{A}_G - \mathcal{A}_S$

6.3 Relaxed ABox Entailment

Relaxed Abox Entailment wird im zweiten Lernalgorithmus für Interpretationsregel verwendet, um die Regelmenge zu reduzieren. Hierzu wird der Körper der Regeln jeweils in eine ABox transformiert und dann geprüft, ob eine ABox $\mathcal{A}$ aus einer anderen ABox $\mathcal{A}'$ folgt: $\mathcal{A}' \models \mathcal{A}$ (siehe Kapitel **??**) .

Da aber bei der Transformation der Regeln die Individuennamen in den einzelnen ABoxen unterschiedlich gewählt werden können, benötigt man zusätzlich eine geeignete Abbildung zwischen den Individuennamen in den ABoxen. Relaxed ABox Entailment ($\rightsquigarrow$) versucht eine entsprechende Abbildung zu finden [3]:

Definition: Relaxed ABox Entailment. Gegeben seien zwei ABoxen $\mathcal{A}$ und $\mathcal{A}'$. Relaxed ABox Entailment $\mathcal{A}' \rightsquigarrow \mathcal{A}$ ist gegeben, wenn eine Abbildung $\phi : inds(\mathcal{A}') \mapsto inds(\mathcal{A})$ existiert, so dass $\phi(\mathcal{A}') \models \mathcal{A}$ gilt.

6.4 Algorithmus 1

Der erste Lernalgorithmus für Interpretationsregeln konstruiert aus den ABoxen der Trainingsmenge eine Menge von Regelkandidaten $\mathcal{R}_K$ und ermittelt die optimale Regelmenge durch Iteration über alle Teilmengen von $\mathcal{R}_K$.

Der Algorithmus 1 besteht aus den folgenden Schritten:

1. Ein Regelkopf wird festgelegt, für den eine optimale Regelmenge erzeugt werden soll.
2. Die Menge der Regelkandidaten wird konstruiert (siehe Abschnitt 6.1).
3. Jeder Regelkandidat wird einzeln auf jeweils alle Analyse-ABoxen der Trainingsmenge angewendet und die Scoring-Funktion für jedes ABox-Regelkandidaten-Paar ausgewertet.
4. Es wird die Potenzmenge der Regelkandidaten gebildet. Für jede Regelmenge in der Potenzmenge werden Coverage $\mathcal{M}_C$ und Accuracy $\mathcal{M}_A$ berechnet (siehe Abschnitt 5.2). Dabei wird eine ABox von einer Regelmenge korrekt klassifiziert, wenn die Regel mit dem besten Scoring-Wert, eine Instanz des korrekten Konzepts hypothetisiert. Eine Klassifizierung ist nicht möglich, wenn mehrere Regeln den gleichen Scoring-Wert haben, aber unterschiedlich klassifizieren.
5. Von den bewerteten Regelmengen wird eine nach folgenden Kriterien ausgewählt:
   (a) Regelmengen mit einer Accuracy $\mathcal{M}_A < 1$ werden verworfen.
   (b) Wähle Regelmengen mit maximaler Coverage $\mathcal{M}_C$.
   (c) Wähle Regelmengen mit minimaler Anzahl an Regeln.
   (d) Wähle eine Regelmenge mit minimaler Gesamtzahl an Regelatomen

6.5 Algorithmus 2

Der zweite Lernalgorithmus für Interpretationsregeln konstruiert aus den ABoxen der Trainingsmenge eine Menge von Regelkandidaten $\mathcal{R}_K$, die die Klassen der Trainingsmenge separieren.

Algorithmus 2 besteht aus folgenden Schritten:

1. Ein Regelkopf wird festgelegt, für den eine optimale Regelmenge erzeugt werden soll.
2. Die Menge der Regelkandidaten wird konstruiert (siehe Abschnitt 6.1).
3. Jeder Regelkandidat wird einzeln auf jeweils alle Analyse-ABoxen der Trainingsmenge angewendet und die Scoring-Funktion für jedes ABox-Regelkandidaten-Paar ausgewertet.

4. Ermittle den Minimalwert $SCORE_{MIN}$ und den Maximalwert $SCORE_{MAX}$ der Scoring-Werte aller ABox-Regelkandidaten-Paare. Konstruiere eine Wertemenge $\mathcal{S} = \{... - 1.5, -0.5, +0.5, +1.5...\}$, wobei für jedes Element $\mathcal{S}_i \in \mathcal{S}$ gilt $SCORE_{MIN} < \mathcal{S}_i < SCORE_{MAX}$.

5. Ermittle für alle Werte $\mathcal{S}_i \in \mathcal{S}$ jeweils eine Regelmenge $\mathcal{R}_i$. Eine Regel wird in die Menge $\mathcal{R}_i$ aufgenommen, wenn alle Scoring-Werte der Regel für korrekt klassifzierte ABoxen größer als $\mathcal{S}_i$ und alle Scoring-Werte für falsch klassifizierte ABoxen kleiner als $\mathcal{S}_i$ sind.

6. Verwerfe Regelmengen, die nicht Regeln für alle Klassen enthalten.

7. Berechne Coverage $\mathcal{M}_C$ und Accuracy $\mathcal{M}_A$ für jede Regelmenge $\mathcal{R}_i$.

8. Von den bewerteten Regelmengen wird eine nach folgenden Kriterien ausgewählt:

   (a) Regelmengen mit einer Accuracy $\mathcal{M}_A < 1$ werden verworfen.

   (b) Wähle Regelmengen mit maximaler Coverage $\mathcal{M}_C$.

9. Reduziere die Anzahl der Regeln in allen noch übrigen Regelmengen. Dazu werden die Regeln in ABoxen transformiert und dann für jede Regelmenge paarweise auf Relaxed ABox Entailment geprüft (siehe Abschnitt 6.3). Falls für zwei ABoxen $\mathcal{A}' \rightsquigarrow \mathcal{A}$ gilt, dann wird die Regel $\mathcal{R}'$, die zu $\mathcal{A}'$ transformiert wurde, verworfen. Allerdings nur, wenn dies Accuracy und Coverage nicht vermindert.

10. Wähle Regelmengen mit minimaler Anzahl an Regeln.

11. Wähle eine Regelmenge mit minimaler Gesamtzahl an Regelatomen.

6.6 Implementierung

Zur Implementierung der Algorithmen wird auf Funktionen von RacerPro[3] zurückgegriffen. RacerPro ist ein Reasoner, der durch Inferenzdienste in der Lage ist, implizites Wissen aus Wissensbasen abzuleiten. Ausserdem bietet RacerPro zusätzliche Dienste, wie die Berechnung einer ABox-Differenz und einen Abductive Query Answering Service an.

Auf die Funktionen von RacerPro kann über die Java API JRacer zugegriffenn werden. JRacer kommuniziert mit dem RacerPro Server über das TCP/IP Protokoll und kapselt die RacerPro-Funktionen in Java-Methoden-Aufrufe.

In Appendix B ist beispielhaft die Java-Klasse für den Aufruf der RacerPro-Funktion retrieve-with-explanation dargestellt, die den Abductive Query Answering Service von RacerPro aufruft.

[3] http://www.racer-systems.com

6.7 Ergebnisse

Die Algorithmen werden auf vier unterschiedliche Trainingsmengen angewendet. Die verwendeten ABoxen sind in Appendix A dargestellt. Als Beobachtung, für die jeweils eine Erklärung gefunden werden soll, wurde die Rolle `isNear` gewählt.

6.7.1 Test 1

Beide Algorithmen werde auf die ABoxen 1a, 1b und 2a angewendet. Diese repräsentieren Hochsprung- bzw. Speerwurfereignisse. Das Ergebnis für Algorithmus 1 sind zwei optimale Regelmengen für Interpretationsregeln die in der Abbildungen 6.11 dargestellt sind. Die Regelmengen unterscheiden sich nur durch eine Regel in der das Konzept `HorizontalBar` durch `Pillar` ersetzt wird. Der Coverage-Wert für beide Regelmengen beträgt $\mathcal{M}_C = 1$.

Algorithmus 2 liefert eine einzige Regelmenge, die aber eine Regel mehr enthält als die Regelmengen des Algorithmus 1. Der Coverage-Wert beträgt ebenfalls $\mathcal{M}_C = 1$.

$$
\begin{aligned}
\texttt{isNear}(X,Y) \quad &\leftarrow \quad \texttt{HighJump}(Z), \texttt{hasParticipant}(Z,X), \texttt{hasPart}(Z,Y),\\
&\qquad\quad \texttt{HighJumper}(X), \texttt{HorizontalBar}(Y)\\
\texttt{isNear}(X,Y) \quad &\leftarrow \quad \texttt{JavelinThrow}(Z), \texttt{hasParticipant}(Z,X), \texttt{hasPart}(Z,Y),\\
&\qquad\quad \texttt{JavelinThrower}(X), \texttt{Javelin}(Y)
\end{aligned}
$$

$$
\begin{aligned}
\texttt{isNear}(X,Y) \quad &\leftarrow \quad \texttt{HighJump}(Z), \texttt{hasParticipant}(Z,X), \texttt{hasPart}(Z,Y),\\
&\qquad\quad \texttt{HighJumper}(X), \texttt{Pillar}(Y)\\
\texttt{isNear}(X,Y) \quad &\leftarrow \quad \texttt{JavelinThrow}(Z), \texttt{hasParticipant}(Z,X), \texttt{hasPart}(Z,Y),\\
&\qquad\quad \texttt{JavelinThrower}(X), \texttt{Javelin}(Y)
\end{aligned}
$$

Figure 6.11: Test 1 / Algorithmus 1

$$
\begin{aligned}
\texttt{isNear}(X,Y) \quad &\leftarrow \quad \texttt{HighJump}(Z), \texttt{hasParticipant}(Z,X), \texttt{hasPart}(Z,Y),\\
&\qquad\quad \texttt{HighJumper}(X), \texttt{HorizontalBar}(Y)\\
\texttt{isNear}(X,Y) \quad &\leftarrow \quad \texttt{HighJump}(Z), \texttt{hasParticipant}(Z,X), \texttt{hasPart}(Z,Y),\\
&\qquad\quad \texttt{HighJumper}(X), \texttt{Pillar}(Y)\\
\texttt{isNear}(X,Y) \quad &\leftarrow \quad \texttt{JavelinThrow}(Z), \texttt{hasParticiapnt}(Z,X), \texttt{hasPart}(Z,Y),\\
&\qquad\quad \texttt{JavelinThrower}(X), \texttt{Javelin}(Y)
\end{aligned}
$$

Figure 6.12: Test 1 / Algorithmus 2

6.7.2 Test 2

Die Algorithmen werden auf die ABoxen 1a, 1b, 3a, und 3b angewendet, die Hochsprung- und Stabhochsprungereignisse beschreiben. Die ABoxen der zwei

Ereignisse haben eine größere Ähnlichkeit als die ABoxen der Hochsprung- und Speerwurfereignisse aus Test 1. So kommen sowohl in den Hochsprung-ABoxen als auch in den Stabhochsprung-ABoxen Individuen vom Typ `HorizontalBar` vor.

Trotz der Ähnlichkeit sind die beiden Klassen noch separierbar. Die Anzahl der Regelkombinationen, die eine korrekte Interpretation aller ABoxen liefern, ist allerdings beschränkt., so dass beide Algorithmen das gleiche Ergebnis zurückgeben. Es werden jeweis zwei Regelmengen mit je zwei Regeln und einem Coverage-Wert von $\mathcal{M}_C = 1$ (Abbildung 6.13) als optimale Lösung konstruiert.

$$
\begin{aligned}
\texttt{isNear}(X,Y) \quad &\leftarrow \quad \texttt{HighJump}(Z), \texttt{hasParticipant}(Z,X), \texttt{hasPart}(Z,Y), \\
&\qquad\quad \texttt{HighJumper}(X), \texttt{HorizontalBar}(Y) \\
\texttt{isNear}(X,Y) \quad &\leftarrow \quad \texttt{PoleVault}(Z), \texttt{hasParticipant}(Z,X), \texttt{hasPart}(Z,Y), \\
&\qquad\quad \texttt{PoleVaulter}(X), \texttt{HorizontalBar}(Y), \texttt{Pole}(W), \\
&\qquad\quad \texttt{isAdjacent}(W,Y)
\end{aligned}
$$

$$
\begin{aligned}
\texttt{isNear}(X,Y) \quad &\leftarrow \quad \texttt{HighJump}(Z), \texttt{hasParticipant}(Z,X), \texttt{hasPart}(Z,Y), \\
&\qquad\quad \texttt{HighJumper}(X), \texttt{Pillar}(Y) \\
\texttt{isNear}(X,Y) \quad &\leftarrow \quad \texttt{PoleVault}(Z), \texttt{hasParticipant}(Z,X), \texttt{hasPart}(Z,Y), \\
&\qquad\quad \texttt{PoleVaulter}(X), \texttt{HorizontalBar}(Y), \texttt{Pole}(W), \\
&\qquad\quad \texttt{isAdjacent}(W,Y)
\end{aligned}
$$

Figure 6.13: Test 2 / Algorithmus 1 & 2

6.7.3 Test 3

In diesem Test werden wieder ABoxen von Hochsprung- und Stabhochsprungereignissen verwendet (1a, 1b, 3c, 3d). Diesmal wird aber eine der Hochsprung-ABoxen so gewählt, dass sie nur die Objekte vom Typ `Person` und `HorizontalBar` und kein Objekt vom Typ `Pole` enthält und damit nicht von einem Hochsprungereignis zu unterscheiden ist (siehe Abbildung **??**).

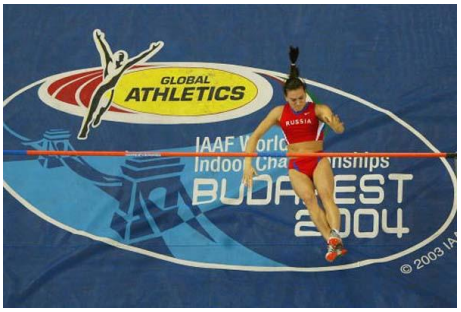
Das Ergebnis ist wie erwartet: Beide Algorithmen sind nicht in der Lage eine Regelmenge zu finden.

6.7.4 Test 4

In letzten Test sollen drei Klassen von ABoxen interpretiert werden (Hochsprung, Stabhochsprung und Speerwurf). Die verwendeten ABoxen für die Trainingsmenge sind 1a, 1b, 2a, 3a und 3b.

Algorithmus 1 liefert wieder zwei Regelmengen, diesmal aber mit je drei Regeln und einem Coverage-Wert unter 1 von $\mathcal{M}_C = 0,625$ (Abbildung 6.15). Dies bedeutet, dass nicht alle ABoxen klassifiziert werden konnten.

(a) Hochsprung



(b) Stabhochsprung

Figure 6.14: Bilder mit identischen Analyse-ABoxen

Algorithmus 2 liefert wieder kein Ergebnis zurück, da keine Regelmenge gefunden werden kann, die alle ABoxen korrekt klassifiziert.

$$\begin{aligned}
\texttt{isNear}(X,Y) &\leftarrow \texttt{HighJump}(Z), \texttt{hasParticipant}(Z,X), \texttt{hasPart}(Z,Y),\\
&\quad \texttt{HighJumper}(X), \texttt{HorizontalBar}(Y)\\
\texttt{isNear}(X,Y) &\leftarrow \texttt{JavelinThrow}(Z), \texttt{hasParticiapnt}(Z,X),\\
&\quad \texttt{hasPart}(Z,Y), \texttt{JavelinThrower}(X), \texttt{Javelin}(Y)\\
\texttt{isNear}(X,Y) &\leftarrow \texttt{PoleVault}(Z), \texttt{hasParticipant}(Z,X), \texttt{hasPart}(Z,Y),\\
&\quad \texttt{PoleVaulter}(X), \texttt{HorizontalBar}(Y), \texttt{Pole}(W),\\
&\quad \texttt{isAdjacent}(W,Y)
\end{aligned}$$

$$\begin{aligned}
\texttt{isNear}(X,Y) &\leftarrow \texttt{HighJump}(Z), \texttt{hasParticipant}(Z,X), \texttt{hasPart}(Z,Y),\\
&\quad \texttt{HighJumper}(X), \texttt{Pillar}(Y)\\
\texttt{isNear}(X,Y) &\leftarrow \texttt{JavelinThrow}(Z), \texttt{hasParticiapnt}(Z,X),\\
&\quad \texttt{hasPart}(Z,Y), \texttt{JavelinThrower}(X), \texttt{Javelin}(Y)\\
\texttt{isNear}(X,Y) &\leftarrow \texttt{PoleVault}(Z), \texttt{hasParticipant}(Z,X), \texttt{hasPart}(Z,Y),\\
&\quad \texttt{PoleVaulter}(X), \texttt{HorizontalBar}(Y), \texttt{Pole}(W),\\
&\quad \texttt{isAdjacent}(W,Y)
\end{aligned}$$

Figure 6.15: Test 3 / Algorithmus 1

Kapitel 7

Zusammenfassung und Auswertung

Im Rahmen dieser Arbeit wurden zwei Algorithmen zum Lernen von Interpretationsregeln durch Abduktion entwickelt und angewendet. Dabei wurden die Menge der Interpretationsregeln als Klassifikator für mehrere Klassen von Konzepten aufgefasst.

Es hat sich gezeigt, dass Algorithums 1 eine geringere Anforderungen an die Separierbarkeit der unterschiedlichen Konzepte hat und auch Regelmengen konstruiert, die weniger Regeln enthalten als die von Algorithmus 2 konstruierten Regeln.

Allerdings hat Algorithmus 1 eine Zeit- und Raumkomplexität von $\mathcal{O}(2^n)$, wobei n die Anzahl der ABoxen der Trainingsmenge und ist somit auf größere Trainingsmengen bzw. Trainingsmengen für viele unterschiedliche Klassen von Konzepten nicht anwendbar ist. Algorithmus 2 dagegen hat lineare Komplexität, stellt aber höhere Anforderungen an die Separierbarkeit der ABoxen in verschiedene Klassen und liefert nicht immer minimale Regelmengen zurück.

Beide Algorithmen sind auf Analyseverfahren angewiesen, die in der Analysephase ausreichend aussagekräftige Strukturen aus den Multimedia-Dokumenten extrahieren, um die einzelnen ABoxen klassifizieren zu können. Ist diese Voraussetzung nicht gegeben, können keine Interpretationsregeln konstruiert werden.

Der Nachteil beider Algorithmen ist die fehlende Anpassungmöglichkeit der Regelmenge nach Abschluss der Lernphase. Werden später neue Zusicherungen zur ABox oder werden neue high-level Konzepte zur TBox hinzugefügt, die nicht korrekt interpretiert werden, dann muss die Regelmenge verworfen und neu gelernt werden. Besser wäre ein Verfahren, dass in einem solchen Fall die alte Regelmenge anpasst. Also neue Regeln hinzufügt und/oder alte Regeln entfernt.

Anhang A

ABoxen

Hier sind die Analyse- und Gold-Standard-ABoxen mit zugehörigem Bild dargestellt, die zum Testen der Algorithmen verwendet wurdden. Auf der linken Seite befindet sich immer die Analyse-ABox und auf der rechten Seite die Gold-Standard-ABox.

A.1 Hochsprung

Bild 1a

$person_1$	:	`Person`
bar_1	:	`HorizontalBar`
$pillar_1$	:	`Pillar`
$(person_1, bar_1)$	:	`isNear`
$(pillar_1, bar_1)$	:	`isAdjacent`
$(pillar_1, person_1)$	:	`isLeft`

$highjump_1$	:	`HighJump`
$person_1$	:	`HighJumper`
bar_1	:	`HorizontalBar`
$pillar_1$	:	`Pillar`
$(highjump_1 bar_1)$	:	`hasPart`
$(highjump_1 person_1)$	:	`hasParticipant`
$(person_1, bar_1)$	:	`isNear`
$(pillar_1, bar_1)$	:	`isAdjacent`
$(pillar_1, person_1)$	:	`isLeft`

Bild 1b

$person_1$	:	`Person`
bar_1	:	`HorizontalBar`
$pillar_1$	:	`Pillar`
$(person_1, bar_1)$	:	`isOverlapping`
$(pillar_1, bar_1)$	:	`isNear`
$(pillar_1, person_1)$	:	`isBehind`

$highjump_1$	:	`HighJump`
$person_1$	:	`HighJumper`
bar_1	:	`HorizontalBar`
$pillar_1$	:	`Pillar`
$(highjump_1 bar_1)$	:	`hasPart`
$(highjump_1 person_1)$	:	`hasParticipant`
$(person_1, bar_1)$	:	`isOverlapping`
$(pillar_1, bar_1)$	:	`isNear`
$(pillar_1, person_1)$	:	`isBehind`

Bild 1c

$person_1$	:	`Person`
bar_1	:	`HorizontalBar`
$(person_1, bar_1)$	:	`isNear`

$highjump_1$	:	`HighJump`
$person_1$	:	`HighJumper`
bar_1	:	`HorizontalBar`
$(highjump_1 bar_1)$	:	`hasPart`
$(highjump_1 person_1)$	:	`hasParticipant`
$(person_1, bar_1)$	:	`isNear`

Bild 1d

$person_1$	:	`Person`
bar_1	:	`HorizontalBar`
$(person_1, bar_1)$	:	`isOverlapping`
$(person_1, bar_1)$	:	`isNear`

$highjump_1$	:	`HighJump`
$person_1$	:	`HighJumper`
bar_1	:	`HorizontalBar`
$(highjump_1 bar_1)$	:	`hasPart`
$(highjump_1 person_1)$	:	`hasParticipant`
$(person_1, bar_1)$	:	`isOverlapping`
$(person_1, bar_1)$	:	`isNear`

A.2 Speerwurf

Bild 2a

$person_1$	:	`Person`
$javelin_1$	:	`Javelin`
$(person_1, javelin_1)$	:	`isNear`

$javelinthrow_1$	:	`JavelinThrow`
$person_1$	:	`JavelinThrower`
$javelin_1$	:	`Javelin`
$(javelinthrow_1 javelin_1)$	:	`hasPart`
$(javelinthrow_1 person_1)$	:	`hasParticipant`
$(person_1, javelin_1)$	:	`isNear`

A.3 Stabhochsprung

Bild 3a

$person_1$	:	`Person`
bar_1	:	`HorizontalBar`
$pole_1$	:	`Pole`
$(person_1, bar_1)$	:	`isNear`
$(pole_1, bar_1)$	:	`isAdjacent`
$(pole_1, person_1)$	:	`isLeft`

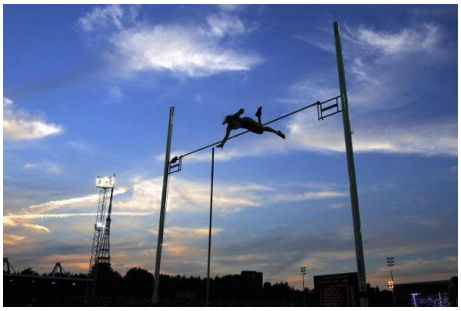
$polevault_1$	:	`PoleVault`
$person_1$	:	`PoleVaulter`
bar_1	:	`HorizontalBar`
$pole_1$	:	`Pole`
$(polevault_1, person_1)$	:	`hasParticipant`
$(polevault_1, bar_1)$	:	`hasPart`
$(person_1, bar_1)$	:	`isNear`
$(pole_1, bar_1)$	:	`isAdjacent`
$(pole_1, person_1)$	:	`isLeft`

Bild 3b

$person_1$	:	`Person`
$pole_1$	:	`Pole`
$(person_1, pole_1)$	:	`isNear`

$polevault_1$	:	`PoleVault`
$person_1$	:	`PoleVaulter`
$pole_1$	:	`Pole`
$(polevault_1, person_1)$	:	`hasParticipant`
$(polevault_1, bar_1)$	:	`hasPart`
$(person_1, pole_1)$	:	`isNear`

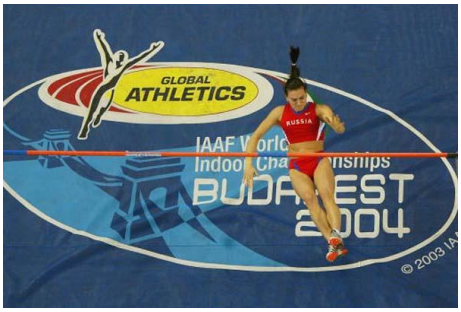
Bild 3c



$person_1$	:	`Person`
bar_1	:	`HorizontalBar`
$pole_1$	:	`Pole`
$pillar_1$	:	`Pillar`
$(person_1, bar_1)$	:	`isNear`
$(pole_1, bar_1)$	:	`isAdjacent`
$(pillar_1, person_1)$	:	`isLeft`

$polevault_1$	:	`PoleVault`
$person_1$	:	`PoleVaulter`
bar_1	:	`HorizontalBar`
$pole_1$	:	`Pole`
$pillar_1$	:	`Pillar`
$(polevault_1, person_1)$	:	`hasParticipant`
$(polevault_1, bar_1)$	:	`hasPart`
$(person_1, bar_1)$	:	`isNear`
$(pole_1, bar_1)$	:	`isAdjacent`
$(pillar_1, person_1)$	:	`isLeft`

Bild 3d



$person_1$	:	`Person`
bar_1	:	`HorizontalBar`
$(person_1, bar_1)$	:	`isNear`

$polevault_1$	:	`PoleVault`
$person_1$	:	`PoleVaulter`
bar_1	:	`HorizontalBar`
$(polevault_1, person_1)$	:	`hasParticipant`
$(polevault_1, bar_1)$	:	`hasPart`
$(person_1, bar_1)$	:	`isNear`

Anhang B

Java Code

B.1 RetrieveWithExplanation.java

```
package de.tuhh.sts.lmir;

import org.apache.log4j.Logger;

import com.racersystems.jracer.RacerClient;
import com.racersystems.jracer.RacerList;
import com.racersystems.jracer.RacerLiteral;
import com.racersystems.jracer.RacerNumber;
import com.racersystems.jracer.RacerResult;
import com.racersystems.jracer.RacerSymbol;

public class RetrieveWithExplanation extends RacerFunction
{
    /**
     * @param racer
     */
    public RetrieveWithExplanation(RacerClient racer)
    {
        this(racer, null);
    }

    /**
     * @param racer
     * @param observation
     */
    public RetrieveWithExplanation(RacerClient racer, String
        observation)
    {
        this.racer = racer;
        this.observation = observation;
    }

    /**
     * @return
```

```
     * @throws Exception
     */
    public RacerResult execute() throws Exception
    {
        String arg = "() " + observation + " :final-consistency
            -checking-p t :show-score-p t";

        RacerResult result = racer.retrieveWithExplanationM$(
            arg);
        parseResult(result);

        return (result);
    }

    /**
     * @return the score
     */
    public Score getScore()
    {
        return score;
    }

    /**
     * @param result
     * @throws Exception
     */
    @SuppressWarnings("unchecked")
    private void parseResult(RacerResult result) throws
        Exception
    {
        // initialization
        score = new Score();

        RacerList<RacerResult> list = (RacerList<RacerResult>)
            result;
        for (RacerResult res : list)
        {
            if (res instanceof RacerList)
            {
                RacerList list2 = (RacerList) res;
                parseTuple(list2);
            }
        }
    }

    /**
     * @param list
     * @throws Exception
     */
    @SuppressWarnings("unchecked")
    private void parseTuple(RacerList<RacerResult> list)
        throws Exception
```

```
{
   for (RacerResult res : list)
   {
      if (res instanceof RacerList)
      {
         RacerList<RacerResult> list2 = (RacerList<
            RacerResult>) res;
         for (RacerResult res2 : list2)
         {
            if (res2 instanceof RacerList)
            {
               RacerList list3 = (RacerList) res2;
               RacerResult res3 = (RacerResult) list3.
                  value.get(0);
               if (res3 instanceof RacerSymbol)
               {
                  RacerSymbol symbol = (RacerSymbol) res3;
                  if (symbol.getValue().equals(":score"))
                     parseScore(list3);
               }
            }
         }
      }
   }
}

/**
 * @param list
 * @throws Exception
 */
@SuppressWarnings("unchecked")
private void parseScore(RacerList<RacerResult> list)
   throws Exception
{
   RacerSymbol symScore = (RacerSymbol) list.value.get(1);
   score.setScore(Integer.parseInt(symScore.getValue()));

   RacerList list2 = (RacerList) list.value.get(3);
   RacerList oldInds = (RacerList) list2.value.get(0);
   RacerList entailedAssertions = (RacerList) list2.value.
      get(1);
   RacerList newInds = (RacerList) list2.value.get(2);
   RacerList hypothesizedAssertions = (RacerList) list2.
      value.get(3);

   parseOldInds(oldInds);
   parseEntailedAssertions(entailedAssertions);
   parseNewInds(newInds);
   parseHypothesizedAssertions(hypothesizedAssertions);
}

/**
```

```
     * @param list
     */
    private void parseOldInds(RacerList<RacerLiteral> list)
    {
        int i = 0;
        for (RacerLiteral literal : list)
        {
            if (0 == i)
            {
                i++;
                continue;
            }
            else if (1 == i)
            {
                i++;
                score.setOldIndCount(((RacerNumber) literal).
                    getValue().intValue());
            }
            else
            {
                score.addOldInds(((RacerSymbol) literal).getValue
                    ());
            }
        }
    }

    /**
     * @param list
     */
    private void parseNewInds(RacerList<RacerLiteral> list)
    {
        int i = 0;
        for (RacerLiteral literal : list)
        {
            if (0 == i)
            {
                i++;
                continue;
            }
            else if (1 == i)
            {
                i++;
                score.setNewIndCount(((RacerNumber) literal).
                    getValue().intValue());
            }
            else
            {
                score.addNewInds(((RacerSymbol) literal).getValue
                    ());
            }
        }
    }
```

```
/**
 * @param list
 * @throws Exception
 */
private void parseEntailedAssertions(RacerList<RacerResult
    > list) throws Exception
{
    int i = 0;
    for (RacerResult result : list)
    {
        if (0 == i)
        {
            i++;
            continue;
        }
        else if (1 == i)
        {
            i++;
            score.setEntailedAssertsCount(((RacerNumber)
                result).getValue().intValue());
        }
        else
        {
            score.addEntailedAsserts(parseAssertion((
                RacerList<RacerSymbol>) result, ":instance",
                ":related"));
        }
    }
}

/**
 * @param list
 * @throws Exception
 */
private void parseHypothesizedAssertions(RacerList<
    RacerResult> list) throws Exception
{
    int i = 0;
    for (RacerResult result : list)
    {
        if (0 == i)
        {
            i++;
            continue;
        }
        else if (1 == i)
        {
            i++;
            score.setHypothesizedAssertsCount(((RacerNumber)
                result).getValue().intValue());
        }
```

```
            else
            {
                score.addHypothesizedAsserts(parseAssertion((
                    RacerList<RacerSymbol>) result, "instance", "
                    related"));
            }
        }
    }

    /**
     * @param list
     * @param instanceMarker
     * @param roleMarker
     * @return
     * @throws Exception
     */
    private Assertion parseAssertion(RacerList<RacerSymbol>
        list, String instanceMarker, String roleMarker) throws
        Exception
    {
        if (list.value.get(0).getValue().equals(instanceMarker)
            )
        {
            ConceptAssertion con = new ConceptAssertion();
            con.setIndividual(list.value.get(1).getValue());
            con.setConceptId(list.value.get(2).getValue());
            return con;
        }
        else if (list.value.get(0).getValue().equals(roleMarker
            ))
        {
            RoleAssertion role = new RoleAssertion();
            role.setAntecessorInd(list.value.get(1).getValue());
            role.setSuccessorInd(list.value.get(2).getValue());
            role.setRoleId(list.value.get(3).getValue());
            return role;
        }
        else
            throw new RuleLearnerException("Parse Error:
                RacerList not recognized as instance or role
                assertion (" + list.value.get(0).getValue() + ")
                ");
    }

    /**
     * @return the observation
     */
    public String getObservation()
    {
        return observation;
    }
```

```
    /**
     * @param observation
     *          the observation to set
     */

    public void setObservation(String observation)
    {
       this.observation = observation;
    }

    private static final Logger LOGGER = Logger.getLogger(
        RetrieveWithExplanation.class);
    private RacerClient racer = null;
    private String observation = null;
    private Score score = null;
}
```

Abbildungsverzeichnis

Literaturverzeichnis

[1] Ethem Alpaydin, *Introduction to machine learning (adaptive computation and machine learning)*, The MIT Press, 2004.

[2] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider (eds.), *The description logic handbook: Theory, implementation, and applications*, Cambridge University Press, 2003.

[3] BOEMIE Team Arbeitsbereich Softwaresysteme TU Hamburg-Harburg, 2009. Computing ABox Differences.

[4] Jiawei Han and Micheline Kamber, *Data mining: Concepts and techniques*, Morgan Kaufmann, 2000.

[5] R. Möller and B. Neumann, *Ontology-based Reasoning Techniques for Multimedia Interpretation and Retrieval*, Semantic multimedia and ontologies : Theory and applications, 2008.

[6] S. Espinosa Peraldi, A. Kaya, S. Melzer, and R. Möller, *On ontology based abduction for text interpretation*, Proc. of 9th international conference on intelligent text processing and computational linguistics (cicling-2008), 2008, pp. 194–205.

[7] S. Espinosa Peraldi, A. Kaya, S. Melzer, R. Möller, and M. Wessel, *Multimedia Interpretation as Abduction*, Proc. dl-2007: International workshop on description logics, 2007.