

Universität Leipzig  
Fakultät für Mathematik und Informatik  
Institut für Informatik

# Semantic Chess

Bachelorarbeit

Leipzig, Februar 2018

vorgelegt von  
Daug, Jörn-Henning  
Studiengang: Informatik B.Sc.

**Betreuende Hochschullehrer:**

**Prof. Dr. Axel-C. Ngonga Ngomo**

**Dr. Ricardo Usbeck**

Universität Leipzig, Institut für Informatik, Betriebliche Informationssysteme

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>8</b>
<b>2</b>	<b>Stand der Technik</b>	<b>11</b>
2.1	Definition von Question Answering . . . . .	11
2.2	Definition von Linked Data, RDF und SPARQL . . . . .	12
2.2.1	Linked Data . . . . .	13
2.2.2	RDF . . . . .	13
2.2.3	SPARQL . . . . .	14
2.3	Funktionsweise und Systeme im Question Answering . . . . .	14
2.3.1	Einteilung nach Unger et al. (2014) . . . . .	16
2.3.2	Einteilung nach Mahender et al. (2016) . . . . .	20
2.4	Template-basiertes Question Answering . . . . .	21
2.5	Herausforderungen im Semantic Question Answering . . . . .	24
2.5.1	Lexikalische Lücke . . . . .	24
2.5.2	Ambiguität . . . . .	25
2.5.3	Mehrsprachigkeit . . . . .	25
2.5.4	Komplexe Anfragen . . . . .	26
2.5.5	Verteiltes Wissen . . . . .	26
2.5.6	Prozedurale, zeitliche und räumliche Fragen . . . . .	26
2.5.7	Templates . . . . .	27
2.6	Benchmark . . . . .	27

2.7	Verwandte Arbeiten und Ansätze im Bereich Schach . . . . .	29
2.7.1	Schach-Question-Answering . . . . .	29
2.7.2	Schach-Ontologien . . . . .	31
2.7.3	Schachpositionen als Problem des Information Retrievals . . . . .	33
2.7.4	Weitere Erwähnungen . . . . .	33
<b>3</b>	<b>Herausforderungen im Schach-Question-Answering</b>	<b>36</b>
3.1	Lexikalische Lücke . . . . .	36
3.2	Ambiguität . . . . .	38
3.3	Mehrsprachigkeit . . . . .	39
3.4	Komplexe Anfragen . . . . .	39
3.5	Verteiltes Wissen . . . . .	40
3.6	Prozedurale, zeitliche und räumliche Fragen . . . . .	40
3.7	Templates . . . . .	41
<b>4</b>	<b>Ansatz</b>	<b>43</b>
4.1	Logischer Aufbau . . . . .	43
4.2	Implementierung . . . . .	52
4.3	Frontend . . . . .	53
4.4	Backend . . . . .	54
4.4.1	Controller . . . . .	55
4.4.2	Service . . . . .	56
4.4.3	Database . . . . .	56
4.4.4	Annotation . . . . .	58
4.4.5	Parser . . . . .	59
4.4.6	Converter . . . . .	67
4.5	Datenbank . . . . .	67
4.6	Lizenzen . . . . .	68

4.7	Urheberrecht . . . . .	69
<b>5</b>	<b>Evaluation und Benchmark</b>	<b>70</b>
5.1	Trainingsdaten . . . . .	71
5.2	Testdaten . . . . .	75
5.3	Evaluierung der Herausforderungen im Schach-Question-Answering . . . .	78
<b>6</b>	<b>Hinweise zur Weiterentwicklung</b>	<b>82</b>
6.1	Frontend . . . . .	82
6.2	Backend . . . . .	83
6.3	Datenbank . . . . .	84
<b>7</b>	<b>Zusammenfassung</b>	<b>85</b>
	<b>Literatur</b>	<b>87</b>
<b>A</b>	<b>UML Diagramm Softwaretechnik-Praktikum 2015</b>	<b>94</b>
<b>B</b>	<b>Benchmark - Auswertungen</b>	<b>96</b>
<b>C</b>	<b>Schachvokabular</b>	<b>101</b>

# Tabellenverzeichnis

3.1	Beispiele für den ECO Code . . . . .	37
3.2	Überblick zu den Fragentypen . . . . .	41
4.1	Semantische Repräsentation benannter Entitäten . . . . .	48
4.2	Semantische Repräsentation unbenannter Entitäten . . . . .	48
4.3	Resultat der Beispielfrage . . . . .	51
4.4	Beispiel einer <i>Token</i> -Liste . . . . .	59
4.5	Beispiel einer erweiterten <i>Token</i> -Liste . . . . .	61
4.6	Frontend: Lizenzen der verwendeten Software . . . . .	68
4.7	Backend: Lizenzen der verwendeten Software . . . . .	69
4.8	Datenbank: Lizenzen der verwendeten Software . . . . .	69
5.1	Benchmark der Testdaten . . . . .	76
5.2	Bearbeitungszeiten der Testfragen . . . . .	77
5.3	Einfluss der Herausforderungen auf Semantic Chess . . . . .	81
B.1	Benchmark für Trainingsdaten mit <i>substringWithDistanceFallback()</i> . . .	97
B.2	Benchmark für Trainingsdaten mit <i>distanceEntities()</i> . . . . .	98
B.3	Benchmark für Trainingsdaten mit <i>subStringEntities()</i> . . . . .	99
B.4	Benchmark für Trainingsdaten mit <i>regexEntities()</i> . . . . .	100
C.1	Schachvokabular, zur Beantwortung der 50 Trainingsfragen . . . . .	102

# Abbildungsverzeichnis

2.1	Komponenten eines QAS . . . . .	15
2.2	Syntaktische Analyse mit semantischen Repräsentationen für jede Phrase . . . . .	16
2.3	Arbeitsschritte des QAS Treo . . . . .	18
2.4	Antwortgraph von Treo . . . . .	19
2.5	Überblick des Template-basierten SPARQL Query Generators . . . . .	22
2.6	Semantische Repräsentation einer Frage . . . . .	23
2.7	Visualisierung der Ontologie aus [3] . . . . .	31
2.8	Ontologie von Krisnadhi et.al . . . . .	32
4.1	Flussdiagramm von Semantic Chess . . . . .	44
4.2	Graph für das Beispiel " <i>Which player with black defeated Howard Staunton in December 1843?</i> " . . . . .	49
4.3	UML-Diagramm von Semantic Chess . . . . .	52
4.4	Beispiel der Suchseite und des Suchergebnisses . . . . .	54
4.5	UML-Diagramm des Pakets: Controller . . . . .	55
4.6	UML-Diagramm des Pakets: Service . . . . .	56
4.7	UML-Diagramm des Pakets: Database . . . . .	57
4.8	UML-Diagramm des Pakets: Annotation . . . . .	58
4.9	UML-Diagramm des Pakets: Parser . . . . .	60
A.1	UML Diagramm Softwaretechnik-Praktikum 2015 . . . . .	95

# Listings

2.1	Tripel-Beispiel . . . . .	13
2.2	Beispiel eines Tripel-Datensatz (Quelle: [64], Abschnitt 2.1) . . . . .	14
2.3	Beispiel eines SPARQL Queries (Quelle: [64], Abschnitt 2.1) . . . . .	14
2.4	Template-basierte SPARQL Query (Quelle: [18], S.642) . . . . .	21
4.1	Grundform eines manuell angelegten SPARQL Template, Sequenz: 010 . . . . .	50
4.2	Grundform eines manuell angelegten SPARQL Template mit Vereinigung, Sequenz: 021 . . . . .	50
4.3	Template für die Beispielfrage, Sequenz: 220 . . . . .	50
4.4	SPARQL Query für die Beispielfrage . . . . .	51
4.5	Beispiel: Registrierung der Seite "game" . . . . .	55
4.6	SPARQL ohne Beachtung der Flags für Farben und Elo . . . . .	63
4.7	SPARQL mit Beachtung der Flags für Farben und Elo . . . . .	63
4.8	SPARQL mit Vereinigung . . . . .	63
4.9	<i>regex</i> für Turm gegen Turm und Bauern . . . . .	64
4.10	<i>regexEntities()</i> . . . . .	65
4.11	<i>subStringEntities()</i> . . . . .	66
4.12	SPARQL Query für die Frage: " <i>When did Wilhelm Steinitz win most often with an ELO over 2500.</i> " . . . . .	66
5.1	Beispiel VALUES-Klausel: "World Championship" . . . . .	73

# Kapitel 1

## Einleitung

Mit der Entwicklung der ersten Computer kam der Wunsch bei Wissenschaftlern auf, mit den Maschinen in natürlicher Sprache zu kommunizieren (vgl. [30]). Im Film wurde das schnell Realität. Der Bordcomputer in der Fernsehserie *“Star Trek”* avancierte zur perfekten Frage-Antwort-Maschine (engl., *Question Answering System*) (vgl. [6]). In der Wissenschaft begannen die ersten Versuche in den sechziger Jahren. Robert Simmons stellte 1965 in einer Ausarbeitung 15 englischsprachige Question-Answering-Systeme vor, die versuchten, mit ihrem jeweiligen Ansatz, Fragen aus einem Themenbereich zu beantworten (vgl. [59], S.53). Darunter befindet sich das System *BASEBALL*, das 1963 von Bert Green et al. entwickelt wurde und Fragen zum Baseball-Sport beantworten kann (vgl. [59], S.55). Das erste Question-Answering-System, das in der Öffentlichkeit größere Bekanntheit erreichte, war *Watson*, das Bestandteil des DeepQA-Projekts von IBM ist (vgl. [30]). In der amerikanischen Quizsendung *Jeopardy!* schlug es die bisher besten menschlichen Konkurrenten und beantwortete die meisten Fragen (vgl. [35]). Spätestens mit der Entwicklung von Amazon Alexa, Google Home, Siri von Apple und Cortana von Microsoft sind die Frage-Antwort-Maschinen auch in die Haushalte eingezogen.

Diese Arbeit beschäftigt sich mit dem *Question Answering* im Bereich Schach. Dafür wurde das System *Semantic Chess* entwickelt. Das Ziel ist es, Schachspielern die Möglichkeit zu geben in natürlicher Sprache nach Partien und spezifischen Informationen zu suchen, ohne dabei ein größeres Hintergrundwissen über die Handhabung einer Datenbank zu besitzen. Zunächst beschäftigt sich die Arbeit mit den verschiedenen Ansätzen und den Herausforderungen auf dem Gebiet des *Question Answering*, um diese dann auf Schach zu übertragen. Mit den gewonnenen Erkenntnissen kann eine Frage-Antwort-Maschine entwickelt werden, das den Template-(Vorlage)-basierten Ansatz verfolgt. Um eine Frage zu



beantworten, durchläuft es verschiedene Prozesse und Zustände. Im ersten Schritt konvertiert es die PGN-Dateien (Dateiformat für Schachpartien) in das RDF-Format. Das Format besteht aus Tripeln, die aus Subjekt, Prädikat und Objekt aufgebaut sind (vgl. [37]). Die Abfragesprache *SPARQL* (vgl. [64]) kann diese Datenbasis anschließend befragen. Auf dieser Grundlage wird eine Suchfunktion implementiert. *Semantic Chess* analysiert eine natürlichsprachige Frage, erkennt daraus Entitäten mit Hilfe eines Schachvokabulars, regulären Ausdrücken und der Analyse-Software *coreNLP* der Stanford Universität (vgl. [22]). Aus den gefundenen Entitäten bildet das System eine semantische Repräsentation in Form von Tripeln und wählt mit ihr ein Template aus. Das System ersetzt im Anschluss die Platzhalter in der Vorlage und erzeugt dadurch eine SPARQL-Anfrage, mit der es die Datenbasis befragt. Zum Schluss präsentiert das System die Antwort dem Fragensteller. Ein Benchmark, der eigens für diese Arbeit entwickelt wurde und sich an der *QALD challenge* (vgl. [50]) orientiert, evaluiert anschließend *Semantic Chess*. Anhand von 50 verschiedenen und eigenständig erstellten Schachfragen wurde das System zunächst entwickelt. Am Ende der Implementierung konnte es die meisten davon beantworten. Im Anschluss musste das System zehn Testfragen beantworten, die von Nutzern eines Schachforums stammen. Aus den daraus resultierenden Stärken und Schwächen von *Semantic Chess* beschreibt die Arbeit Hinweise und Informationen für eine Weiterentwicklung des Systems. Der gesamte Code ist auf <https://github.com/dice-group/semanticchess> zu finden.

## Motivation und Problemstellung

Schach wird seit Jahrhunderten gespielt. Seitdem sammeln, dokumentieren und kommentieren Spieler und Theoretiker die Partien. Daneben entwickelte sich ein Schachvokabular, das zahlreiche Bezeichnungen für jede Phase des Schachspiels enthält. Eröffnungen, Mittel- und Endspiele enthalten Stellungsmuster, die immer wieder auftauchen und zum Bestandteil der Schachtheorie gehören. Heutzutage spielen nach Schätzungen der Dachorganisation FIDE mehrere hundert Millionen Menschen Schach (vgl. [24]). Sie studieren die Theorie und analysieren die Partien, die statt auf Papier meistens mit Hilfe des PGN-Dateiformats in entsprechende Datenbanken gespeichert werden. Das Format erlaubt aber keine semantische Interpretation der Partien, wie z. B. "*Who lost against Magnus Carlsen in 2016?*". Nutzer, die nach Informationen suchen, müssen eine der vorhandenen Datenbanken mit Hilfe einiger Filter befragen und die Antwort eigenständig aus der Liste der Partien ermitteln. Im genannten Beispiel müsste der Nutzer zunächst im entsprechenden Filter "Magnus Carlsen" als Weiß- oder Schwarzspieler angeben, diesen als Gewinner der

Partie nennen und 2016 im Filter für Datumswerte hinterlegen. Im Anschluss erhält der Nutzer eine Liste von Partien, aus denen dann die Gegner abgelesen werden können.

Diese Informationssuche soll *Semantic Chess* erleichtern. Der Nutzer erhält nur eine Sucheingabe für eine natürlichsprachige Frage. Zusätzliche Angaben müssen nicht gemacht werden. Die Analyse der Anfrage übernimmt das System, ermittelt die Informationen aus den hinterlegten Partien und liefert passende Antworten zurück.

Damit zukünftige Systeme an diese Arbeit anknüpfen können, listet der für *Semantic Chess* entwickelte Benchmark 50 Schachfragen auf. Die Entwickler können ihre Ansätze mit diesen Fragen trainieren und anschließend die Performance, mit der von *Semantic Chess* verglichen. Dadurch soll es in Zukunft möglich sein, Partien, Informationen und Stellungsmuster mit natürlichsprachigen Anfragen zugänglich zu machen und bspw. Spieler beim Training oder der Analyse von Partien zu unterstützen.

## Kapitel 2

# Stand der Technik

Das folgende Kapitel beschreibt die theoretischen Grundlagen dieser Arbeit. Es definiert zunächst einige Begriffe, die im späteren Verlauf häufiger auftreten und erklärt die Funktionsweise der verschiedenen Ansätze im Question Answering anhand von ausgewählten Beispielen. Der Template-basierte Ansatz wird genauer beschrieben, da es die Grundlage des entwickelten Systems *Semantic Chess* ist. Die zweite Hälfte des Kapitels geht auf Herausforderungen bei der Entwicklung von Question-Answering-Systemen ein und wie sie mit Hilfe eines Benchmarks evaluiert werden können. Abschließend werden verwandte Arbeiten im Bereich des Question Answering für Schach und technische Ansätze zum Indexieren und Beschreiben von Schachpositionen vorgestellt.

### 2.1 Definition von Question Answering

*Question Answering* (QA) ist das Beantworten einer in natürlicher Sprache verfassten Frage durch ein Computerprogramm. Ein Question-Answering-System (QAS) versucht mit Hilfe von Informationen aus Text oder Daten, die bestmögliche Antwort zu finden (vgl. [39]).

Als Teilgebiet des *Information Retrieval* (IR) umfasst QA auch die maschinelle Verarbeitung der natürlichen Sprache (engl., *natural language processing*), sowie Disziplinen im Wissens- und Datenbankmanagement und der Kognitionswissenschaft, also u. a. der künstlichen Intelligenz und Linguistik (vgl. [38], S. 1).

Im Allgemeinen besitzt ein klassisches QAS drei Komponenten. Zunächst klassifiziert das System die Frage, um die zu erwartende Antwort einzugrenzen, sowie Entitäten und

Schlüsselwörter (engl., *keywords*) zu erkennen. Anschließend versucht das QAS im Prozess des *Information Retrievals* die Datenquellen mit der möglichen Antwort zu finden. Im letzten Schritt wird dann die Antwort extrahiert und diese dem Nutzer des QAS präsentiert (vgl. [38], S. 1).

Andere Autoren schlüsseln die Komponenten weiter auf. Mahender et al. beschreiben in ihrem Text Module zur Fragen- und Dokumentenverarbeitung, sowie zur Paragrafen- und Antwortextraktion (vgl. [49], S.1). Unger et al. beschreiben Module über Datenverarbeitung, Fragenanalyse, Datenabgleich, Anfragenkonstruktion, Scoring, Antwort-Retrieval, -bewertung und Präsentation (vgl. [17], S. 12-13).

Die Systeme sind von Mahender et al. in zwei Kategorien unterteilt. *Open-domain*-Ansätze versuchen alle Fragen zu beantworten, was allerdings in spezifischen Themengebieten zu Wissenslücken im System führen kann. Dagegen konzentriert sich das System bei *closed domains* auf ein Themengebiet mit bereits definierten Datenquellen und Vokabular (vgl. [49], S.1).

Ein Beispiel für *open domain* ist das in der Einleitung erwähnte System von IBM *Watson*. *Wolfram Alpha* ist ein System in der *closed domain*, das Fragen im mathematischen Bereich beantwortet, wie z. B. "*What is the height of the Eiffel tower?*". Ein weiteres Beispiel ist das QAS dieser Arbeit, das Antworten auf Fragen zum Thema Schach sucht.

Mit Bezugnahme auf das semantische Web kann zusätzlich noch das *Semantic Question Answering* (SQA) definiert werden. Der Unterschied zur oben aufgestellten Definition ist, dass ein SQA mit *Linked Data* arbeitet und im Gegensatz zu anderen Systemen, die z. B. ausschließlich Texte benutzen, auf eine RDF-Wissensdatenbank zurückgreift (vgl. [40], S.1).

## 2.2 Definition von Linked Data, RDF und SPARQL

Das semantische Web (engl., *semantic web*) bezieht sich auf ein Web aus Daten. Die dazugehörigen Werkzeuge, wie RDF und SPARQL können dieses Web abfragen, durchsuchen und Beziehungen unter den Daten verdeutlichen (vgl. [63]).

### 2.2.1 Linked Data

*Linked Data* bezeichnet die Sammlung von zusammenhängenden Daten im Web. Anwendungen können diese Daten abfragen und Verbindungen zwischen ihnen erkennen (vgl. [63]).

Tim Berners-Lee stellte 2006 folgenden Designprinzipien für *Linked Data* auf (vgl. [10]):

- Nutzung von URIs zur Bezeichnung von “Dingen”
- Nutzung von HTTP URIs, damit die Daten für jeden abrufbar sind
- Wenn die Daten abgerufen werden, sollen nützliche Informationen hinterlegt sein, die sich an Standards wie RDF und SPARQL halten
- Hinterlegen von weiteren URIs, damit weitere Informationen entdeckt werden

So soll ein Web aus Daten entstehen, das ein Standardformat besitzt, ständig aufrufbar, einfach zu verwalten und für Maschinen lesbar ist. Ein Beispiel für *Linked Data* ist DBpedia (vgl. [63]).

### 2.2.2 RDF

Das *Resource Description Framework* (RDF) ist ein Standard für den Datenaustausch im semantischen Web. Das Framework beschreibt Ressourcen (z. B. Dokumente, Menschen, Gegenstände, etc.) mit Hilfe von gerichteten Graphen, dient zur Zusammenführung von Daten, stellt maschinenlesbare Informationen zur Verfügung und erweitert die Verlinkungsstruktur durch das Nutzen von Tripeln. Tripel bestehen aus Subjekt, Prädikat und Objekt (vgl. [37]). Ein Beispiel ist

```
1 <Bob> <is a> <person> .
```

Listing 2.1: Tripel-Beispiel

Formal bedeutet das, dass ein RDF-Wissensgraph (engl., *knowledge graph*)  $K$  als Menge von Tripeln  $(s, p, o) \in (R \cup B) \times P \times (R \cup B \cup L)$  modelliert werden kann. Dabei ist  $R$  die Menge von RDF-Ressourcen,  $B$  die Menge der leeren Knoten (engl., *blank nodes*),  $P \subseteq R$  die Menge aller RDF-Prädikate (binäre Relationen, die zwischen Ressourcen und Literalen existieren) und  $L$  die Menge aller Literale (Datenwerte, wie z. B. eine Jahreszahl) (vgl. [51], S.2).

### 2.2.3 SPARQL

SPARQL (*SPARQL Protocol And RDF Query Language*) ist eine Anfragesprache für RDF. Durch Tripel, Konjunktionen, Disjunktionen und optionalen Mustern (engl., *pattern*) können Anfragen (engl., *queries*) formuliert werden. Zusätzlich bestehen weitere Möglichkeiten die Anfragen einzuschränken oder zu sortieren, u. a. durch eine Limitierung (engl., *limit*) oder durch eine Ordnung (engl., *order by*) (vgl. [64]). Zum Beispiel kann auf Grundlage der Daten in Listing 2.2 die SPARQL Query in Listing 2.3 formuliert werden.

```
1 <http://example.org/book/book1><http://purl.org/dc/elements/1.1/title>  
  "SPARQL Tutorial".
```

Listing 2.2: Beispiel eines Tripel-Datensatz (Quelle: [64], Abschnitt 2.1)

SPARQL Query:

```
1 @prefix ex:<http://example.org/book/book1>  
2 @prefix purl:<http://purl.org/dc/elements/1.1/title>  
3 SELECT ?title  
4 WHERE {  
5   ex:book1 purl:title ?title .  
6 }
```

Listing 2.3: Beispiel eines SPARQL Queries (Quelle: [64], Abschnitt 2.1)

Die Ergebnismenge umfasst das Literal *"SPARQL Tutorial"*.

## 2.3 Funktionsweise und Systeme im Question Answering

Im QA haben sich verschiedene Lösungsansätze entwickelt, um Fragen in natürlicher Sprache beantworten zu können. Wie in der Definition beschrieben, nutzen viele Systeme dafür ähnliche Komponenten (s. Abb. 2.1).

Durch die Datenvorverarbeitung (siehe *data preprocessing* in Abb. 2.1) soll die Laufzeit des Systems reduziert werden, durch bspw. einem Index für die Datenmenge (vgl. [17], S.12).

Nach dieser Vorbereitung ist der erste Schritt die in natürlicher Sprache formulierte Frage zu analysieren (*question analysis*). Mit Hilfe einer linguistischen Analyse sollen prägnante Teile der Frage entdeckt werden, wie z. B. Orte, Organisationen und Personennamen, oder um welchen Fragetypen es sich handelt. Hier kommen u. a. *part of speech*-Tagger

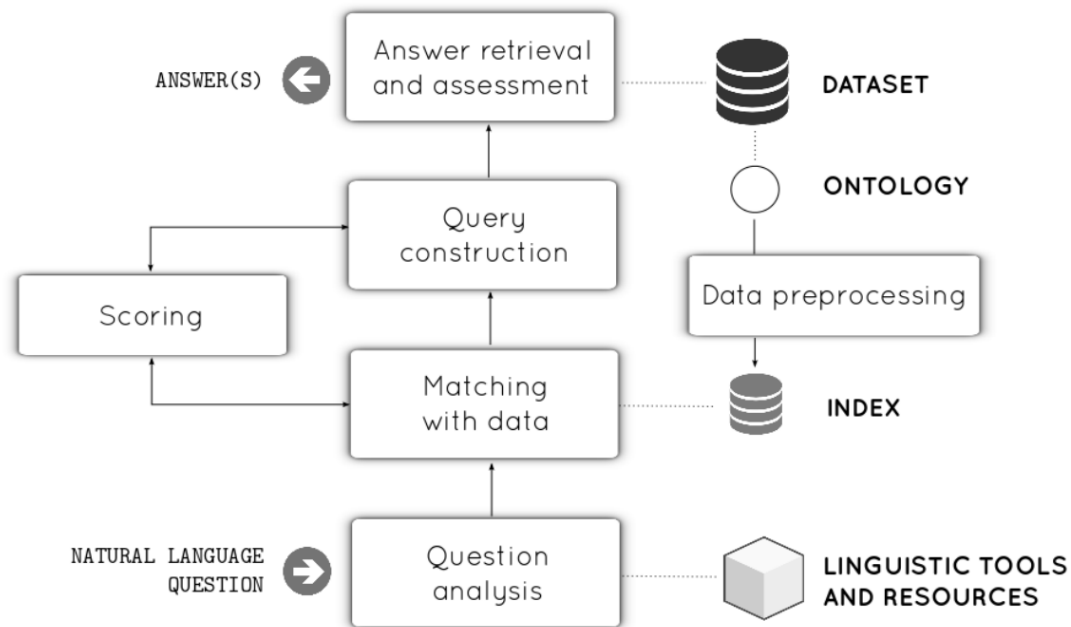


Abbildung 2.1: Komponenten eines QAS (Quelle: [17], S. 12)

(POS Tagger), wie der Stanford POS Tagger und Wörterbücher, wie *WordNet* zum Einsatz, das Gruppen von Synonymen enthält, sogenannten Synsets (vgl. [17], S.12-13).

Im nächsten Schritt muss das Vokabular der Frage mit dem der Datenbasis abgeglichen werden (*matching with data*). Dies geschieht z. B. durch eine Recherche der Begriffe. Danach wird die Anfrage formuliert (*query construction*) (vgl. [17], S.13).

Durch die letzten beiden Komponenten entstehen verschiedene Kandidaten, wie die Anfrage lauten kann. Ein *Scoring*-Modul soll diese gewichten und passende Anfragen höher listen. Dafür nutzt das *Scoring* Ähnlichkeitsmaße, sowie die Popularität der gefundenen Elemente und Zusammenhänge der verschiedenen Kandidaten mit der Datenbasis (vgl. [17], S.13).

In den letzten beiden Schritten erfolgt die Ausführung der Anfrage. Die gefundenen Antworten (*answer retrieval*) aus der Datenbank müssen anschließend bewertet (*assessment*) werden, u. a. ob sie dem Fragetypen entsprechen. Das System präsentiert danach dem Fragensteller das Resultat. Dabei gilt die Antwort so aufzubereiten, dass jeder Nutzer sie versteht, z. B. durch natürliche Sprache, Tabellen oder Graphen (vgl. [17], S.13).

### 2.3.1 Einteilung nach Unger et al. (2014)

Unger et al. teilen QAS, die *Linked Data* als Datenbasis besitzen, in sechs Gruppen ein.

#### Kontrollierte natürliche Sprache

Eine Herangehensweise basiert auf einer kontrollierten natürlichen Sprache. Das bedeutet, dass dem Nutzer zur Formulierung der Frage nur ein bestimmtes Vokabular zur Verfügung steht und dem System dadurch keine Verständnisprobleme entstehen (vgl. [17], S.14).

Diesen Ansatz wählt das QAS *Ginseng*. Sobald der Nutzer das Eingabefeld für die Frage nutzt, prognostiziert *Ginseng* die möglichen Vervollständigungen. Beispielsweise schlägt es für das "h" u. a. *hamilton* und *hammond* vor, nachdem der Nutzer "what is the h" eingetragen hat. Das System leitet den Nutzer so durch das Vokabular der genutzten Ontologie und sichert die Beantwortung der Frage ab (vgl. [45], S.1).

#### Formale Grammatik

Ein anderer Ansatz basiert auf formaler Grammatik. Das Ziel ist es eine semantische Repräsentation der Frage zu erstellen. Mit Hilfe der Grammatik können dann Fragen unterschiedlicher Komplexität beantwortet werden. Kann die Grammatik die Frage nicht zerlegen, erhält der Nutzer auch keine Antwort (vgl. [17], S.14). Ein Beispiel hierfür ist *ORAKEL*. *ORAKEL* akzeptiert als Eingabe Faktenfragen. Das System interpretiert die Frage und erzeugt für die Datenbasis eine Anfrage in logischer Form (Prädikatenlogik erste Stufe). Dazu wird zunächst eine Baumstruktur der Frage erstellt (s. Abb. 2.2), hier mit dem Beispiel "Which river passes through Berlin?".

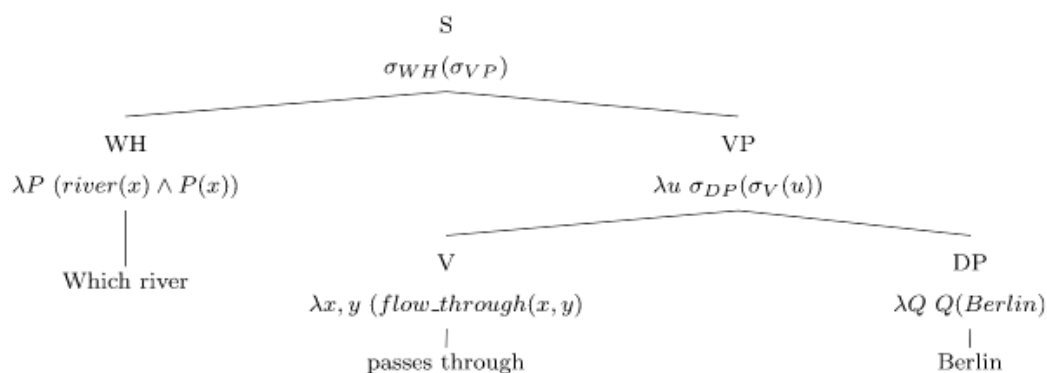


Abbildung 2.2: Syntaktische Analyse mit semantischen Repräsentationen für jede Phrase (Quelle: [19], S. 334)



In der Abbildung 2.2 wird die Frage (S) in eine Bestimmungsphrase (WH) und in eine Verbphrase (VP) aufgeteilt. Letzteres wird in ein Verb (V) und eine weitere Bestimmungsphrase (DP) zerlegt. Aus diesem lässt sich nach ein paar Zwischenschritten die Formel

$$?x \quad (river(x) \wedge flow\_through(x, Berlin)) \quad (2.1)$$

ablesen. Die Konvertierung dieser logischen Form, u. a. in SPARQL, übernimmt ein Prolog-Programm (vgl. [19], S. 326ff).

### **Zuordnung von sprachlichen Strukturen auf Ontologie-konforme semantische Strukturen**

Im nächsten beschriebenen Ansatz von Unger et al. sollen sprachliche Strukturen auf Ontologie-konforme semantische Strukturen abgebildet werden. Diese Systeme versuchen die Frage direkt in Tripel zu übersetzen. Mit Hilfe von Ähnlichkeitsmaßen wollen diese QAS eine Art bijektive Zuordnung zwischen Wörtern der Frage und den Elementen der Datenbasis ermitteln (vgl. [17], S.14).

*PowerAqua* verfolgt diesen Ansatz. Es kann verschiedene Datenbanken abfragen und besteht aus vier Komponenten. Die erste ist zuständig für die linguistische Analyse und übersetzt die Frage in Tripel.

*“Give me actors starring in movies directed by Clint Eastwood?” is transformed into the following set of QTs [Query Triple - Anm. d. Verf.]: QT<sub>1</sub>: <actors, starring, movie> & QT<sub>2</sub>: <actors/movies, directed, Clint Eastwood>” ([48], S. 6)*

Im Beispiel erkennt das Programm *actors* und *movies* als Subjekt, *starring* und *directed* als Prädikate, sowie *movie* und *Clint Eastwood* als Objekte. In QT<sub>2</sub> entsteht eine Mehrdeutigkeit, *actors* und *movies* passen potentiell an diese Position (vgl. [48], S.6).

Im nächsten Schritt sollen semantische Ressourcen in Ontologien recherchiert werden, in denen die Antwort enthalten sein kann. Das System ermittelt mögliche Kandidaten und überprüft, ob sie semantisch zur Frage passen. *PowerAqua* findet z. B. für die Phrase *“groups that play rock”* Ergebnisse in Ontologien, die Steine oder Musikgenres beschreiben. Nach der Analyse des Kontextes via *WordNet*, in dem *rock* steht, entscheidet sich *PowerAqua* für das Musikgenre als wahrscheinlichere Interpretation (vgl. [48], S6f).

Die nächste Komponente ordnet jetzt die Tripel aus der ersten Komponente den Ergebnissen aus der zweiten zu. Für QT<sub>1</sub>:<actors, starring, movies> findet *PowerAqua* <actor, starring, film> und <actor, starring, American\_movie> in DBpedia, sowie <Actor,

*participates, Movie*> und *<Actor, plays, Movie*> in einer Ontologie einer Filmdatenbank. Aus diesen kann die Datenabfrage formuliert werden (vgl. [48], S.7f).

Im letzten Schritt bildet das QAS aus den Antworten der verschiedenen Datenquellen ein zusammengefasstes Ergebnis. Dabei vermeidet das System redundante Daten durch eine Vereinigung der Antworten und ermittelt anschließend eine Schnittmenge. Im Beispiel mit Clint Eastwood listet *PowerAqua* 35 Schauspieler auf (vgl. [48], S.8).

## Template-basierte Ansätze

Beim Template-basierten Ansatz erzeugt das QAS im ersten Schritt ein Template. Im nächsten Schritt werden die Wörter der Frage den Platzhaltern in der Vorlage zugeordnet (vgl. [17], S.14f).

Dieser Ansatz bildet die Grundlage für das QAS *Semantic Chess* und wird detaillierter im Abschnitt 2.4 beschrieben.

## Suche in Graphen

Bei der Suche in Graphen ordnen QAS die Terme der Frage den Entitäten der Wissensbasis zu. Beginnend von diesen Elementen durchläuft das System den Graphen, um Verbindungen zu entdecken und daraus die Anfrage zu formulieren (vgl. [17], S.15).

Das QAS *Treo* verfolgt diesen Ansatz und ermittelt in fünf Phasen eine Antwort auf die vom Nutzer gestellte Frage. Im ersten Schritt versucht das System die Entitäten und das Schlüsselement (Pivot-Element) zu erkennen (vgl. [1], S.4).

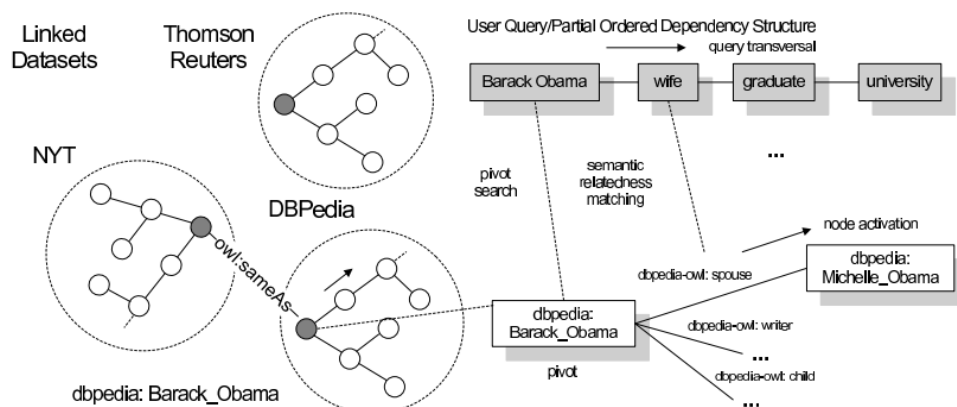


Abbildung 2.3: Arbeitsschritte des QAS Treo (Quelle: [1], S. 3)

In der Frage *“From which university did the wife of Barack Obama graduate?”* (s. Abb. 2.3) ermittelt *Treo* *“Barack Obama”* als Schlüsselwort. Nach diesem sucht *Treo* in DBpedia.

Das System ermittelt dann eine teilweise geordnete Struktur der Frage (PODS: *partial ordered dependency structure*), was etwa "Subjekt, Prädikat, Objekt" entspricht. Aus dem Beispiel wird dann: *Barack Obama* → *wife* → *graduate* → *university*. Für diese Bestandteile sucht *Treo* in DBpedia nach semantischen Beziehungen. Das Pivot-Element *Barack Obama* bildet den Ausgangspunkt. Ihm wurde *dbpedia:Barack\_Obama* zugewiesen. Von hier aus durchläuft *Treo* alle mit dem Pivot-Element verknüpften Eigenschaften und assoziierten Typen u. a. *dbpedia-owl:spouse*, *dbpedia-owl:writer* und *dbpedia-owl:child*. Durch eine Berechnung der semantischen Beziehungen ordnet das System *wife* dann *dbpedia-owl:spouse* zu, das auf den Knoten *dbpedia:Michelle\_Obama* verweist. Das Verfahren wird anschließend mit *graduate* und *university* wiederholt. Am Ende entsteht dadurch eine finale Zuordnung zu den Ressourcen aus DBpedia, aus der der Antwortgraph (vgl. Abb. 2.4) gebildet wird (vgl. [1], S.5ff).

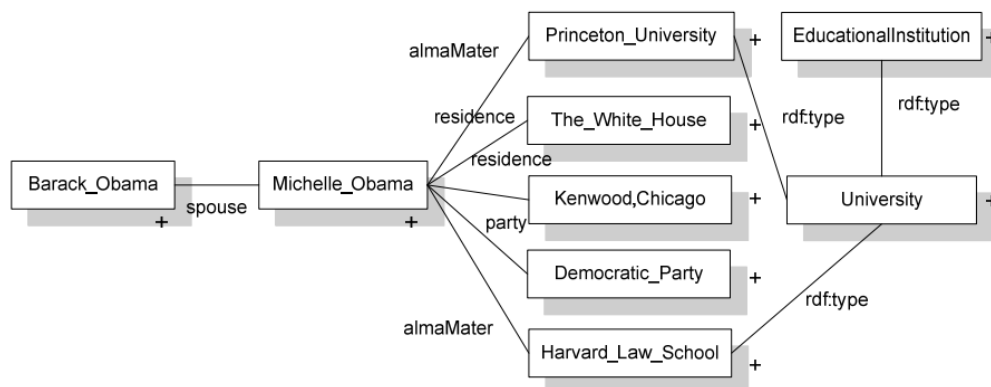


Abbildung 2.4: Antwortgraph von Treo (Quelle: [1], S. 3)

Die Antworten auf die Beispielfrage befinden sich in den Knoten *Princeton\_University* und *Harvard\_Law\_School*. *Treo* präsentiert die Ergebnisse dem Nutzer in natürlicher Sprache (vgl. [25]).

## Maschinelles Lernen (Machine Learning)

In den hier entwickelten Systemen werden Algorithmen verwendet, die z. B. das semantische Parsen durch die Bereitstellung von Wissensdatenbanken und Paaren aus Fragen und Antworten erlernen (vgl. [17], S.15).

Beispielsweise analysiert *DeepQA* von IBM Watson zunächst die Frage, zerlegt diese in ihre Bestandteile und interpretiert welche Rolle sie in der Frage spielen. Damit ermittelt *DeepQA*, welchen Fragetypen der Nutzer gestellt hat und welches Thema behandelt wird. *DeepQA* sammelt zu diesem Zeitpunkt eine Vielzahl von Interpretationen der Frage. Für

jede gesammelte Option durchsucht es Dokumente, in denen die Antwort stehen könnte und generiert daraus eine große Anzahl an Hypothesen. Falsche Hypothesen werden vom QAS aussortiert. Die übrigen werden mit einem *Scoring* Modell bewertet, das u. a. die Vertrauenswürdigkeit der Quelle berücksichtigt, sowie die Platzierung der möglichen Antwort im Text. Für das *Scoring* Modell benutzt es mehrere parallel ablaufende Algorithmen. Aus den Erfahrungen, die *DeepQA* beim Beantworten ähnlicher Fragen erworben hat, gewichtet es die Ergebnisse aus jedem einzelnen Algorithmus und kombiniert sie zu einem finalen Wert. Am Ende besitzt *DeepQA* ein *Scoring* für jede übrig gebliebene Antwort. Die höchst bewertete Antwort wird von *DeepQA* ausgegeben, wenn sie einen bestimmten Schwellenwert überschreitet (vgl. [29], S.67ff).

### 2.3.2 Einteilung nach Mahender et al. (2016)

Mahender et al. nennen allgemeiner drei verschiedene Ansätze, den linguistischen, den statistischen und den mustererkennenden Ansatz.

#### Linguistische Ansätze

Linguistische Systeme nutzen vor allem POS Tagger, Tokenizer und Parser. Die Systeme können dadurch die natürliche Sprache verstehen und die Anfragen für die Datenbanken dementsprechend formulieren. Mahender et al. listen in dieser Gruppe nur Systeme der *closed domains* auf, da sie nur auf eine strukturierte Wissensbasis zurückgreifen. Ein Beispiel ist *BASEBALL*, das Fragen zum Baseball-Sport beantworten kann (vgl. [49], S.35).

#### Statistische Ansätze

Statistische Ansätze basieren auf Techniken, wie dem Bayes-Klassifikator oder der Maximum-Entropie-Methode. Cai et al. haben in diesem Ansatz ein webbasiertes chinesisches QAS mit Antwortvalidierung entwickelt, das Ähnlichkeiten in Sätzen ermittelt. Diese Systeme sind v.a. bei *open domains* nützlich (vgl. [49], S.35f), da sie verschiedene Online-Quellen verwenden.

## Mustererkennende Ansätze

Den mustererkennenden Ansatz ordnen Mahender et al. wieder der Gruppe für (*closed domains*) zu und unterteilen sie in Oberflächenstruktur- (*surface pattern*) und Template-basierte Gruppen ein. Erstere lernt automatisch oder mit Hilfe von menschlichen Experten. Beispielsweise lassen Soubbtin et al. ihr QAS Satzmuster erlernen, die dann als Hinweise zur Beantwortung von Fragen dienen. Letztere lässt manuell oder automatisch Templates erstellen, denen dann eine passende Frage zugeordnet wird. Ein Beispiel stammt von Gunawerdena et al. Das System versucht die Sprache in einer SMS zu verstehen (vgl. [49], S.36).

Mahender et al. erwähnen zusätzlich die hybriden Systeme, die verschiedene Ansätze miteinander kombinieren. Als Beispiel führen sie IBM Watson aus dem DeepQA-Projekt auf, das u. a. *surface pattern* und Entropie verbindet (vgl. [49], S 36f).

## 2.4 Template-basiertes Question Answering

Viele Ansätze im QA versuchen Nutzerfragen direkt in Tripel zu übersetzen. Sie beruhen dabei auf Ähnlichkeitsmaßen und Suchheuristiken. Allerdings können die Tripel nicht alle semantischen Strukturen der Frage erfassen. An diesem Punkt setzt das Template-basierte QA an (vgl. [18], S.639). Dieser Ansatz erstellt auf Grundlage einer Analyse der Frage automatisch oder manuell ein Template. Im nächsten Schritt wird das Template instanziiert und die Platzhalter in der Vorlage ersetzt. Zum Beispiel wird für die Frage *“Who produced the most films?”* folgende SPARQL Template produziert:

```
1 SELECT ?x
2 WHERE {
3   ?x ?p ?y .
4   ?y rdf:type ?c .
5 }
6 ORDER BY DESC(COUNT(?y)) LIMIT 1 OFFSET
```

Listing 2.4: Template-basierte SPARQL Query (Quelle: [18], S.642)

Die Platzhalter sind in diesem Fall *c* für die URI einer Klasse, der *films* zugeordnet wird und *p* für das Schlüsselwort *produced* (vgl. [18], S.640). Der genaue Ablauf von der Frage über die instanziierte SPARQL Query zur Antwort an den Nutzer ist in Abbildung 2.5 dargestellt.

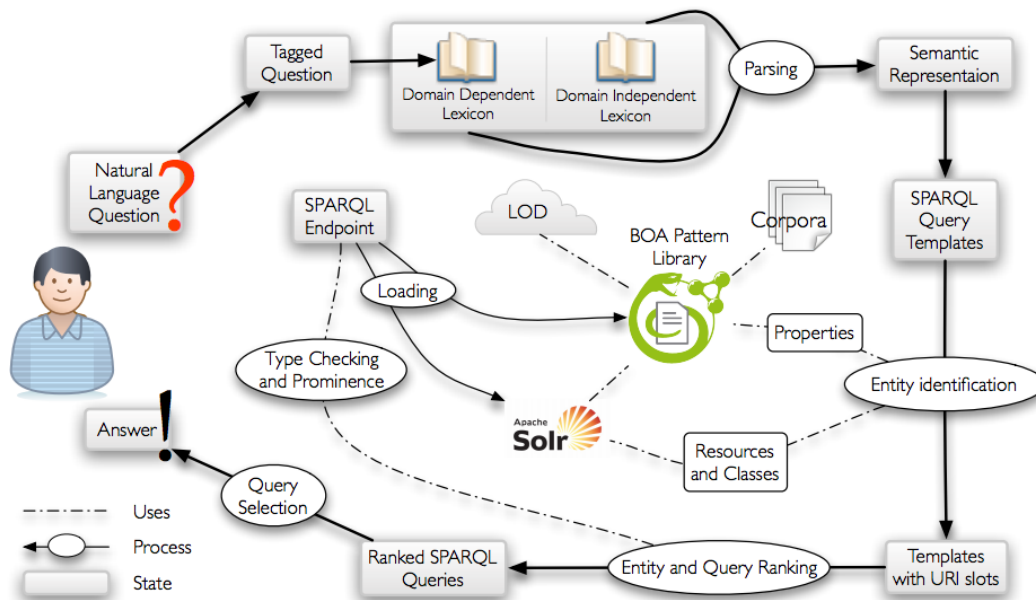


Abbildung 2.5: Überblick des Template-basierten SPARQL Query Generators (Quelle: [18], S. 641)

### Tagged Question

Der POS Tagger liefert Labels, um Wörter zu klassifizieren. Zum Beispiel erkennt der Tagger “rot” als Adjektiv und ordnet dem Wort das Label *JJ* zu. Diese Informationen werden genutzt, um syntaktische und semantische Zusammenhänge für die Templates aufzudecken (vgl. [18], S.641).

### Domain Dependent and Independent Lexicons

Der Prozess des Parsers beinhaltet domain-abhängige und -unabhängige Lexika. Im domain-unabhängigen Lexikon stehen Begriffe, die auf verschiedene Themengebiete passen, wie z.B. “oft”, “haben”, “sein”, etc. Sie werden manuell eingetragen. Im domain-abhängigen Lexikon stehen Wörter, die für ein Thema relevant sind. Unger et al. lassen sie in ihrem Ansatz nur bei Bedarf aufbauen. Im Vorfeld wird nicht bestimmt, welche URIs den Wörtern zugeordnet werden.

Der Stanford POS Tagger liefert die Informationen, um die Wörter zu gruppieren. Benannte Entitäten (z. B. “Wilhelm Steinitz”) deuten auf Ressourcen hin, für die dann das System einen Eintrag im Lexikon erstellt, mit syntaktischer und semantischer Repräsentation, sowie dazugehörigem Platzhalter. Für Substantive werden Einträge im Lexikon als Klasse und u.U. auch als Eigenschaft hinzugefügt. Verben treten im Lexikon hauptsächlich

als Eigenschaften auf (vgl. UNGER 2012, S.641).

Bezogen auf das Beispiel *“Who produced the most films”* bedeutet das (vgl. [18], S.642):

- Labels vom POS Tagger: *who/WP produced/VBD the/DT most/JJS films/NNS*;
- Im domain-unabhängigen Lexikon werden folgende Einträge gefunden: *who, the most, the, most*;
- Für *produced/VBD* und *films/NNS* werden Einträge im domain-abhängigen Lexikon erstellt

## Semantic Representation und SPARQL Query Template

Aus den Informationen der Lexika erstellt der Parser eine semantische Repräsentation (vgl. Abb. 2.6). Das Fragepronomen *who* wird durch die Variable *x* dargestellt. Das Substantiv *films* erhält die Repräsentation *films(y)* mit einem Slot *y*. *<THE MOST>* steht im domain-unabhängigen Lexikon und verweist auf *films*. Das Verb *produced* wird als Eigenschaft mit der Beziehung zwischen *x* und *y* dargestellt (vgl. [18], S.642).

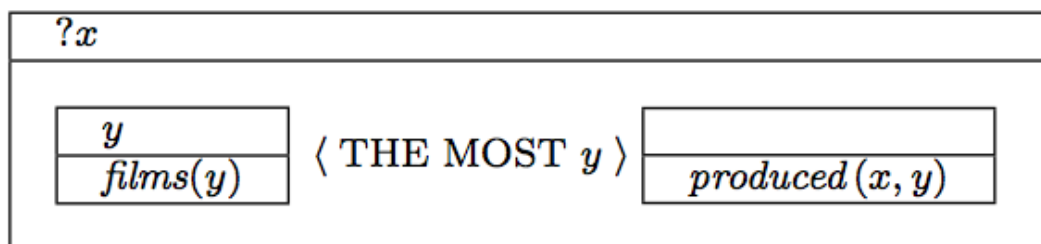


Abbildung 2.6: Semantische Repräsentation einer Frage (Quelle: [18], S. 642)

Im Anschluss entsteht daraus ein Template für ein SPARQL Query (s. Listing 2.4).

## Entity Identification

Unger et al. beschreiben Entitäten als Klasse, Eigenschaft oder Ressource, die mit Hilfe der Wörter aus der semantischen Repräsentation in einer Wissensbasis entdeckt werden. Zunächst beginnt das System jeweils alle Synonyme aus *WordNet* zu sammeln. Mit den Synsets versucht das System in der Wissensbasis entsprechende Labels zu entdecken. Die Zuordnung von Eigenschaft ist dabei schwieriger als die von Klassen und Ressourcen. Eigenschaften können auf unterschiedliche Weise formuliert werden (z. B. “X, der Autor von Y” und “Y ist das Buch von X”). Unger et al. nutzen dabei das Framework

*BOA Pattern Library*. BOA ermittelt dabei verschiedene Muster für eine Eigenschaft. Das System berechnet aus diesen Ergebnissen das passende Muster. Es entstehen mehrere SPARQL Queries mit den Entitäten, die für die Platzhalter geeignet sind (vgl. [18], S.642f).

### **Query Ranking und Auswahl**

Das System erstellt eine Liste mit gewichteten SPARQL Queries. Allerdings muss die Anfrage mit den höchsten Bewertungen nicht unbedingt ein Ergebnis zurückliefern. Unger et al. können aus Gründen der Performance nicht ausschließen, dass die erzeugten Tripel in dieser Query von Relevanz sind. Um den Nutzer trotzdem antworten zu können, werden die Anfragen der Liste (beginnend mit der am besten bewerteten Query) nach und nach ausgeführt. Die erste Anfrage, die kein leeres Resultat enthält, wird dem Nutzer präsentiert (vgl. [18], S.643f).

## **2.5 Herausforderungen im Semantic Question Answering**

Die vorgestellten Technologien in Abschnitt 2.2 verlangen von Nutzern ein Wissen über ihre Syntax und Semantik, sowie über ihr spezifisches Vokabular. Das (Semantic) Question Answering mit den verschiedenen Ansätzen aus Abschnitt 2.3 setzt an diesem Punkt an und übernimmt diese Aufgabe für einen Nutzer eines QAS (vgl. [40], S.1).

2016 haben Konrad Höffner et al. von der AKSW Group 62 Systeme im SQA untersucht und sieben Herausforderungen identifiziert, die bei der Entwicklung im Bereich QA auftreten. Ziel ist es einheitliche und strukturierte Lösungsansätze und Empfehlungen für zukünftige Entwicklungen zu erarbeiten (vgl. [40], S.1). Das Kapitel 3 überträgt diese Herausforderungen auf das Themengebiet Schach.

### **2.5.1 Lexikalische Lücke**

Die lexikalische Lücke im QA tritt auf, wenn in der Wissensdatenbank ein anderes Vokabular vorhanden ist als in der gestellten Frage. Die natürliche Sprache besitzt die Eigenschaft Sachverhalte auf unterschiedliche Weise auszudrücken. Eine Wissensdatenbank enthält meistens nicht alle unterschiedlichen Synonyme für einen Begriff. Wenn diese Herausforderung in einem zu entwickelnden System beachtet wird, erhöht dies den Anteil der



Fragen, die das QAS beantworten kann. Stammformreduktion, Ähnlichkeitsmaße, *Automatic Query Expansions* oder *Pattern Libraries* sind Techniken, die die lexikalische Lücke abschwächen (vgl. [40], S. 7).

In der Linguistik beschreibt die lexikalische Lücke das Fehlen eines Begriffs, z. B. um auszudrücken, dass eine Person "nicht durstig" ist (vgl. [40], S. 7).

### 2.5.2 Ambiguität

Ambiguität beschreibt die Mehrdeutigkeit von Wörtern. Höffner et al. verweisen auf die verschiedenen Arten der Ambiguität, die entweder syntaktisch ("*flying planes*") oder semantisch ("*bank*") sein können:

- Homonyme: ein Begriff, der unterschiedliche Konzepte besitzt
- Polyseme: ein Begriff, der unterschiedliche, aber verwandte Konzepte anspricht
- Metonyme: ein Begriff, der nicht seiner ursprünglichen Bedeutung entspricht
- Hypernyme: beschreibt einen Oberbegriff

Ein QAS setzt zur automatischen Disambiguierung korpus- oder ressourcenbasierte Methoden ein. Ersteres basiert auf Wahrscheinlichkeiten und untersucht mit Hilfe von Kookkurrenzen, Synonymen, Hyponymen (Unterbegriffe), POS Tags und Syntaxbäumen, welche Bedeutung ein Wort hat. Letzteres untersucht die RDF-Ressourcen und baut mit Hilfe ihrer Eigenschaften *Scoring* Modelle auf. Zu den verwendeten Techniken zählen das *Hidden Markov Model*, *Markov Logic Networks* oder der Ansatz vom System *DEANNA* (vgl. [40], S.10), die Disambiguierung als *Integer Linear Programming* Problem anzusehen.

Um Mehrdeutigkeiten manuell zu lösen, setzen einige Systeme den Nutzer ein. Die Systeme stellen dann u. a. ein kontrolliertes Vokabular zur Verfügung oder zerlegen komplexe in einfache Anfragen, um sie anschließend mittels *Crowdsourcing* zu klären (vgl. [40], S. 9-11).

### 2.5.3 Mehrsprachigkeit

Das Web ist mehrsprachig. Nicht nur liegen Datenmengen in unterschiedlichen Sprachen vor, auch die Nutzer eines QAS verwenden verschiedene Muttersprachen. QAS, die mehr-

sprachig entwickelt werden, sind flexibler. Die Systeme verfolgen z. B. den Ansatz multilinguale Wissensdatenbanken einzubinden oder Verbindungen zwischen den Sprachversionen von Wikipedia herzustellen (vgl. [40], S. 11).

#### **2.5.4 Komplexe Anfragen**

Komplexe Anfragen enthalten Kombinationen und Verknüpfungen von Fakten. Um diese Herausforderung zu lösen, gibt es verschiedene Ansätze. IBM Watson ermittelt den Fokus der Frage, um damit den Fragentyp zu erkennen und die Zahl der möglichen Antworten einzugrenzen. YAGO-QA erlaubt verschachtelte Anfragen, wenn die Unteranfragen beantwortet sind (vgl. [40], S.11-12).

#### **2.5.5 Verteiltes Wissen**

Es gibt verschiedene Wissensdatenbanken mit unterschiedlichen Inhalten. Um eine Frage beantworten zu können, müssen gelegentlich Datenbanken verknüpft werden. QAS durchsuchen dafür mittels der gefundenen Entitäten in der Frage verschiedene Wissensdatenbanken und verbinden die Ergebnisse mit Hilfe von Ähnlichkeitsmaßen. Der Ansatz von ALOQUS zerlegt die Anfrage und lässt die Teile in verschiedenen Zielsystemen untersuchen (vgl. [40], S.12).

#### **2.5.6 Prozedurale, zeitliche und räumliche Fragen**

Es gibt verschiedene Fragetypen. QAS beantworten Faktenfragen, Ja-oder-Nein-Fragen oder Listenfragen, da diese einfach in SPARQL umgewandelt werden können.

Verfahrensfragen, z. B. "warum" oder "wie" sind im Bereich der SQA zur Zeit nicht zu lösen. Ein Forschungsansatz stammt von KOMODO. Das System gibt auf solche Fragen keine Antwort, sondern bietet dem Nutzer eine Anleitung zur Beantwortung der Frage (vgl. [40], S.12).

Im Bereich der zeitlichen Fragen gibt es verschiedene Lösungsansätze, die z.B zeitliche Relationen analysieren ("vorher" oder "danach"). Räumliche Fragen lassen sich durch den Einsatz von Geokoordinaten (Längen- und Breitengrade) in RDF-Darstellung beantworten (vgl. [40], S.12-13).

### 2.5.7 Templates

Der Einsatz von Templates (Vorlagen für SPARQL Queries) erfolgt bei komplexen Fragen. Bei der Erstellung der Templates verfolgen die QAS zwei unterschiedliche Ansätze. Ein Ansatz baut die Templates anhand der syntaktischen Struktur der Frage auf. Hierfür werden z. B. semantische Labels den Phrasen der Frage zugeordnet und anschließend durch ein Perzeptron (ein neuronales Netz) gelöst (vgl. [40], S.13). Ein weiterer Ansatz teilt die Fragen einem manuell oder automatisch erstellten Template zu. Diese Methode ist Grundlage dieser Arbeit. Der Ablauf ist im Abschnitt 4.1 beschrieben.

Den Einfluss der Herausforderungen auf das System *Semantic Chess* fasst das Kapitel 5.3 und die dazugehörige Tabelle 5.3 zusammen.

## 2.6 Benchmark

Um die Ergebnisse der unterschiedlichen Ansätze miteinander vergleichen zu können, werden Benchmarks benutzt. Hauptziele der daraus entstehenden Referenzwerte sind die Sichtung der Stärken und Schwächen der einzelnen Systeme. Zukünftige QAS sollen aus diesen Erfahrungen lernen (vgl. [50], S.1).

Ein solches Benchmark hat z. B. die *Question Answering over Linked Data (QALD) challenge* seit sechs Jahren vorangetrieben. Die siebte Ausgabe von QALD stellte den teilnehmenden Systemen Arbeitsaufgaben aus vier Bereichen: mehrsprachiges QA mit DBpedia, hybrides QA, *large-scale* QA mit RDF und QA mit Wikidata. In jedem Bereich gab es zwischen 100 und 215 Fragen als Trainingsdaten und 50 Testfragen. Die Ausnahme bildet das *large-scale* QA mit 2 Mio. Testfragen (vgl. [50], S.4ff).

Um die Antworten der Systeme einschätzen zu können, werden für die Fragen im Vorfeld die bestmöglichen Antworten ermittelt (*gold standard*). Mit dieser Hilfe kann QALD die Genauigkeit und Trefferquote der Ergebnisse berechnen. Dafür dienen die Mikro- und Makro-Werte der Maße *Recall*, *Precision* und der aus diesen beiden zu berechnende *F-Measure* (vgl. [50], S.7).

## Recall

Recall beschreibt das Verhältnis von relevanten Antworten des QAS zum *gold standard*. Es beschreibt die Trefferquote. Für eine Frage  $q$  gilt dann (vgl. [50], S.7):

$$\text{recall}(q) = \frac{\text{Anzahl der korrekten Antworten des Systems für } q}{\text{Anzahl der } \textit{gold standard} \text{ Antworten für } q} \quad (2.2)$$

## Precision

Precision ist das Verhältnis aller relevanten Antworten des QAS zur Summe aller relevanten und irrelevanten Antworten des Systems für eine Frage  $q$ . Das Maß berechnet die Genauigkeit des Resultats (vgl. [50], S.7):

$$\text{precision}(q) = \frac{\text{Anzahl der korrekten Antworten des Systems für } q}{\text{Anzahl aller Antworten des System für } q} \quad (2.3)$$

## F-Measure

Das F-Measure berechnet aus Precision und Recall den gewichteten harmonischen Mittelwert:

$$f\text{-measure}(q) = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (2.4)$$

## Mikro- und Makrowerte

Für die Makrowerte werden zunächst für jede Frage Precision und Recall ermittelt. Von diesen Einzelergebnissen wird dann der Durchschnitt berechnet. Im Gegensatz dazu summieren die Mikrowerte erst alle Antworten jeder Frage zusammen und berechnen im Anschluss Precision und Recall (vgl. [50], S.7).

Diese Werte können sich zum Teil stark unterscheiden. Der Grund hierfür ist, dass Makrowerte jeder Frage das gleiche Gewicht zuordnen und somit unterschiedlich große Antwortmengen der Fragen nicht berücksichtigen. Dagegen betrachten die Mikrowerte den Anteil jeder Frage, den sie für den finalen Wert leisten. Fragen mit einer großen Antwortmenge fließen somit mehr in das Endergebnis ein und zeigen, wie effektiv ein System mit diesen umgeht (vgl. [2]).

## Weitere Benchmarks

Es gibt noch weitere Organisationen, die Benchmarks entwickeln, allerdings nicht in dem Umfang wie QALD im Bereich Linked Data, *large-scale* und hybride Daten (vgl. [50], S. 2).

Unter anderem erstellt BioASQ ein Benchmark im biomedizinischen Bereich, das QAS herausfordert Fragen mit Hilfe von Texten und strukturierten Daten zu beantworten (vgl. [11]).

Ein weiteres Beispiel stammt von der Stanford Universität. Mit SQuAD haben die Forscher eine Datenmenge mit ca. 100.000 Antwort-Frage-Paaren aus etwa 500 Wikipedia-Dokumenten aufgebaut (vgl. [55]).

## 2.7 Verwandte Arbeiten und Ansätze im Bereich Schach

Schach beschäftigt Entwickler vor allem in der Programmierung von Engines. Schon 1996 gelang es dem Computer *Deep Blue* den damals amtierenden Weltmeister Garri Kasparow in einer Partie zu schlagen (vgl. [32]). Die heutigen Computer sind für die meisten menschlichen Gegner nicht mehr zu besiegen. *Houdini*, *Stockfish*, *Komodo*, *Shredder* u. v. m. treten in eigenen Wettkämpfen gegeneinander an, um unter sich die beste Engine zu ermitteln (vgl. [52]).

Daneben gibt es zahlreiche Datenbanken, die Schachpartien speichern. Bekannte sind u. a. `chess-db.com` und `chessbase.com`, die jeweils bis zu mehreren Millionen Partien verwalten (vgl. [15] und [58]).

### 2.7.1 Schach-Question-Answering

Question Answering im Bereich Schach ist noch relativ unerforscht. Die Recherche für diese Arbeit konnte zwei Richtungen identifizieren. Die erste konzentriert sich auf die Metadaten der Partien (Spieler, Ort, Datum, Event) und die zweite auf Positionen.

#### QAS für Metadaten

Die Grundlage dieser Arbeit bilden zwei Praktika von der Universität Leipzig aus den Jahren 2013 und 2015.

Im Jahr 2013 entwickelte eine Gruppe von Studenten eine auf RDF basierende Schachplattform. Das System kann das Schachdateiformat PGN (*portable game notation*) in RDF übersetzen und bietet Nutzern eine Oberfläche zur Abfrage der gespeicherten Daten. Es existieren Eingabefelder zu den Metadaten der Schachpartien und zu den Daten der Schachspieler, wie z. B. für den Geburtsort oder der Elo (Zahl zur Bewertung der Stärke von Schachspielern) (vgl. [3], S.5f).

Auf diesem Programm basierend entwickelte eine weitere Studentengruppe 2015 ein QAS (vgl. Anhang Abb. A.1, S. 95). Über eine Web-Oberfläche kann der Nutzer seine Frage formulieren (*ServerProcess*). Das System analysiert diese mit Hilfe der Klasse *Question-Tagger* und erstellt automatisch mehrere Templates im *QuestionParser*. Der *QuestionParser* greift auf ein Lexikon mit ausgewählten Wörtern zurück. Mit deren Hilfe stellt die *TemplateFactory* mehrere Templates zur Verfügung. Die Vorlagen werden anschließend gewichtet und die Platzhalter ersetzt. Daraus entsteht ein SPARQL Query für die Abfrage der eingesetzten Datenbank Virtuoso. Als Ergebnis entsteht ein JSON-Objekt, das für den Nutzer als Tabelle angezeigt wird (vgl. [4], S.5f). Aufgrund fehlender Ressourcen kann diese Arbeit nicht mit *Semantic Chess* verglichen werden.

Da diese Arbeiten die Grundlage für das hier entwickelte System *Semantic Chess* sind, konzentriert es sich vor allem auf die Metadaten und -informationen, die aus den PGN-Dateien gewonnen werden können.

## **QAS für Schachpositionen**

Ein weiterer Ansatz versucht Fragen zu Schachpositionen zu beantworten, wie z. B. "Welche Figur steht auf Feld d6?", "Ist diese Stellung Patt?" oder "Kann Weiß rochieren?". Cirik et al. entwickelten ein System, das auf maschinellem Lernen beruht und auf 15 verschiedene Fragetypen antwortet. Dazu zählen u. a. Positionen, Anzahl und Existenz von Figuren, sowie Fragen nach legalen Zügen, dem Materialverhältnis und Rochaderechten. Die Antworten beschränken sich dementsprechend auf Ja, Nein, Anzahl und Namen von Figuren (vgl. [41], S.1ff).

Zu jedem Typ wurden 1000 Fragen und Antworten erstellt, aus denen das System lernen konnte. Als Ergebnis kann ein Nutzer Fragen zu einer von ihm angegebenen Position stellen, die das QAS mit einer Wahrscheinlichkeit zwischen ca. 10% ("*what is the material count of x*") und 100% ("*is x-y a legal move*") beantwortet (vgl. [42], S.7ff).

*Semantic Chess* kann einige Fragen zu Schachpositionen beantworten. Dazu nutzt es die

in der Datenbank für jede Position hinterlegten FEN-Zeichenketten (vgl. Kap. 2.7.4). Auf eine Frage wie *“Show me games with rook against queen.”* testet das System jede FEN mit einem regulären Ausdruck und überprüft damit die Partien nach Stellungen, in denen ein Turm gegen eine Dame spielt.

## 2.7.2 Schach-Ontologien

Die Studentengruppe aus dem Praktikum von 2013 hat für ihr Programm eine eigene Ontologie entwickelt (vgl. Abb. 2.7). Auf diese greift *Semantic Chess* zurück.

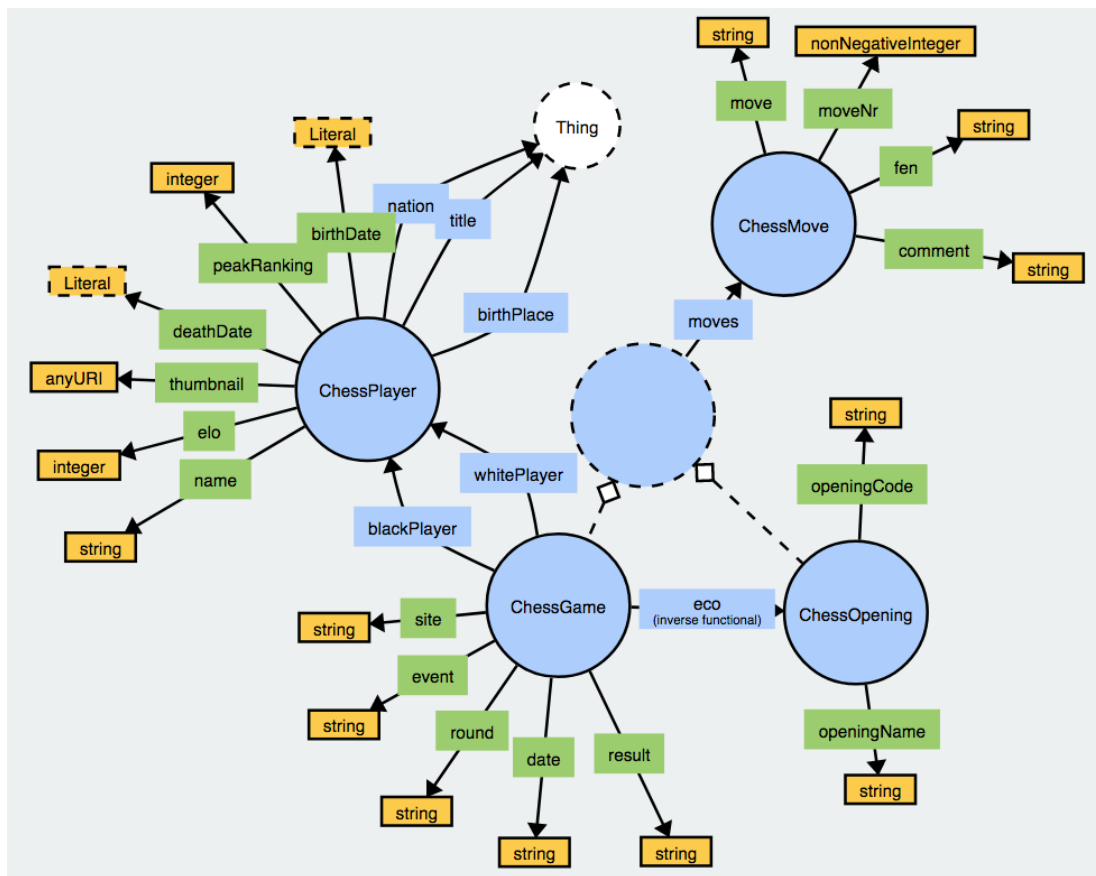


Abbildung 2.7: Visualisierung der Ontologie aus [3] unter Verwendung von <http://visualdataweb.de/webvowl>

Es entstanden vier Klassen: Schachpartie, -zug, -spieler und -eröffnung. Die Klasse der Schachpartien erhielt die Metadaten einer PGN-Datei (Runde, Event, Resultat, Standort, Datum) und die Schachzüge die Datentypen Zug, -nummer, Kommentar und FEN (Kurznotation für Schachpositionen). Auf die Klasse der Schachspieler wird mit den Eigenschaften Schwarz- und Weißspieler verwiesen. Die vierte Klasse ist für die Schacheröffnungen zuständig und deren Eigenschaften.

Eine weitere Ontologie stammt von Krisnadhi et al. aus dem Jahr 2015 (vgl. Abb. 2.8).

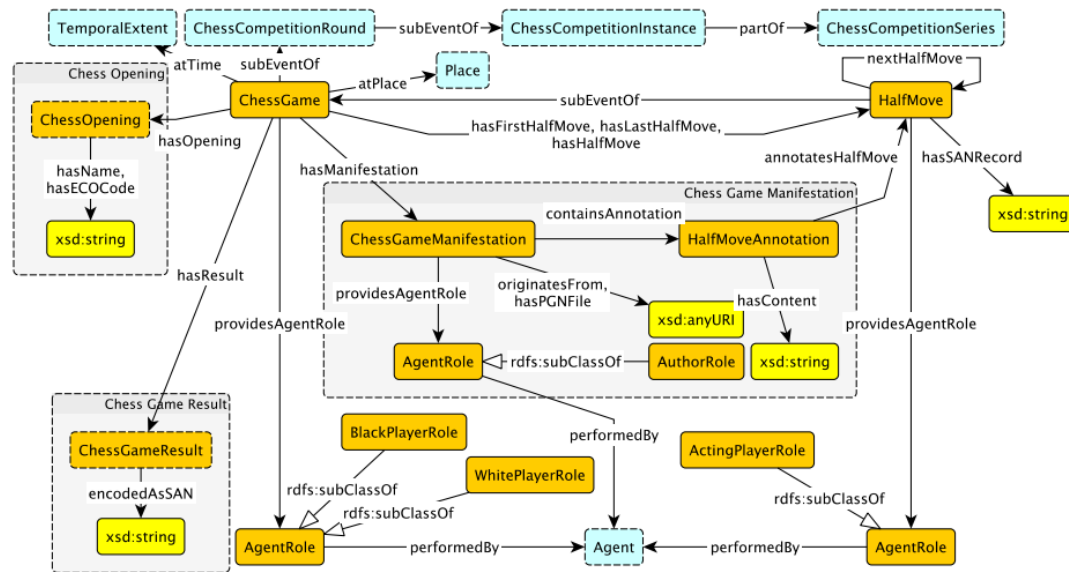


Abbildung 2.8: Ontologie von Krisnadhi et.al: “The core of Chess pattern modeling chess games. Though Event is omitted, half moves, chess games, and chess competition rounds/instances/series are events. Orange boxes are atomic classes. Blue boxes are classes with details hidden in a separate pattern/ontology. Grouping boxes are sub-patterns that can be modeled separately.” (Quelle: [46], S. 2)

Die Klasse *ChessGame* ist als Event angelegt, da sie zu einem bestimmten Zeitpunkt an einem Ort von Akteuren gespielt wird. Im Gegensatz zu der Studentengruppe der Universität Leipzig haben Krisnadhi et al. bewusst auf die Eigenschaften *whitePlayer* und *blackPlayer* verzichtet. Dafür führen sie die Klasse *AgentRole* ein. Sie wollen damit eine größere Flexibilität erreichen und Informationen erst einfügen, wenn sie benötigt werden. Die Züge einer Schachpartie sind mit der Klasse *HalfMove* dargestellt. Die Halbzüge bilden eine endliche Sequenz und sind so definiert, dass eine Schachpartie nicht von der Startposition aus beginnen muss. Damit erlauben Krisnadhi et al. auch die Beschreibung von Schachproblemen (vgl. [46], S. 3).

Die Klasse *ChessGameManifestation* entsteht aus einer URI zu einer PGN-Datei und enthält u. a. Kommentare zur Partie und den Zügen, die von einem weiteren Agenten (*AgentRole*) verfasst sind. Sie lässt zudem Platz für weitere Inhalte, die in Zukunft in eine PGN-Datei hinzugefügt werden (vgl. [46], S.3f).

Die Wissenschaftler entwickelten zudem eine Webseite, auf der der Nutzer PGN zu RDF konvertieren und nach Partien im RDF-Format suchen kann. Um die Datenbank zu füllen, verwenden sie einen Webcrawler, der PGN-Dateien sammelt. Das System wandelt



danach PGN zu RDF um, vervollständigt die Standorte mit einer Abfrage an die API von `geonames.org` und verlinkt die Schachspieler mit den entsprechenden Einträgen in DBpedia. Die Schacheröffnungen werden ebenfalls mit ihren DBpedia-Ressourcen und zusätzlich hierarchisch untereinander in Verbindung gesetzt (vgl. [56]).

### 2.7.3 Schachpositionen als Problem des Information Retrievals

Question Answering im Schach beschäftigt sich zur Zeit hauptsächlich mit den Metadaten einer Partie oder wie bei Cirik et al. mit einer spezifischen Position. Ein zukünftiges Anliegen eines QAS kann auch das Beantworten von Fragen sein, die sich auf bestimmte Stellungsmuster beziehen, wie z. B. *“Show me games with the Epaulette Mate.”* oder *“Are there similar positions like in the game XY on move 35?”*.

Eine Grundlage zum Lösen dieser Fragen bietet die Arbeit von Ganguly et al. Ziel ist es ähnliche, aber nicht stellungsgleiche Positionen zu entdecken. Für jede Stellung entwerfen sie eine Textdarstellung, in die Position, Erreichbarkeit (berechnet mit einer Gewichtungsfunktion) und Verbindungen der Figuren einfließen. Die Verbindungen untereinander sind nochmals in angegriffene, verteidigte und “Röntgen”-angegriffene (engl., *x-ray attack*) Felder unterteilt. Ein Beispiel für die angegriffenen Felder sieht wie folgt aus: *“b>Nc3 r>Nc3 p>Ph5 R>pd6 ...”*. In dieser Darstellung greift u. a. ein schwarzer Läufer (b) den weißen Springer (N) auf c3 an und ein schwarzer Bauer (p) den weißen Bauern (P) auf h5 (vgl. [36], S.4ff).

Mit einem auf dieser textuellen Darstellung beruhenden Algorithmus konnten sie ähnliche Stellungen zu gegebenen Positionen ermitteln. Die Ergebnisse wurden im Anschluss von Menschen bewertet, die ein gewisses Grundverständnis von Schach besitzen (vgl. [36], S.7f).

### 2.7.4 Weitere Erwähnungen

In der Vergangenheit sind verschiedene Methoden entstanden, die zum Ziel haben, Schachpositionen zu beschreiben und zu indexieren, sowie Schachpartien besser durchsuchbar zu machen. Sie werden in diesem Abschnitt kurz beschrieben.

## **Chess Query Language (CQL)**

Die CQL ist von Gady Costeff und Lewis Stiller entwickelt und hat zum Ziel nutzerdefinierte Muster und Themen in PGN-Dateien zu finden. Die Funktionalitäten reichen von einfachen Anfragen, wie "Figuren XY auf der h-Linie" bis hin zu Schachmotiven, wie dem Zugzwang (vgl. [23]).

## **Chess Game Markup Language (ChessGML)**

ChessGML basiert auf XML und versucht ein standardisiertes Schachdateiformat zu entwickeln, um damit jede Form von Schachdaten leicht veröffentlichen und austauschen zu können. Es wird seit 2000 von Andreas Saremba entwickelt (vgl. [57]).

## **Forsyth-Edwards-Notation (FEN)**

FEN ist eine bekannte Kurznotation für Schachpositionen. In einer Textzeile beschreibt sie die Stellung der Figuren, welche Farbe am Zug ist, welche Rochaderechte existieren, ob En Passant auf einem Feld möglich ist, wann die 50-Züge-Regel erreicht wird und die aktuelle Zugnummer (vgl. [28], Abschnitt 16).

## **Extended Position Description (EPD)**

Das EPD-Format ähnelt dem FEN, wird aber hauptsächlich von Computerprogrammen genutzt. Es fügt zusätzliche Informationen in die Textzeile ein, wie z. B. die Zahl für Zugwiederholungen oder Remis-Angebote (vgl. [43]).

## **Guy / Blandford / Roycroft Code (GBR Code)**

Der GBR Code besteht im Kern aus vier durch einen Punkt getrennte Komponenten. Die erste zeigt die Anzahl der Figuren und die zweite die Anzahl der Bauern. Damit wird das Materialverhältnis beschrieben. Die letzten beiden Komponenten beschreiben die Position der Figuren und Bauern. Zusätzlich können noch Kontrollzahlen und das Ziel der Position (z. B. "Weiß gewinnt") angegeben werden (vgl. [8]).

## **Numeric Annotation Glyphs (NAG)**

NAGs sind Sequenzen mit einem Dollar-Zeichen (\$) gefolgt von einer Zahl (0 bis 255). Sie bewerten Züge ("fragwürdiger Zug"), Stellungen nach Zügen ("Weiß steht besser") und verdeutlichen in der PGN, ob ein Spieler Zeitdruck hatte. Schach-Engines verwenden NAGs, Schachspieler eher eine Sammlung von Zeichen, wie "!!" ("starker Zug") oder "?!" ("zweifelhafter Zug") (vgl. [28], Abschnitt 10).

## Kapitel 3

# Herausforderungen im Schach-Question-Answering

Das Question-Answering-System *Semantic Chess* soll Fragen aus dem Themengebiet Schach beantworten (vgl. Kapitel 2.7). Über eine Weboberfläche kann ein Nutzer seine Fragen stellen. *Semantic Chess* analysiert die Frage und präsentiert dem Nutzer die gefundenen Ergebnisse. Um das System robust zu gestalten, geht dieses Kapitel zunächst auf die Herausforderungen ein, die bei der Erstellung des QAS auftreten. Danach erklärt das folgende Kapitel den logischen Aufbau und die Implementierung des Programms. Eine Beispielfrage verdeutlicht dann den Ablauf.

Die folgenden Abschnitte sollen die Herausforderungen aus Kapitel 2.5 auf das Themengebiet Schach übertragen und Lösungsansätze bieten. Diese Ansätze werden zum Teil im entwickelten Programm wiederverwendet, sind aber auch für weitere Arbeiten gedacht. Das Kapitel 5, zur Evaluation des QAS, greift auf die Herausforderungen zurück und analysiert die Stärken und Schwächen des finalen Systems.

### 3.1 Lexikalische Lücke

Die natürliche Sprache kann einen Sachverhalt unterschiedlich ausdrücken. Die Wissensbasis dagegen enthält nicht alle Bedeutungen. Daher tritt auch im Schach die lexikalische Lücke auf. Es gibt zwar ein abgegrenztes und speziell auf Schach ausgelegtes Fachvokabular, allerdings ist dies umfangreich und verändert sich gelegentlich.

Ein großer Teil des Vokabulars bezieht sich auf die Eröffnungen. In der aktuellen Versi-

on nutzt das System ca. 3400 verschiedene Bezeichnungen, die unter <https://github.com/hayatbiraalem/eco.json> gesammelt wurden. Die Struktur der Eröffnungen unterteilt sich in offene, halboffene und geschlossene Systeme, sowie Flankensysteme und indische Verteidigungen, die jeweils verschiedene Haupt- und mehrere Nebenvarianten besitzen. Einige Eröffnungen tragen den Namen von Schachspielern (z. B. "Réti-Eröffnung"), Städten (z. B. "Wiener Partie"), Ländern (z. B. "Spanische Eröffnung"), Orten (z. B. "Sizilianischen Verteidigung"), Tieren (z. B. "Orang-Utan-Eröffnung"), Ereignissen (z. B. "Halloween Gambit"), eine Kombination (z. B. "Nimzowitsch-Indisch") oder ungewöhnliche Bezeichnungen (z. B. "Frankenstein-Drakula-Variante"). Es gibt Eröffnungen mit Zusatzbezeichnungen, wie "Gambit", "Anti", "Gegen" oder "Halb" und es existieren offizielle und inoffizielle Bezeichnungen, die nur von einer Gruppe von Schachspielern verwendet werden, sowie Varianten, die noch keinen Namen tragen.

Der ECO-Code (*"Encyclopedia of Chess Openings"*) hat sich zur Aufgabe genommen, die Eröffnungen zu katalogisieren (s. Tab. 3.1). Das System bildet Sequenzen aus einem Anfangsbuchstaben (A bis E), gefolgt von einer Zahl (00 - 99). Diese Sequenz verweist dann auf den Namen der Eröffnung, sowie deren Alternativen und zu den Zügen. Zum Beispiel trägt die Eröffnung "Sokolski-Eröffnung" auch die Bezeichnungen "Orang-Utan-Eröffnung" und "Polnische Eröffnung". Der ECO-Code kann als Index verwendet werden, um verschiedene Eröffnungsamen zu entdecken oder als invertierter Index, um für einen Namen über den Code die dazugehörigen Züge zu ermitteln.

Code	Name	Züge
B02	Alekhine Defense	1.e4 Nf6
...	...	...
D47	Semi-Slav Defense: Meran Variation	1. d4 d5 2. c4 c6 3. Nc3 Nf6 4. ...

Tabelle 3.1: Beispiele für den ECO Code

Die Struktur, die der ECO-Code vorgibt, kann als RDF konzipiert werden. Die Schwierigkeit dabei ist, das Framework aktuell zu halten und Neuerungen in Varianten zu aktualisieren. Eine weitere Herausforderung sind z. B. Personen- und Städtenamen im Eröffnungsamen. Das Programm zur Erkennung von Entitäten muss hier auf Schlüsselwörter, wie "Eröffnung", "Verteidigung" und "Gambit" achten und Personen- und Städtenamen als Teil der Eröffnung kennzeichnen. Hinzu kommen falsch formulierte Eröffnungsamen des Nutzers (z. B. "Variante" statt "Gambit"). Das System sollte mit Ähnlichkeitsfunktionen, die richtige Eröffnung aus der Wissensbasis ermitteln.

Neben den Eröffnungen gibt es auch in den End- und Mittelspielen wiederkehrende Muster,

die bestimmte Bezeichnungen besitzen. Die Lucena-Position oder die Philidor-Stellung sind zwei bekannte Beispiele.

Auch einfache Phrasen können von Nutzern anders formuliert werden, wie z. B. statt "Remis" etwa "Das Spiel endet unentschieden". Weitere Schachbegriffe führt bspw. Wikipedia in einem Glossar auf (vgl. [65]). Diese können in einem Wörterbuch gesammelt werden, das die verschiedenen Synonyme beinhaltet.

*Semantic Chess* stellt ein Vokabular (vgl. Anhang C.1) bereit, in dem Synonyme bzw. Wortgruppen einem Hauptbegriff zugeordnet werden. Beispielsweise verweist es auf den Hauptbegriff "*game*" für die unterschiedlichen Begriffe einer Partie, wie "*match*", "*game*", "*encounter*" oder "*pairing*".

## 3.2 Ambiguität

Bei der Mehrdeutigkeit muss das QAS zwischen dem Fachvokabular und der alltäglichen Sprache unterscheiden können. Schach verwendet einige Begriffe anders als im Sprachgebrauch üblich. Beispielsweise bezeichnet die mittelalterliche Katapultwaffe Trébuchet Schachpositionen, bei der sich beide Seiten, im Zugzwang befinden, wenn sie am Zug sind (vgl. [5], S. 7f).

Ein anderes Beispiel im Englischen ist "*exchange*". Als Verb bezeichnet es den Abtausch von Figuren, als Substantiv, häufig mit bestimmten Artikel ("*the exchange*"), dagegen den Qualitätsvorteil ("Turm gegen Springer oder Läufer") einer Seite.

Auch in den Eröffnungen tauchen Personennamen häufiger auf. Es gibt mehrere Theoretiker und Großmeister, die sich mit verschiedenen Varianten beschäftigt haben. Ein Beispiel ist Siegbert Tarrasch. Seinen Namen tragen mehrere Varianten in der Französischen Verteidigung, der Spanischen Eröffnung und im abgelehnten Damengambit.

*Semantic Chess* muss den Kontext der Begriffe erkennen (wie z. B. bei "*exchange*") und für Wörter, die im Alltagssprachgebrauch eine andere Bedeutung besitzen, ein Wörterbuch führen. Da das System sich allerdings auf das Schach konzentriert, kann das System bei der Verwendung bestimmter Begriffe davon ausgehen, dass das Fachvokabular gemeint ist.

### 3.3 Mehrsprachigkeit

In der Weltspitze befinden sich Spieler u. a. aus den USA, Kuba, Norwegen, Frankreich, Russland und China. Höhepunkt des internationalen Schachs sind zum einen die Schacholympiade mit über 170 teilnehmenden Nationen (vgl. [20]) und der World Cup, an dem 128 Spitzenspieler der verschiedenen Kontinente im K.O.-Modus gegeneinander antreten (vgl. [31]). Dadurch kommen auch im Schach zahlreiche Sprachen zusammen, mit ihrem jeweils eigenem Vokabular für das Spiel.

Aufgrund der Jahrhunderte langen Geschichte des Schachs, die durch verschiedene Regionen in verschiedenen Zeitabschnitten unterschiedlich stark geprägt wurde, ist das englische Vokabular gemischt mit z. B. deutschen, italienischen und französischen Begriffen. Bekannte deutsche Begriffe sind u. a. Zugzwang und Luft, französische Vertreter sind das *En Passant* und *en prise*. Das Italienische hat bspw. den Begriff *Fianchetto* ins Schach gebracht. Ein Tagger und die Erkennung von Entitäten in englischer Sprache müssen um diese Wörter erweitert werden.

Ein Question-Answering-System sollte diese Punkte berücksichtigen, wenn es multilingual ausgerichtet ist. Das System könnte hier die verschiedenen Sprachversionen von Wikipedia abfragen, um die Begriffe zu erkennen. Die Eröffnungen können zusätzlich im RDF hinterlegt sein.

*Semantic Chess* konzentriert sich auf die englische Sprache. Eine nicht in Englisch gestellte Nutzerfrage versteht das System dagegen nicht.

### 3.4 Komplexe Anfragen

Auch in der *closed domain* eines QAS zum Schach kommen komplexe Anfragen vor. Wie in anderen Sportarten auch, kann im Schach u. a. nach Siegesserien oder Turniersiegern gefragt werden. Die Erstellung von SPARQL Queries kann sich hier als schwierig herausstellen, wenn nur PGN-Dateien als Wissensbasis dienen, da Siegesserien nicht explizit gespeichert werden. Eine Erweiterung der Daten mit zusätzlichen Informationen über den Ausgang von Turnieren oder zu Spielern muss hier im Vorfeld geprüft werden.

Bei *Semantic Chess* sind komplexe Anfragen solange möglich, wie das System für diese ein Template finden kann.

### 3.5 Verteiltes Wissen

Schachpartien werden im Web nicht als RDF abgespeichert. Dazu dient v.a. das Dateiformat PGN. Eine größere öffentliche Sammlung von Partien im RDF gab es auf der Webseite `salonica.dia.fi.upm.es:8080/rdfchess/` (vgl. [56], S. 3). Sie ist zur Zeit der Arbeit allerdings nicht erreichbar.

Ein QAS müsste zunächst eine größere Sammlung von Partien anlegen und dann eigenständig in RDF umwandeln. Öffentliche PGN-Datenbestände sind z. B. auf folgenden Seiten zu finden:

- `pgnmentor.com/files.html`
- `kingbase-chess.net`
- `sourceforge.net/projects/codekiddy-chess`
- `ficsgames.org/download.html` (Online-Partien)

Da es keine Partiensammlung in RDF gibt, kann *Semantic Chess* diese Herausforderung nicht angehen. Es erstellt aus den PGN-Dateien eine eigene RDF-Datenbasis.

### 3.6 Prozedurale, zeitliche und räumliche Fragen

Prozedurale Fragen sind, wie bei anderen QAS, nicht oder nur schwer zu beantworten. Für Fragen (vgl. Punkt 1 in Tab. 3.6 ), wie *“How do I win in the Queen’s Gambit?”* kann es zwei Lösungswege geben. Entweder kann das System den Nutzer auf Bücher von Theoretikern verweisen oder die Variante zeigen, die nach einer Analyse der gespeicherten Partien die höchste Gewinnchance bietet. Bei Fragen (2), wie *“What does the Epaulette Mate look like?”* müsste das System dem Nutzer ein Diagramm präsentieren oder die Stellung beschreiben. Fragen (3) nach dem Ausgang einer Partie in einer bestimmten Stellung können über eine Endspieldatenbank bis sieben Figuren auf dem Brett erfragt (vgl. [16]) oder von einer Schach-Engine eingeschätzt werden. Zeitliche Fragen (4) sind durch die Metadaten der PGN-Dateien oder die geordnete Abfolge von Zügen zu lösen. Begriffe wie *“zuerst”*, *“zuletzt”*, *“vorher”* oder *“nachher”* sollten in einem Wörterbuch definiert sein. Räumliche Fragen (5) können sich auf den Ort der Partie oder auf die Stellung der Figuren auf dem Schachbrett beziehen. Ersteres ist durch die Metadaten einer Partie zu



Fragen- typ	Beispielfrage	mögliche Lösung	von Semantic Chess zu beantworten
1	<i>"How do I win in the Queen's Gambit?"</i>	- Verweis auf Bücher - Variante mit höchsten Gewinnchancen anzeigen	nein
2	<i>"What does the Epaulette Mate look like?"</i>	- Diagramm anzeigen	nein
3	<i>"How does the game end?"</i>	- Endspieldatenbank verwenden - Einschätzung durch eine Schach-Engine	nein
4a	<i>"When should I develop the queen?"</i>	- theoretisches Wissen im System hinterlegt	nein
4b	<i>"When did the first World Championship take place"</i>	- durch Meta-Informationen in PGN-Datei	ja, durch Speicherung der Meta-Informationen im RDF-Format
5a	<i>"Where is the best place for my knights on the board?"</i>	- theoretisches Wissen im System hinterlegt	nein
5b	<i>"Where did the last World Champion take place?"</i>	- durch Meta-Informationen in PGN-Datei	ja, durch Speicherung der Meta-Informationen im RDF-Format

Tabelle 3.2: Überblick zu den Fragentypen (1-3: prozedurale Fragen, 4a-b: zeitliche Fragen, 5a-b: räumliche Fragen)

beantworten. Für letzteres benötigt das System ein Verständnis von Stellungen, Materialverhältnis und Beziehungen der Figuren untereinander. Als Index für das Materialverhältnis kann der GBR-Code (vgl. Kap. 2.7.4) dienen. Dieser wird bereits dazu genutzt Positionen in Endspieldatenbanken zu indexieren (vgl. [8]). Stellungen können außerdem mit FEN abgebildet und Beziehungen als Graph (vgl. Kap. 2.7.3) im RDF eingepflegt werden.

*Semantic Chess* kann diese Fragen teilweise beantworten. Sie müssen sich allerdings durch die Metadaten und Züge einer Partie erschließen, wie z. B. der Metawert für das Datum ("*date*"), über den dann die zeitlichen Fragen beantworten werden können.

### 3.7 Templates

Bei schwierigen Formulierungen einer SPARQL Query, wie im Abschnitt zu den komplexen Anfragen oder den prozeduralen Fragen erwähnt, können Templates helfen. Wenn das QAS diese Fragentypen identifiziert, kann es im nächsten Schritt versuchen die entsprechende Vorlage auszuwählen und die Frage zu beantworten. Templates besitzen den Vorteil, dass

sie eine genauere semantische Struktur abbilden können als Tripel (vgl. [18], S.639). Allerdings können Fragen nicht beantwortet werden, sobald keine Vorlage existiert, die der Frage entspricht.

*Semantic Chess* besitzt eine Sammlung von Templates, um damit die Fragen zu beantworten.

# Kapitel 4

## Ansatz

Das QAS *Semantic Chess* durchläuft für jede Frage vier Zustände und sieben Prozesse, um dem Nutzer antworten zu können (vgl. Abb. 4.1). Die folgenden Abschnitte sollen den Ablauf von der Analyse über die Auswahl eines Templates bis hin zur Ergebnispräsentation erklären. Das Beispiel *“Which player with black defeated Howard Staunton in December 1843?”* dient zur Veranschaulichung der Zustände und Prozesse.

### 4.1 Logischer Aufbau

#### Prozess: Nutzer stellt Frage

Der Nutzer kann eine in englischer Sprache verfasste Schach bezogene Frage stellen. Dabei sollte der Nutzer sich vor allem an der Struktur der Metadaten und den Zügen einer PGN-Datei orientieren. Folgende Beispielfragen kann ein Nutzer stellen:

- *Against which players did Magnus Carlsen win in 2016?*
- *Show me endgames with rook against rook and pawn.*
- *Give me games in which Wilhelm Steinitz won with white against Emanuel Lasker.*
- *Are there games with the opening moves f4 d5 Nf3?*
- *What does Levon Aronian play against the Semi Slav Defense?*

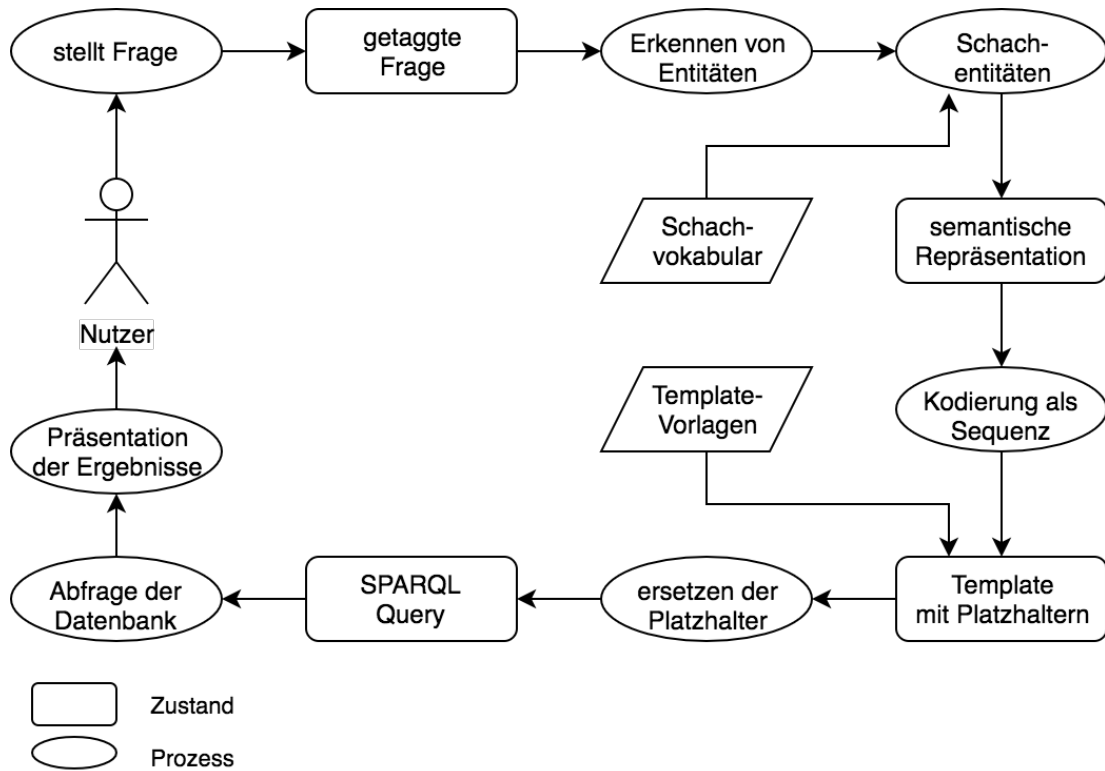


Abbildung 4.1: Flussdiagramm von Semantic Chess

Die Möglichkeiten unterteilen sich in Fragen zu Partien, Spielern und ihren Wertzahlen (Elo), Events, Runden der Events, Datumswerte, Resultate, Eröffnungen, Zügen und Kombinationen aus diesen.

### Zustand: Getaggte Frage

Das System analysiert die Frage zunächst mit einem POS Tagger und speichert die verschiedenen Wortarten ab. Die Tags dienen später dazu, herauszufinden in welchem Kontext die Wörter stehen, wie z. B. das im vorherigen Kapitel genannte *“the exchange”*.

Zeiten und Datumswerte werden auf das für die angelegten SPARQL Queries geeignete Format *“YYYY.MM.DD”* mit Hilfe des *Stanford POS Taggers* umformatiert. Der übliche Datentyp *xsd:date* mit dem Format *“YYYY-MM-DD”* wird nicht verwendet, da Datumswerte in der Datenbank als Zeichenkette, aufgrund möglicher fehlender Angaben zu Tagen, Monaten oder Jahren, gespeichert sind.

Für das Beispiel *“Which player with black defeated Howard Staunton in December 1843?”* werden folgende Tags abgespeichert:

*“Which|WDT player|NN with|IN black|JJ defeated|VBN Howard|NNP Staunton|NNP in|IN December|NNP 1843|CD ?|.”*

Der *Stanford POS Tagger* erkennt *which* als Wh-Fragewort, *player* als Substantiv, *Howard*, *Staunton*, *December* als Eigennamen, *black* als Adjektiv und *1843* als Kardinalnummer. *December 1843* speichert der Prozess als *1843.12* ab.

### **Prozess: Erkennen von Entitäten**

Das System versucht im Anschluss benannte Entitäten zu erkennen (engl., *named entity recognition*), wie z. B. Personen- oder Ortsnamen. Das *Stanford coreNLP Framework* unterstützt diesen Schritt. *Semantic Chess* konzentriert sich hier zunächst auf die Alltagssprache. Im nächsten Prozess ermittelt das QAS mit Hilfe der hier entdeckten Entitäten und eines Schachvokabulars fachspezifische Entitäten (vgl. Abschnitt "Prozess: Schachentitäten"). Sollte eine Phrase schon als Entität in der Alltagssprache entdeckt sein, wird die Schachentität dann immer höher gewichtet.

Für das Beispiel erkennt das System mit Hilfe des *Stanford coreNLP* "*Howard Staunton*" als Person und "*December 1843*" als Datum. Bei einer Anfrage, wie "*Show me games with the Staunton Gambit.*", kann das System zunächst einen Personennamen aus der Kombination "*Staunton Gambit*" interpretieren. Wegen des Schlüsselwortes "Gambit" wird sie trotzdem durch den folgenden Prozess als Eröffnung erkannt.

### **Prozess: Schachentitäten**

*Semantic Chess* legt für die gefundenen Entitäten zwei Sammlungen an. Es unterscheidet dabei zwischen den benannten und den für diese Arbeit so bezeichneten "unbenannten" Entitäten. Die benannten Entitäten beziehen sich u. a. auf Personen, Orte und Gegenstände, die einen Namen tragen. Das kann z. B. der Schachspieler "Bernhard Horwitz", die Stadt "Amsterdam" oder die Schachfigur "Springer" sein. Formuliert der Nutzer diese benannten Entitäten in seiner Frage, bezieht dieser sich also auf etwas Konkretes. Zusätzlich überprüft das System diese mit einer Ähnlichkeitsfunktion (Jaccard-Distanz), um gleiche Namen in der Datenbank zu finden. Die Schreibweise "William Steinitz" wird dann als "Wilhelm Steinitz" erkannt. Für das Erkennen von Orten, Personen, Organisationen, Geld, Prozent, Datumswerten und Zeitangaben ist das *Stanford coreNLP* aus dem vorangehenden Prozess zuständig. Schachspezifische Begriffe versucht das System mit dem Schachvokabular abzudecken. Dafür stehen Funktionen zur Verfügung, die die Wörter überprüfen, ob diese im Vokabular enthalten sind, einem der hinterlegten regulären Aus-

drücke entsprechen (z. B. um ECO-Codes, wie *C33* oder Züge, wie *Nxf3* zu erkennen) oder ob die vom *Stanford coreNLP* entdeckte Entität ein Schlüsselwort enthält. Ist das letztere der Fall, überschreibt das System die Entität mit der Schachentität. Die Funktionen werden im Kapitel 4.4.5 genauer beschrieben.

Die unbenannten Entitäten sind dagegen allgemeine Begriffe, bei denen der Nutzer die konkrete Bezeichnung nicht kennt oder nach Gruppen fragt. Beispielfragen für die unbenannten Entitäten sind u. a. *“In which cities did Bernhard Horwitz play?”* oder *“Against which players did Bernhard Horwitz win?”*. Bei der ersten Frage kennt der Nutzer die konkreten Stadtnamen nicht und möchte diese erfahren. Bei der zweiten geht es um die Gruppe der Spieler, gegen die Bernhard Horwitz gewinnen konnte. Diese Entitäten werden ausschließlich entdeckt, wenn sie im Schachvokabular hinterlegt sind.

In diesem Prozess erkennt das System noch weitere Bestandteile für die SPARQL Query. Zunächst entscheidet das Programm, ob die SPARQL Query eine Vereinigung (*UNION*) benötigt. Das ist bspw. der Fall, wenn der Nutzer nach Sieg- oder Verlustpartien fragt: *“Who won against Wilhelm Steinitz?”*. Danach folgen Entscheidungen darüber, ob die Anfrage Filter (*FILTER*), Optionen (engl. *modifier*, wie z. B. *LIMIT*, *OFFSET*) oder Aggregationen (*GROUP BY*, *HAVING*) erhalten soll. Filter treten bei Größenvergleichen (z. B. *“more than 2600 elo”*) oder bei regulären Ausdrücken (*regex*-Filter, z. B. bei Fragen nach Positionen) auf. Ein Beispiel hierfür ist: *“Show me games with rook against rook and pawn”*. Das System erkennt die Figuren Turm und Bauer als Entitäten, wobei der erstgenannte Turm als weiße Figur gilt und die Figuren Turm und Bauer nach dem Schlüsselwort *“against”* als schwarze Steine zählen. Danach legt das System diese in einem vorgefertigten regulären Ausdruck ab. Mit diesem können dann die FEN-Zeichenketten durchsucht und die entsprechende Position gefunden werden. Optionen werden in die Anfrage aufgenommen, wenn der Nutzer ordinale (*“first”*, *“second”*, *“third”*) für den *OFFSET* oder kardinale Zahlen (*“one”*, *“two”*, *“three”*) für das *LIMIT* verwendet. Aggregationen treten nur in Verbindungen mit Funktionen in der *SELECT*-Klausel auf, wie z. B. *COUNT*, *MAX*, *MIN* oder *AVG* (*“average”*), da diese eine Gruppierung der Elemente verlangen. Eine Beispielfrage lautet: *“How often did Wilhelm Steinitz win in 1894?”*. Durch das Schlüsselwort *“often”* erkennt das System, dass es hier die Anzahl der Gewinnpartien von Wilhelm Steinitz im Jahr 1894 zählen soll.

Im letzten Schritt analysiert das Programm das Thema der Frage, um damit die *SELECT*-Klausel zu füllen. Das System bevorzugt hier vor allem die Entitäten, die am Anfang der Frage gefunden wurden. Das können z. B. die englischen Wh-Fragepronomen (z. B. *“who”* oder *“where”*) oder ein Schlüsselwort sein (z. B. *“games”* in der Anfrage *“Show me games*

where..."). Dadurch erhält der Nutzer die gewünschten Informationen und nicht alle im Prozess angelegten Variablen, wie z.B Links zu einzelnen Ressourcen, die für den Nutzer keinen Mehrwert bieten. Fragt der Nutzer nur nach Partien, wird nicht nur die URL zur Partie ausgegeben, sondern zusätzliche Informationen, wie Spieler und Datum. Bei Fragen nach Häufigkeiten, maximalen oder minimalen Werten schreibt der *Parser* die entsprechende Aggregatsfunktion in die Klausel (*COUNT*, *MAX* oder *MIN*).

Aus "Which player with black defeated Howard Staunton in December 1843?" wird "Howard Staunton" in die Sammlung der benannten Entitäten aufgenommen. Außerdem erstellt das QAS die Entität "0-1" aus den erkannten fachspezifischen Begriffen "defeated" und "black". Dadurch kann die Person "Howard Staunton" nur als Weißspieler im Resultat auftauchen.

Das Datum "December 1843" ist unvollständig. Es fehlt der Tag. Daher legt das System eine unbenannte Entität "date" an und zusätzlich einen Filter mit einem regulären Ausdruck für "1843.12". Zu dieser Sammlung kommt noch der Begriff "player", der als unbenannte Person erkannt wurde und im Resultat nur als Schwarzspieler erscheint.

Der Grund für den Einsatz von regulären Ausdrücken bei Datumswerten ist, dass diese als *String* (Zeichenkette) in der Datenbank liegen. PGN-Dateien sind nicht immer vollständig ausgefüllt, vor allem gilt dies für Partien aus dem 19. Jahrhundert. Fehlende Tage, Monate oder Jahre, aber auch andere Metadaten werden daher als "???" ergänzt, z. B. "1843.12.???".

## **Zustand: Semantische Repräsentation**

Die zwei Sammlungen der benannten und unbenannten Entitäten werden in einer semantischen Repräsentation abgelegt und dienen später dazu die Platzhalter in den Templates zu ersetzen. Eine semantische Repräsentation besteht aus einem Tripel (Subjekt, Prädikat, Objekt) und der Position der Entität in der Frage. Die Entität ist meistens das Objekt. Das dazu passende Subjekt und Prädikat wählt das System aus. Beispielsweise legt es für eine Person immer das Prädikat *prop:white* (Weißspieler) bzw. *prop:black* (Schwarzspieler) an und ordnet es dem Subjekt *?game* (Partie) zu. Ursache hierfür ist die Struktur einer PGN-Datei, in der eine Person immer nur als Schwarz- oder Weißspieler einer Partie (*?game*) geführt wird. Genauso verhält es sich mit anderen Entitäten, wie z. B. dass Datumswerte immer *prop:date* und Ortsnamen immer *prop:site* zugeordnet werden. Die Entität kann auch Subjekt sein, wenn von ihr weitere Eigenschaften zur Beantwortung der Frage notwendig sind, wie z. B. "Show me games with the opening King's

*Gambit*.”. Hier möchte der Nutzer Partien als Antwort erhalten, die die Eröffnung mit der Eigenschaft “Königsgambit” thematisieren. Das System baut daher eine Verbindung zwischen den Klassen “ChessGame” und “ChessOpening” aus der Ontologie 2.7 auf, um anschließend Partien mit dem entsprechenden Eröffnungsnamen zu finden.

Folgende semantische Repräsentationen werden für das Beispiel “Which player with black defeated Howard Staunton in December 1843?” erstellt:

Nr.	Index Subjekt	Subjekt	Index Prädikat	Prädikat	Index Objekt	Objekt	Pos.
1	R_1	?game	P_1	prop:result	E_1	'0-1'	4
2	R_2	?game	P_2	prop:white	E_2	'Howard Staunton'	5-6

Tabelle 4.1: Semantische Repräsentation benannter Entitäten

Nr.	Index Subjekt	Subjekt	Index Prädikat	Prädikat	Index Objekt	Objekt	Pos.
1	S_1	?game	D_1	prop:black	C_1	?player	1
2	S_2	?game	D_2	prop:date	D_2	?date	9

Tabelle 4.2: Semantische Repräsentation unbenannter Entitäten

Die Struktur der Tripel lässt sich wie folgt aus den Tabellen 4.1 und 4.2 ablesen. Die Subjekte sind mit R bzw. S gekennzeichnet, Prädikate mit P bzw. D und Objekte mit E bzw. C. Sie tragen zusätzlich eine Zahl (z. B. “E\_1” oder “P\_2”), die die Reihenfolge des Eintrags in die Tabelle zeigt und eine zusätzliche Unterscheidung ermöglicht. Ein Tripel lautet bspw. *?game*→*prop:result*→*'0-1'*. Der Nutzer hat hier nach einem Schwarzsieg gefragt (“0-1”). Das Schlüsselwort “defeated” befindet sich im Schachvokabular. Dieses verweist auf die Entität (“1-0”), das durch das Schlüsselwort “black” zu (“0-1”) geändert wird. Im System ist fest implementiert, dass dieser Entität das Prädikat “*prop:result*” zugeordnet wird. Das Subjekt *?game* wählt das System deswegen aus, weil das Resultat eine Eigenschaft einer Partie ist (vgl. Ontologie 2.7). Aus dieser semantischen Repräsentation lässt sich der Graph 4.2 erstellen.

Die Position dient dazu Beziehungen der Entitäten untereinander zu ermitteln. Beispielsweise wird in der Frage “Show me games by Wilhelm Steinitz with white against Emanuel Lasker” aufgrund der Positionen der Entitäten dem Spieler “Wilhelm Steinitz” die weißen Steine (“white”) zugeordnet. Das geschieht mit Hilfe einer Abstandsfunktion, die die Entfernung des Schlüsselworts “white” zwischen den einzelnen Personen misst. Der Person



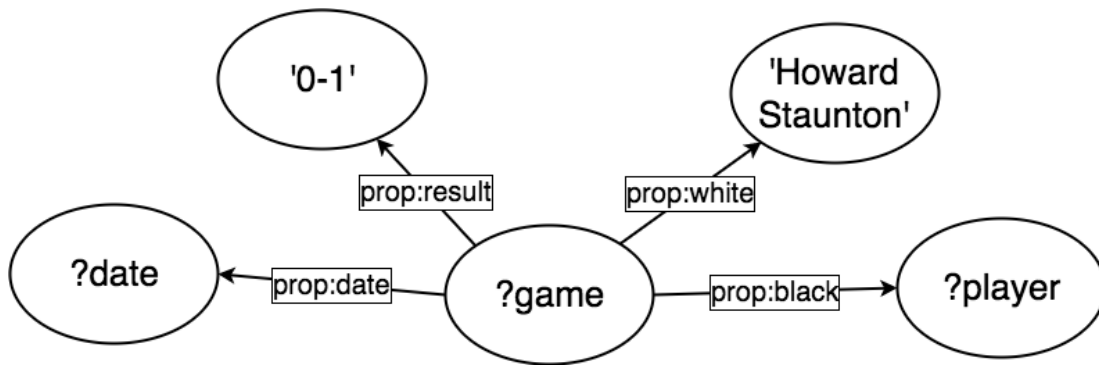


Abbildung 4.2: Graph für das Beispiel “Which player with black defeated Howard Staunton in December 1843?”

mit dem niedrigeren Wert wird die Farbe zugeordnet. Bei gleichem Abstand entscheidet sich das System für die erstgenannte Person.

Das System setzt im Hintergrund *Flags* (Markierungen) für die anderen Bestandteile einer SPARQL Query, wie z. B. *FILTER* (vgl. mit Abschnitt “Prozess: Schachentitäten”). Datumswerte sind nur als Zeichenkette in der Datenbank gespeichert, die für eine exakte Abfrage die Form *yyyy.mm.dd* haben müssen. Da im Beispiel der Tag fehlt, legt das System einen Filter an und überprüft mit einem regulären Ausdruck alle in der Datenbank gespeicherten Datumswerte.

### Prozess: Kodierung als Sequenz

Das System bildet eine Sequenz aus den gesammelten Entitäten und Informationen. Die Sequenz ist gleichzeitig der Index für vorher manuell angelegte Templates. Sie besteht aus folgenden Informationen:

- Anzahl der benannten Entitäten
- Anzahl der unbenannten Entitäten
- Vereinigung vorhanden — ja (1) bzw. nein (0)

Aus der Beispielfrage entsteht die Sequenz 220. Die Sequenz beschreibt zwei benannte, zwei unbenannte Entitäten und keine Vereinigung (*UNION*).

## Zustand: Template mit Platzhaltern

Im Vorfeld wurden Templates für SPARQL Queries manuell eingepflegt. Sie haben die Grundform:

```
1 SELECT DISTINCT *
2 WHERE {
3   R_1 P_1 E_1 .
4   FILTER
5 }
```

Listing 4.1: Grundform eines manuell angelegten SPARQL Template, Sequenz: 010

Notwendige Präfixe (*PREFIX*) sind ebenfalls im System hinterlegt und kommen erst bei der Ausführung der SPARQL Query zum Einsatz. Ein Beispiel ist das in Tab. 4.1 erwähnte “prop:”, das die Kurzform für `http://example.com/prop/` darstellt.

Vorlagen mit einer Vereinigung haben die Form:

```
1 SELECT DISTINCT *
2 WHERE {
3   {R_1 P_1 E_1 .}
4   UNION
5   {R_2 P_2 E_2 .}
6   FILTER
7 }
```

Listing 4.2: Grundform eines manuell angelegten SPARQL Template mit Vereinigung, Sequenz: 021

Sequenzen mit höheren Nummern erhalten dementsprechend die Anzahl der Entitäten.

Für das Beispiel nimmt das System die Vorlage:

```
1 SELECT DISTINCT *
2 WHERE {
3   R_1 P_1 E_1 .
4   R_2 P_2 E_2 .
5   S_1 D_1 C_1 .
6   S_2 D_2 C_2 .
7   FILTER
8 }
```

Listing 4.3: Template für die Beispielfrage, Sequenz: 220

Sollte kein Template für die Sequenz angelegt sein, erhält der Nutzer eine Fehlermeldung.

## Prozess: Ersetzen der Platzhalter

Im nächsten Schritt tauscht das System die Platzhalter aus, in dem es die Subjekte, Prädikate und Objekte mit den entsprechenden Indizes (vgl. Tab. 4.1 und 4.2) an die gewünschte Position einsetzt.

In diesem Schritt wird der Platzhalter *FILTER* mit den vom System ermittelten Filtern ersetzt, das Thema in die *SELECT*-Klausel geschrieben und Optionen und Aggregationen an die SPARQL Query angehängt (vgl. mit Abschnitt "Prozess: Schachentitäten").

## Zustand: SPARQL Query

Aus dem vorherigen Prozess entsteht eine SPARQL Query zur Abfrage der Datenbank. Für die Beispielfrage "Which player with black defeated Howard Staunton in December 1843?" entsteht dadurch die folgende Anfrage:

```
1 PREFIX prop:<http://example.com/prop/>
2
3 SELECT DISTINCT ?player
4 WHERE {
5     ?game    prop:result    '0-1' .
6     ?game    prop:white     'Howard Staunton' .
7     ?game    prop:black     ?player .
8     ?game    prop:date      ?date .
9     FILTER   regex(?date , '1843.12' , 'i')
10 }
```

Listing 4.4: SPARQL Query für die Beispielfrage

## Prozesse: Abfrage der Datenbank und Präsentation der Ergebnisse

Das System führt jetzt die fertige Anfrage aus. Die Ergebnisse aus der Datenbank *Virtuoso* werden gesammelt. Im Anschluss erhält der Nutzer die Antwort in einer Tabelle.

Die Beispielfrage liefert folgende Antwort zurück:

player
Pierre Charles Fournier deSaint Amant

Tabelle 4.3: Resultat der Beispielfrage

## 4.2 Implementierung

Der technische Aufbau für die Implementierung des QAS wird in diesem Abschnitt vorgestellt und an Beispielen verdeutlicht. Auf <https://github.com/dice-group/semanticchess> ist der gesamte Code zu finden.

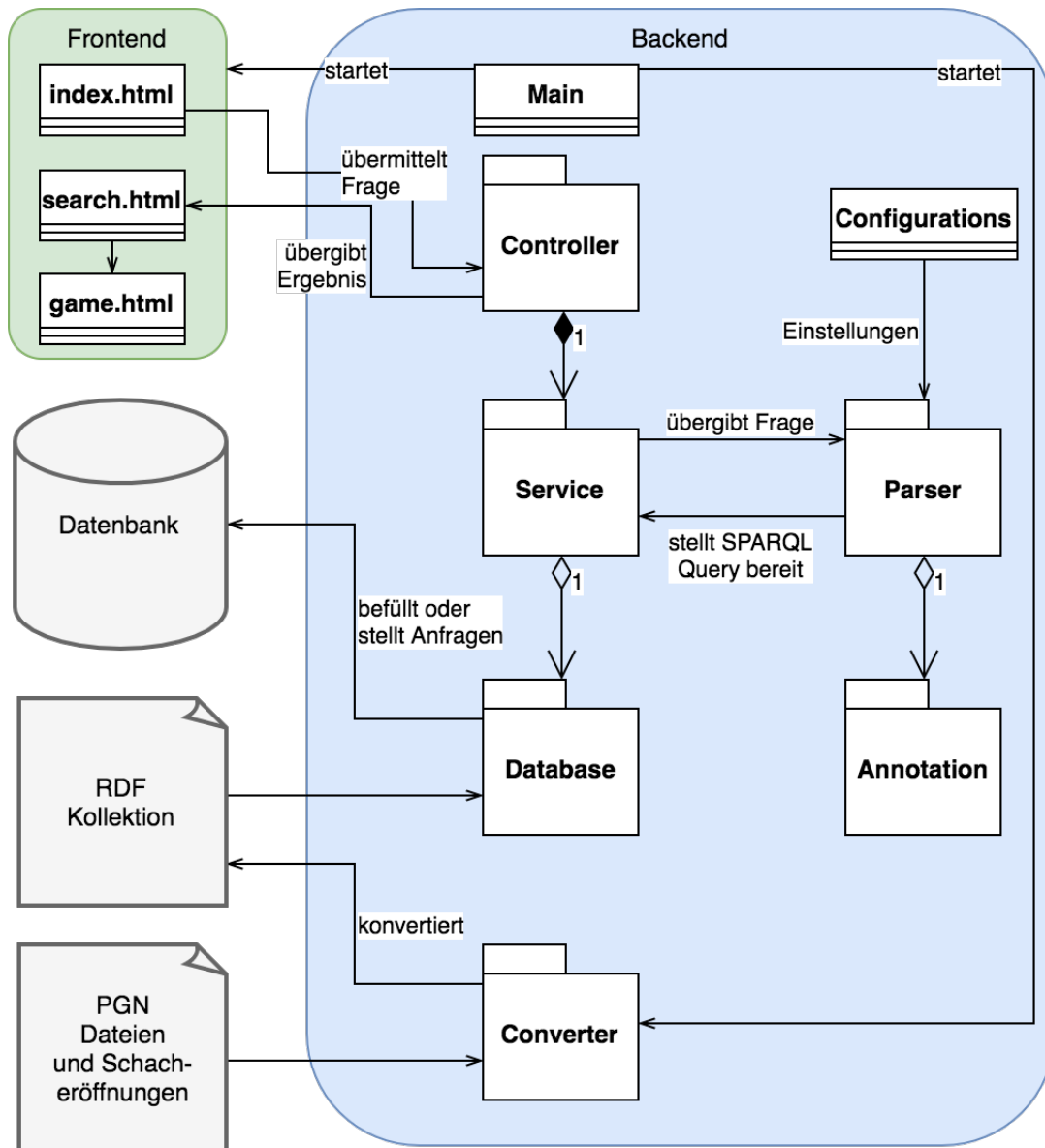


Abbildung 4.3: UML-Diagramm von Semantic Chess

Das System unterteilt sich in die drei Hauptkomponenten Frontend, Backend und Datenbank (vgl. Abb. 4.3). Dabei liegt das Hauptaugenmerk der Erklärung auf dem Backend mit dem darin enthaltenen Parser. Dieser übernimmt die meisten der vorgestellten Prozesse und Zustände aus dem vorangehenden Abschnitt. Die Datenbank speichert die Schachdaten im RDF-Format ab und das Frontend dient zur Kommunikation des Nutzers mit dem System.

## 4.3 Frontend

Das Frontend enthält die Weboberfläche, mit der der Nutzer interagieren kann. Hier kann er seine Frage formulieren, die Ergebnisse betrachten und Schachpartien nachspielen.

Folgende Technologie wurden für den Aufbau des Frontends verwendet:

- HTML
- CSS
  - Bootstrap 4
  - pgnvjs.css (PgnViewerJS)
- JavaScript
  - AngularJS
  - Angular-Route.js
  - PgnViewerJS
  - FontAwesome

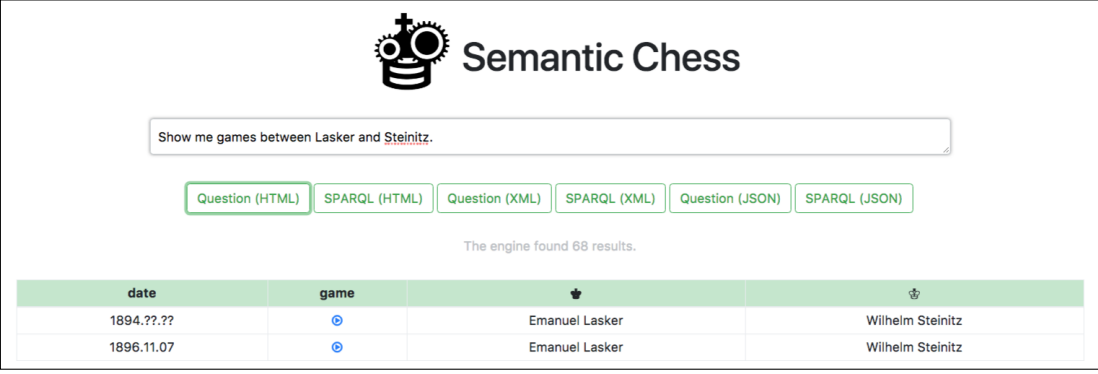
### **index.html**

Das Frontend besteht aus drei Webseiten (vgl. Abb. 4.3). Die Hauptseite, *index.html* dient zur Organisation der benutzten Technologien (CSS, JavaScript) und zur Bereitstellung eines Views (Ansicht) für die Seiten *search.html* und *game.html*. Das geschieht mit Hilfe von *Angular-Route.js*, um nicht die gesamte Website bei jedem Seitenaufruf neu laden zu müssen.

### **search.html**

In *search.html* ist das Eingabefeld und die Ergebnispräsentation implementiert. In das Eingabefeld kann der Nutzer seine Frage schreiben, die dann mit Hilfe des *\$http*-Service (*HTTP-POST*) von AngularJS an das Backend gesendet wird. Für die Präsentation wertet AngularJS das vom Backend zurückgelieferte *JSON* aus.

Der Nutzer hat weitere Möglichkeiten zur Interaktion mit der Seite. Neben dem Stellen von Fragen in natürlicher Sprache, kann er auch eigene SPARQL Queries schreiben, die



The screenshot shows the Semantic Chess website. At the top is a logo with a chess knight and the text "Semantic Chess". Below it is a search bar containing the text "Show me games between Lasker and Steinitz.". Under the search bar are six buttons: "Question (HTML)", "SPARQL (HTML)", "Question (XML)", "SPARQL (XML)", "Question (JSON)", and "SPARQL (JSON)". Below the buttons, it says "The engine found 68 results.". At the bottom is a table with four columns: "date", "game", and two columns with knight icons. The table contains two rows of results.

date	game	♠	♠
1894.???	♣	Emanuel Lasker	Wilhelm Steinitz
1896.11.07	♣	Emanuel Lasker	Wilhelm Steinitz

Abbildung 4.4: Beispiel der Suchseite und des Suchergebnisses

dann direkt an die Datenbank gesendet werden. Bei der Darstellungsform kann er statt der Tabelle das Ergebnis auch in RDF (RDF/XML-Serialisierung) oder im JSON-Format anzeigen lassen. Das soll Nutzern mit größerem Hintergrundwissen die Option bieten, die Datenbank und das RDF weiter zu nutzen und zu erforschen.

### game.html

Falls das Ergebnis Verlinkungen zu Partien enthält, kann der Nutzer sie sich auf der Seite *game.html* ansehen. PgnViewerJS und das dazugehörige CSS stellen die nötige Logik dafür bereit. Zu diesem Zweck erstellt das Backend aus dem RDF einer Partie ein PGN-Format, das dann vom Skript verarbeitet werden kann.

Zur Gestaltung der Website nutzt das Frontend Bootstrap. Dieses bietet bereits Vorlagen, um die Optik der Elemente, wie z. B. der Tabelle aufzuwerten. Da in der vierten Version von Bootstrap Icons nur noch über Drittanbieter eingebunden werden können (vgl. [12]), hilft FontAwesome aus.

## 4.4 Backend

Das Backend enthält alle Funktionalitäten, um eine Frage in ein SPARQL Query zu konvertieren und die Kommunikation mit der Datenbank durchzuführen. Es ist in Java 8 geschrieben und besteht aus sechs Hauptkomponenten, sowie einer Haupt- und einer Konfigurationsklasse. Mit der Hauptklasse (*Main*) kann das System gestartet werden. Die Konfigurationsklasse enthält häufig verwendete und wichtige Daten, wie z.B die Zugangsdaten für die Datenbank. Folgende Technologien wurden für das Backend verwendet:

- Spring Boot von <http://spring.io/>
- coreNLP von <https://nlp.stanford.edu/>
- Jena ARQ von <http://jena.apache.org/>
- swp-sc13 aus dem Softwaretechnik-Praktikum 2013 (vgl. Abb. 2.7)
- commons-text von <https://commons.apache.org/>

Spring Boot regelt die Kommunikation zwischen Front- und Backend. Es erzeugt REST-Schnittstellen und baut eine MVC-Umgebung auf (**M**odel **V**iew **C**ontroller) (vgl. [53]), um die Funktionalitäten des QAS zu trennen.

#### 4.4.1 Controller

Im Paket *Controller* sind die REST-Schnittstellen implementiert. Sie warten auf Anfragen des Frontends und leiten diese an die passende Klasse im Paket *Service* weiter (vgl. Abb. 4.5).

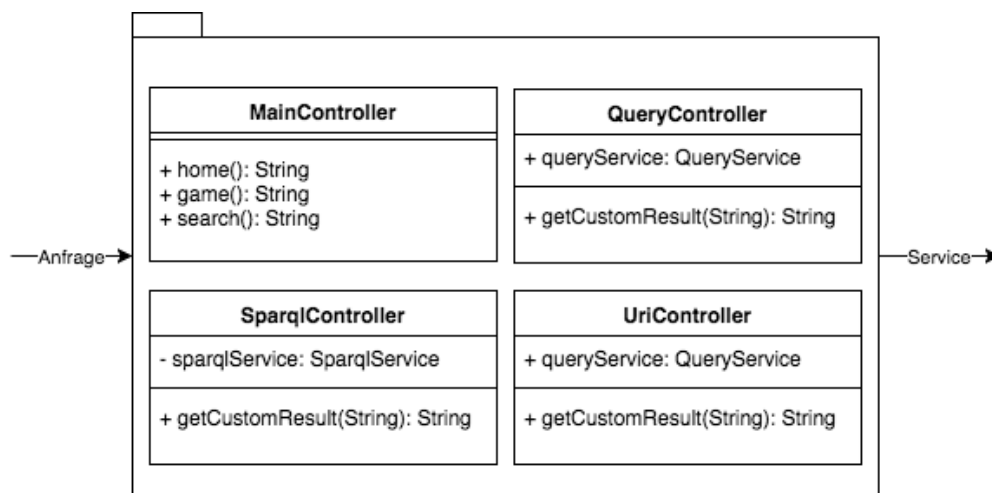


Abbildung 4.5: UML-Diagramm des Pakets: Controller

Die Klasse *MainController* registriert die gewünschten Seiten und ordnet ihnen eine HTML-Seite zu (vgl. Listing 4.5). Für das *Mapping* sorgt Spring Boot.

```

1  @GetMapping("/game")
2  public String game() {
3      return "game";
4  }
  
```

Listing 4.5: Beispiel: Registrierung der Seite "game"

Die anderen Klassen leiten entweder eine Frage (*QueryController*), eine SPARQL Query (*SparqlController*) oder eine URI (*UriController*) an den entsprechenden Service weiter. Sie dienen als REST-*Controller* und warten auf mögliche Anfragen des Frontends.

#### 4.4.2 Service

Die *Service*-Klassen rufen, wenn nötig, verarbeitende Klassen auf (vgl. Abb. 4.6).

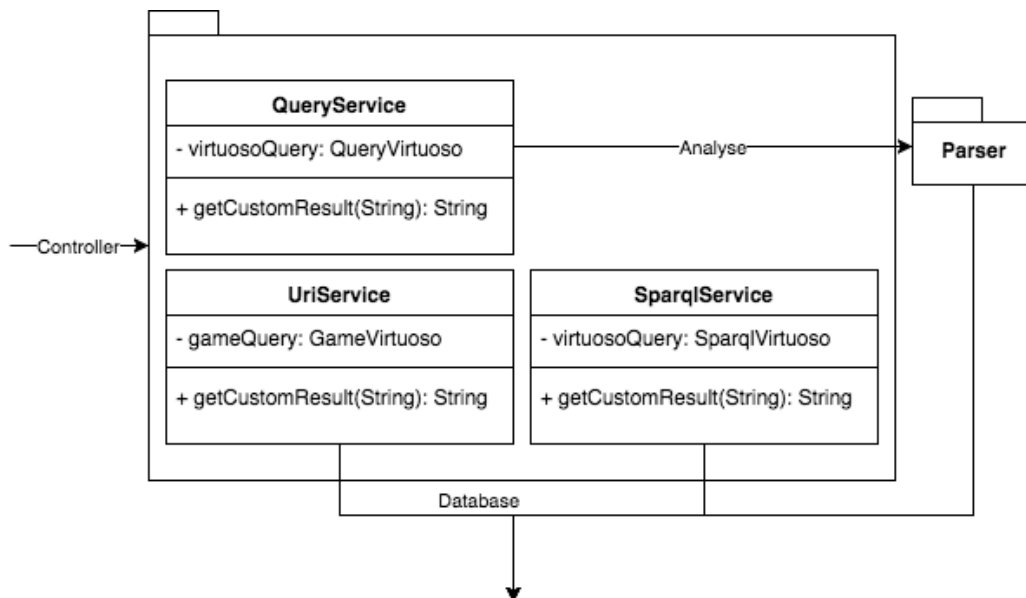


Abbildung 4.6: UML-Diagramm des Pakets: Service

*QueryService* leitet die Frage vom *Controller* erst über den *Parser*, um den daraus gebildeten SPARQL Query weiter an das Paket *Database* zu senden. Außer der direkten Weiterleitung an *Database* sind für *SparqlService* und *UriService* keine weiteren Verarbeitungsschritte implementiert. Sie stehen für zukünftige Verarbeitungsschritte bereit und sollen SPARQL Queries und Anfragen, die nur eine URI enthalten, voneinander trennen.

Die Informationen der Klassen aus *Database* werden über das Paket *Service* zurück an die *Controller* gesendet.

#### 4.4.3 Database

Das Paket *Database* regelt die Kommunikation mit der eingesetzten Datenbank *Virtuoso* (Version: 7.2.4) (vgl. Abb. 4.7). Die Klasse *ConnectVirtuoso* ermöglicht den Verbindungsaufbau, *LoadData* sorgt für das Befüllen der Datenbank mit RDF-Daten. Letztere basiert auf einer Vorlage des Unternehmens *OpenLink Software* (vgl. [61]) und nutzt die Funk-



tionalitäten von Jena ARQ. *GameVirtuoso*, *QueryVirtuoso* und *SparqlVirtuoso* sind die entsprechenden Klassen für die Funktionalitäten aus dem Paket *Service*.

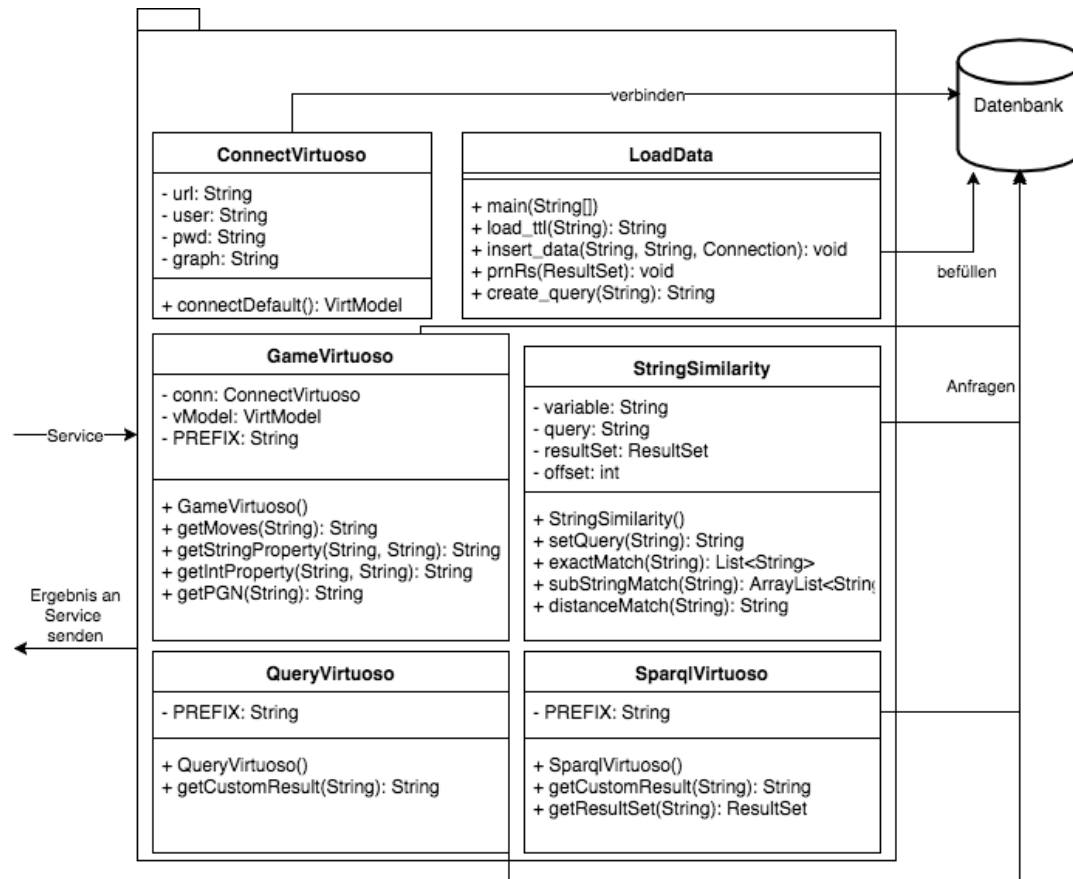


Abbildung 4.7: UML-Diagramm des Pakets: Database

*GameVirtuoso* sammelt auf Anfrage des *UriService* die RDF-Metadaten (*getStringProperty()* oder *getIntProperty()*) und -Züge (*getMoves()*) für eine Partie zusammen. Daraus erstellt es eine PGN-Datei (*getPGN()*), die dann zurück an das Paket *Service* geschickt wird.

Die Klassen *QueryVirtuoso* und *SparqlVirtuoso* leiten die SPARQL Queries zur Ausführung an die Datenbank. Die zwei Klassen dienen zur Trennung der beiden Informationsflüsse, die entweder nur eine reine SPARQL Anfrage besitzen oder zunächst eine Nutzeranfrage umwandeln müssen. Die Aufteilung in zwei Klassen soll es auch ermöglichen in Zukunft für beide Fälle individuelle Funktionen zu implementieren. Das Ergebnis wird zurück an *Service* gesendet.

*StringSimilarity* ist eine Hilfsklasse für den Parser. Sie vergleicht die gefundenen Entitäten mit den gespeicherten Informationen in der Datenbank. Dazu kann sie drei verschiedene Funktionen nutzen. *exactMatch* sucht nach der genauen Schreibweise. *subStringMatch* untersucht die Zeichenketten nach gleichen Bestandteilen. Das heißt, falls der *Parser* die Person "Steinitz" findet, findet die Methode in der Datenbank "Wilhelm

*Steinitz*“ und *“V Steinitz*“ und liefert die Ergebnisse zurück. Die Funktion *distanceMatch* setzt die Jaccard-Distanz als Ähnlichkeitsmaß ein und zusätzlich ein *substring*-Vergleich. Die Jaccard-Distanz misst die Unähnlichkeit von zwei Zeichenmengen (vgl. [21]). Die Implementierung in *commons-text* von *org.apache.commons* ermittelt die Werte für die Entität mit jedem möglichen Kandidaten in der Datenbank. Danach gibt es das Komplement zurück. Der *substring*-Vergleich sorgt dafür, dass bei einer passenden Zeichenkette die Funktion abbricht. Die Jaccard-Distanz mit Hilfe des *substring*-Vergleichs kann so einfache Rechtschreibfehler korrigieren oder bei einer Angabe einer Teilbezeichnung den kompletten Namen in der Datenbank auffinden.

#### 4.4.4 Annotation

Das Paket *Annotation* annotiert eine vom Parser gesendete Frage. Dafür sind zwei Möglichkeiten implementiert.

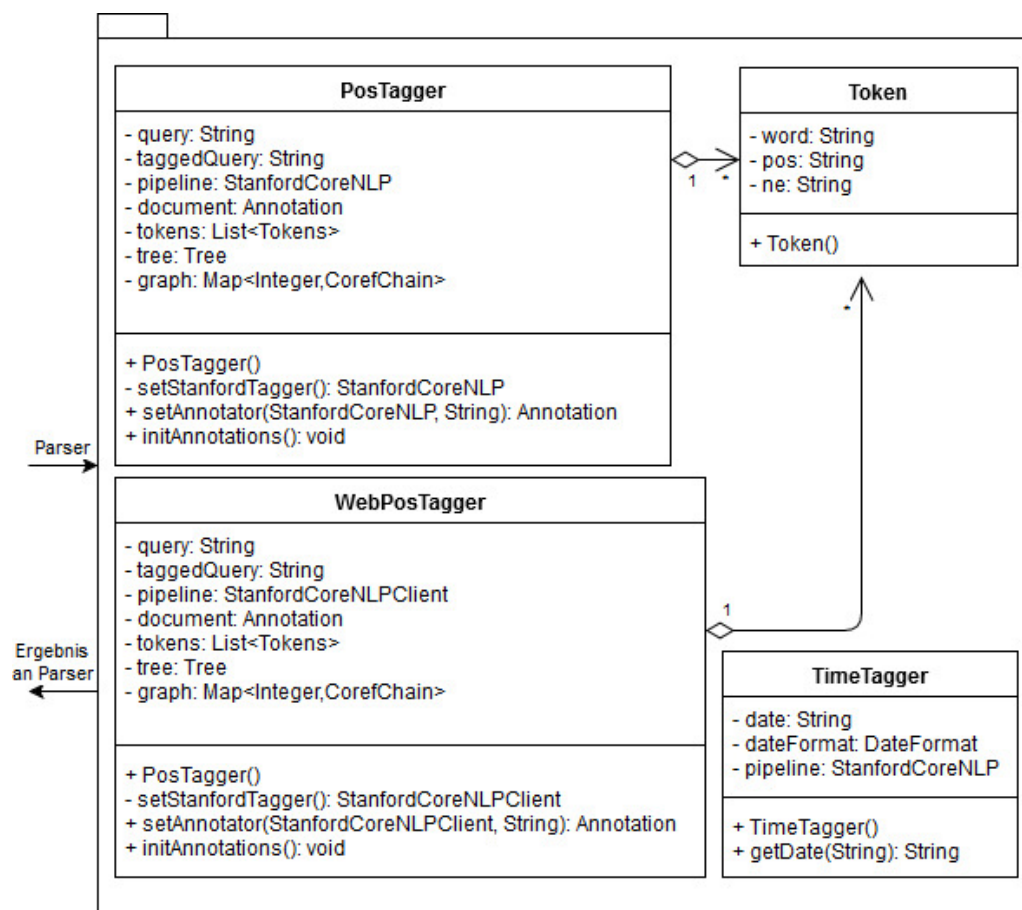


Abbildung 4.8: UML-Diagramm des Pakets: Annotation

Das System kann auf die Klasse *PosTagger* zurückgreifen. Die Funktionalitäten des *Stanford coreNLP* sind im System hinterlegt. Sie müssen bei der ersten Ausführung zu-

nächst geladen werden. Alternativ kann das QAS die Klasse *WebPosTagger* aufrufen. *WebPosTagger* stellt dann eine Anfrage an den Webserver 139.18.2.39:9000, ein Service der DICE-Gruppe (<http://cs.uni-paderborn.de/ds/>). Dort ist das *Stanford coreNLP* geladen und bereit zur Ausführung. *Semantic Chess* kann dadurch Zeit sparen, vor allem beim ersten Aufruf.

Nachdem die Klassen das *Stanford coreNLP* initialisiert haben (*setStanfordTagger()* und *setAnnotator()*), analysieren sie für eine Frage jedes Wort und sammeln die Ergebnisse in der Klasse *Token*. In ihr speichert das System das Wort, das POS-Tag und NE-Tag (*named entity*). Für das Beispiel *“Did Tigran Petrosian lose a game in any Olympiad?”* baut das QAS folgende *Token*-Liste auf (vgl. Tab. 4.4).

<b>Word</b>	Did	Tigran	Petrosian	lose	a	game	in	any	Olympiad	?
<b>POS</b>	VBD	NNP	NNP	VB	DT	NN	IN	DT	NNP	.
<b>NE</b>	O	PERSON	PERSON	O	O	O	O	O	MISC	O

Tabelle 4.4: Beispiel einer *Token*-Liste

Für den Parser ist diese Liste Grundlage der weiteren Analyse.

Die Klasse *TimeTagger* wandelt Datumswerte in ein Format um, das auch von PGN-Dateien genutzt wird (z. B. “20. Juni 2014” in “2014.06.20”). Der genutzte Webserver 139.18.2.39:9000 bietet zur Zeit nicht die Möglichkeit diese Funktion des *Stanford coreNLP* zu nutzen. Sie muss daher beim ersten Aufruf einer Frage mit Datumsangabe zunächst geladen werden.

#### 4.4.5 Parser

Das Paket *Parser* beinhaltet die Logik, um eine Frage eines Nutzers in ein SPARQL Query umzuwandeln. Es besteht aus den Hauptklassen *Allocator* und *Parser*, sowie einer Klasse für die Templates (*Sequences*) und für das Schachvokabular (*ChessVocabulary*) (vgl. Abb. 4.9).

##### Klasse: Parser

Der *Parser* analysiert die Frage, kontrolliert die Zuordnung der Wörter zu den passenden Entitäten, erzeugt für die Entitäten Tripel und erstellt einen Sequenzcode für die Auswahl eines Templates. Die meisten Funktionalitäten des *Parsers* befinden sich im Paket *Utils*.

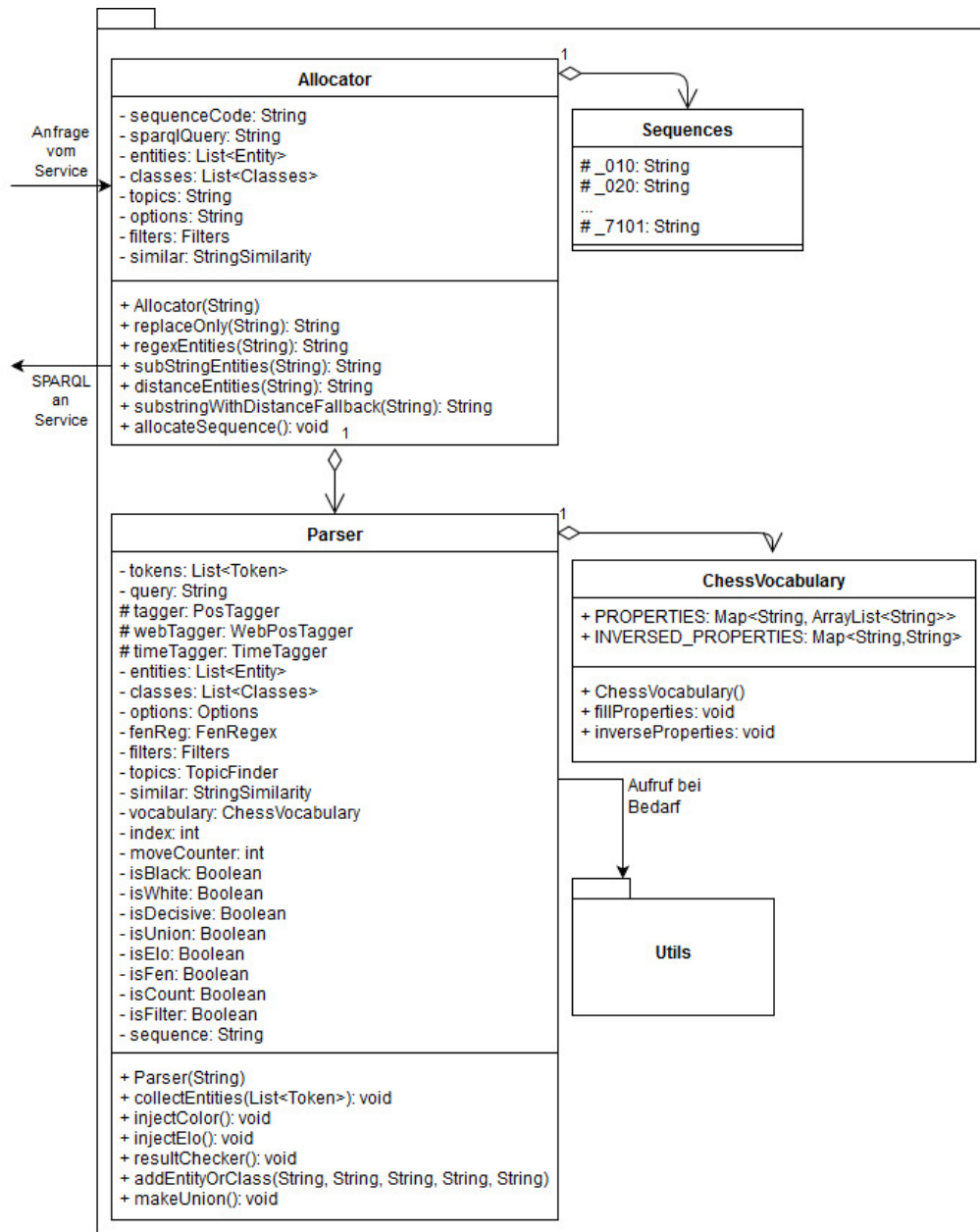


Abbildung 4.9: UML-Diagramm des Pakets: Parser

Mit den Ergebnissen des *Stanford coreNLP*, wie z. B. aus Tab. 4.4, startet der *Parser* die Klasse *CustomNer* (Paket *Utils*), die die Frage auf schachspezifische Wörter prüft. Die Klasse enthält vier Funktionen. Die erste (*stemming()*) ist für die Stammformreduktion zuständig, die mit Hilfe eines Pakets (`edu.stanford.nlp.process.Morphology`) der Stanford Universität durchgeführt wird. Dadurch müssen nicht alle möglichen Varianten eines Wortes in das Schachvokabular hinterlegt werden, sondern nur die Stammform. Eine weitere Funktion (*checkChessVocabulary()*) überprüft, ob ein Eintrag für das zu kontrollierende Wort im Schachvokabular existiert und markiert gegebenenfalls das Wort mit der hinterlegten Entität. Mit Hilfe von regulären Ausdrücken überprüft die Funktion zudem, ob das Wort einem ECO-Code (z. B. "E99") oder einem Zug (z. B. "Qxf3") ent-

spricht.

Die Funktion *checkElo()* nimmt die Trennung von Jahres- und Elozahlen (Wertzahlen für Schachspieler) vor. Der Stanford POS Tagger klassifiziert Zahlen zwischen 1000 und 3000 als Jahr, allerdings befinden sich die Elozahlen ebenfalls in diesem Bereich. Deswegen überprüft die Klasse die Umgebung, in dem sich die Zahl befindet. Gibt der Nutzer einen Hinweis zur Interpretation der Zahl (Schlüsselwörter, wie *ELO* oder *rating* in der Nähe der Zahl), wandelt die Klasse die ursprüngliche Entität von einer Jahres- zu einer Elozahl um. In der Anfrage *“Show me players with an ELO rating over 2800.”* ist das Schlüsselwort *“rating”* entscheidend, um die Klassifizierung von *“2800”* zu ändern. Es wird zunächst im Schachvokabular gefunden und als Entität markiert. Wenn das System die Zahl *“2800”* analysiert, bemerkt es, das *“rating”* als vorangehende Entität existiert und ändert die Klassifizierung von *“2800”* von einer Jahres- in eine Wertzahl um.

Ein ähnliches Prinzip setzt die Klasse *CustomNer* mit Hilfe der Funktion *checkOpening()* beim Erkennen von Eröffnungsnamen um. Die Klasse geht davon aus, dass eine Eröffnung immer mit einem Schlüsselwort in Verbindung steht, wie z. B. *“opening”*, *“system”*, *“gambit”*, *“defense”* oder *“indian”*. Beispiele hierfür sind *“Queen’s Gambit Declined”* oder *“French Defense”*. Zunächst erkennt der *Stanford POS Tagger*, diese Beispiele als Entitäten und ordnet ihnen das Label *MISC* oder *ORGANIZATION* zu. Aufgrund der Schlüsselwörter *“gambit”* und *“defense”* erkennt das System, dass es sich bei diesen beiden Entitäten um Eröffnungen handelt.

*CustomNer* erweitert das Beispiel aus der Tab. 4.4 wie folgt:

Word	Did	Tigran	Petrosian	lose	a	game	in	any	Olympiad	?
POS	VBD	NNP	NNP	VB	DT	NN	IN	DT	NNP	.
NE	O	PERSON	PERSON	0-1	O	game	O	O	event	O

Tabelle 4.5: Beispiel einer erweiterten *Token*-Liste

Die Entität des Schlüsselworts *“Olympiad”* überschreibt *CustomNer* von *MISC* zu *“event”*, da es im Schachvokabular hinterlegt ist. *“Olympiad”* verweist von dort auf die Entität *“event”*.

Im Anschluss sammelt der *Parser* benannte Entitäten in der Klasse *Entity*, wie z. B. *“Tigran Petrosian”* und unbennante Entitäten in der Klasse *Classes*, wie z. B. *“player”* (vgl. auch 4.1, Abschnitt *“Prozess: Schachentitäten”*). Das geschieht mit Hilfe einer *switch*-Anweisung, in der jede Entität aus dem Schachvokabular und ihre Rolle als Objekt (bzw. Subjekt), sowie das dazugehörige Prädikat und Subjekt (bzw. Objekt) fest hinterlegt ist.

Durch die unterschiedlichen Fälle (engl., *case*) in der *switch*-Anweisung können Ausnahmen eingearbeitet werden, die z. B. die Betrachtung der Umgebung benötigen. Ein Beispiel ist *"Show me 2 games from round 2 of the Tata Steel 2016"*. Hier betrachtet der Parser im *case* *"NUMBER"* der *switch*-Anweisung jeweils die erste und die letzte Zwei. Der Parser erkennt, dass in der Nähe der ersten Zwei *"games"* steht. Die Ausnahme regelt dadurch die Anpassung des *LIMIT* im SPARQL Query. Die nächste Zwei bezieht sich auf die Metainformation *"Runde"* einer PGN-Datei, da vor der Zwei die Entität *"round"* aus dem Schachvokabular entdeckt wird. Der Parser muss hier die Zwei als benannte Entität erkennen und ihr diese Metainformation zuordnen.

Während der *Parser* diesen Prozess für jedes Wort durchführt, setzt er *Flags*, die durch verschiedene Entitäten aus dem Schachvokabular und der Nutzerfrage ausgelöst werden und die später das SPARQL Query ergänzen. Dazu zählen *Flags* für *FILTER*, *LIMIT*, *OFFSET*, *UNION*, *GROUP BY*, *ORDER BY* und für die *SELECT*-Klausel *COUNT*, *MAX*, *MIN* oder *AVG* (vgl. 4.1, Abschnitt "Prozess: Schachentitäten"). Filter werden vom *Parser* nur an bestimmten Stellen verwendet, z. B. bei Komparativen, wie z. B. "größer als 2700". Der *Parser* sammelt sie mit Hilfe der Klasse *Filters* aus *Utils*. Diese Klasse enthält Funktionen, die Zeichenketten für verschiedene Filter erzeugen (z. B. *addGreaterThan()*, *addRegex()*). Optionen und Aggregationen werden mit *Options* (ebenfalls *Utils*) gesammelt. Wie bei *Filters* gibt es Funktionen, die bestimmte Zeichenketten erstellen, z. B. *setLimitStr()* oder *setOrderStr()*. Limitierung (*LIMIT*) werden durch Kardinalzahlen und Verschiebungen (*OFFSET*) durch Ordinalzahlen ausgelöst. Ordnungen (*ORDER BY*) entstehen u. a. durch Zählungen (*COUNT*), die durch ihr Auftreten eine Gruppierung (*GROUP BY*) auslösen.

Eine *UNION*-Klausel wird im Template genutzt, wenn eine Elozahl vorkommt oder bei mehreren Spielern nicht bestimmt ist, wer die weißen oder schwarzen Steine führt. Dazu überprüft der *Parser*, ob die Frage eine Farbe enthält, also die entsprechende Entität *"white"* oder *"black"* auftaucht. Ist das der Fall müssen gegebenenfalls die Eigenschaften der Entitäten angepasst werden, die sich auf Personen und Elozahlen beziehen, bspw. für *"Show me winning games by Magnus Carlsen with the black pieces and an Elo of 2870"*. Standardmäßig erkennt der Parser im Wort *"winning"* die Entität *"1-0"* (vgl. Listing 4.6). Da aber Schwarz gewinnen soll, muss das Ergebnis *"0-1"* lauten. Der Parser muss die Entität ändern, zusätzlich "Magnus Carlsen" die Eigenschaft "Schwarzspieler" zuordnen und für die Elo notieren, dass sie dem Schwarzspieler gehören soll (vgl. Listing 4.7).

```

1 SELECT DISTINCT *
2 WHERE {
3   ?game prop:white|prop:black 'Magnus Carlsen' .
4   ?game prop:result '1-0' .
5   ?game prop:whiteelo|prop:blackelo '2870' .
6 }

```

Listing 4.6: SPARQL ohne Beachtung der Flags für Farben und Elo

```

1 SELECT DISTINCT *
2 WHERE {
3   ?game prop:black 'Magnus Carlsen' .
4   ?game prop:result '0-1' .
5   ?game prop:blackelo '2870' .
6 }

```

Listing 4.7: SPARQL mit Beachtung der Flags für Farben und Elo

Wenn verschiedene Personen in der Frage vorkommen, ermittelt der *Parser* über eine Abstandsfunktion, wem eine Farbe bzw. Elozahl zuzuordnen ist. Die Abstandsfunktion zählt den Wortabstand der Farbe zur Person.

Existiert keine Farbe, muss eine *UNION*-Klausel im Template vorhanden sein, wenn die Frage eine Gewinn- oder Verlustpartie verlangt. Für das Beispiel *“Show me winning games by Magnus Carlsen with an Elo of 2870.”* setzt der *Parser* bei der Analyse eine *Flag* für eine Vereinigung, mit der der Sequenzcode angepasst wird und mit dem dann ermittelten Template folgende Query entstehen kann:

```

1 SELECT DISTINCT *
2 WHERE {
3   {?game prop:white 'Magnus Carlsen' .
4     ?game prop:result '1-0' .
5     ?game prop:whiteelo '2870' .}
6   UNION
7   {?game prop:black 'Magnus Carlsen' .
8     ?game prop:result '0-1' .
9     ?game prop:blackelo '2870' .}
10 }

```

Listing 4.8: SPARQL mit Vereinigung

In jeder Untermenge müssen die Farbe und die Elozahl eindeutig einer Person zugeordnet sein. Durch die Abstandsfunktion erkennt das System, dass im Beispiel die Elozahl “Magnus Carlsen” zuzuordnen ist. Da der Nutzer Gewinnpartien von “Magnus Carlsen” sehen möchte, muss das System in der ersten Untermenge dem Spieler die weißen Figuren

und ihm außerdem die Elozahl des Weißspielers mit dem entsprechenden Resultat ("1-0") zuteilen. In der nächsten Untermenge wiederholt das System die Zuordnung für Schwarz.

Falls der Nutzer nach Positionen fragt, müssen die zu jedem Zug generierten FEN-Zeichenketten mit Hilfe eines *regex*-Filters im SPARQL Query überprüft werden. Ein positionsbezogenes Beispiel lautet "Show me games with rook against rook and pawn". Der *Parser* interpretiert die Figuren vor "against" (oder einem Synonym) als weiße Steine, die folgenden als schwarze. Der *Parser* muss also einen regulären Ausdruck bilden mit einem weißen Turm, sowie einem schwarzen Turm und schwarzen Bauern. Weiße Figuren werden im FEN-String mit großen Buchstaben, schwarze Figuren mit kleinen dargestellt.

```
1 ^ (?=(.*R){1}) (?!(.*R){2}) (?=(.*r){1}) (?!(.*r){2}) (?=(.*p){1}) (?!(.*p){2}) ((?![QqBbNnP])).*
```

Listing 4.9: *regex* für Turm gegen Turm und Bauern

Der *Parser* zählt für jede Figur die Häufigkeit im FEN-String. Für den weißen Turm legt er dafür `(?=(.*R){1})(?!(.*R){2})` an. Damit taucht nur ein "R" im String auf. Das gleiche wiederholt die Klasse mit den schwarzen Figuren. Anschließend fügt er die Figuren an den regulären Ausdruck an, die nicht vorkommen sollen `((?![QqBbNnP]))`. Ist vom Nutzer keine Stellung gefragt, überspringt der *Parser* diesen Prozess.

Im letzten Schritt sammelt der *Parser* mit Hilfe der Klasse *TopicFinder* aus dem Paket *Utils*, welche Entitäten in der *SELECT*-Klausel vorkommen sollen. Dazu bevorzugt er vor allem die Entitäten, die möglichst weit vorne in der Frage stehen. Beispielsweise schreibt der *Parser* für die Frage "Who won against Magnus Carlsen in 2017?" die Entität "Person", wegen dem Fragepronomen *who*, in die *SELECT*-Klausel.

Anhand der gesammelten Entitäten legt der *Parser* eine Sequenz an. Diesen Code übergibt es dem *Allocator*, zusammen mit den Entitäten und Informationen für die *SELECT*-Klausel, Filter und Optionen.

### Klasse: Allocator

Der Ausgangspunkt des Pakets *Parser* ist die Klasse *Allocator*. Sie startet den Prozess, wählt auf Grundlage der Analyse der Klasse *Parser* ein SPARQL Template aus und ersetzt die Platzhalter im Template. Der *Allocator* bestimmt außerdem, in welcher Form die Klasse *StringSimilarity* die benannten Entitäten mit den Informationen aus der Datenbank abgleichen soll.



Das Template erhält der *Allocator* aus der Klasse *Sequences*. Mit der Sequenz aus dem *Parser* kann das Programm das nötige Template zuordnen. Danach bereitet der *Allocator* die Vorlage als ausführbare SPARQL Query auf. Im ersten Schritt ersetzt die Klasse die Platzhalter. Dazu stehen vier verschiedene Funktionen zur Verfügung.

Die Funktion *replaceOnly()* ersetzt lediglich die Platzhalter mit den Entitäten ohne vorher *StringSimilarity* aufzurufen.

Mit *regexEntities* nutzt der *Allocator* den *regex*-Filter von SPARQL. Jede benannte Entität erhält so im Query einen Filter. Für das Beispiel *“Show me games by Lasker”* erstellt der *Allocator* folgende SPARQL Query:

```
1 SELECT DISTINCT *
2 WHERE {
3   ?game prop:white|prop:black ?person .
4   FILTER regex(?person, 'Lasker', 'i')
5 }
```

Listing 4.10: *regexEntities()*

Dadurch sollen alle Informationen in der Datenbank gefunden werden, die die Zeichenkette enthalten.

Einen ähnlichen Ansatz wählt die Funktion *subStringEntities()*. Sie befragt allerdings zuerst die Datenbank nach Informationen, die die Zeichenkette enthalten. Dazu nutzt sie *StringSimilarity*. Mit Hilfe einer *VALUES*-Klausel listet der *Allocator* alle gefunden Informationen in der Query auf. Gibt es mehrere Entitäten permutiert die Klasse die Ergebnisse, um alle Kombinationen in der Query abbilden zu können. Im Beispiel *“Show me games by Lasker against Steinitz”* findet *StringSimilarity* für “Lasker” die Personen “Emanuel Lasker” und “Edward Lasker”, für “Steinitz” die Personen “Wilhelm Steinitz” und “V Steinitz”. Durch die Permutation entsteht eine Query, die prüft, ob “Wilhelm Steinitz” gegen “Emanuel Lasker”, “Wilhelm Steinitz” gegen “Edward Lasker”, “V Steinitz” gegen “Emanuel Lasker” oder “V Steinitz” gegen “Edward Lasker” gespielt hat (vgl. Listing 4.11).

Die Funktion *distanceEntities()* befragt ebenfalls *StringSimilarity*. Hier liefert die Klasse aber nur das von der Jaccard-Distanz ermittelte passendste Ergebnis zurück. Dieses ersetzt der *Allocator* mit dem entsprechenden Platzhalter.

Um die Vorteile von *subStringEntities()* und *distanceEntities()* zu nutzen, kombiniert die Funktion *substringWithDistanceFallback()* diese beiden. Zunächst sammelt *StringSimilarity* alle Informationen mit Hilfe von *subStringEntities()*. Liefert diese Funktion keine Ergebnisse, startet die Klasse die Distanzfunktion für eine neue Suche.

```

1 SELECT DISTINCT *
2 WHERE {
3   VALUES (?value1 ?value2){
4     ('Wilhelm Steinitz' 'Emanuel Lasker')
5     ('Wilhelm Steinitz' 'Edward Lasker')
6     ('V Steinitz' 'Emanuel Lasker')
7     ('V Steinitz' 'Edward Lasker')
8   }
9   ?game prop:white|prop:black ?values1 .
10  ?game prop:white|prop:black ?values2 .
11 }

```

Listing 4.11: subStringEntities()

Im Anschluss fügt der *Allocator* Variablen in die *SELECT*-Klausel ein, sowie Filter, Optionen und Aggregationen in die SPARQL Query. Eine Anfrage mit mehreren eingesetzten Möglichkeiten kann dann wie folgt aussehen:

```

1
2 SELECT DISTINCT ?date (COUNT(?date) AS ?nr)
3 WHERE {
4   VALUES (?value1 ?value2 ) {('Wilhelm Steinitz' 'Wilhelm Steinitz')}{
5     ?game prop:white ?value1 .
6     ?game prop:result '1-0' .
7     ?game prop:date ?date .
8     ?game prop:white ?person .
9     ?game prop:whiteelo ?elo .
10  }
11 UNION {
12    ?game prop:black ?value2 .
13    ?game prop:result '0-1' .
14    ?game prop:date ?date .
15    ?game prop:black ?person .
16    ?game prop:blackelo ?elo .
17  }
18 FILTER (?elo > 2500)
19 }
20 GROUP BY ?date ORDER BY DESC(?nr) LIMIT 2000 OFFSET 0

```

Listing 4.12: SPARQL Query für die Frage: “When did Wilhelm Steinitz win most often with an ELO over 2500.”

Die fertige Anfrage sendet das Paket zurück an die *Service*-Klasse, die dann die Datenbank abruft.

#### 4.4.6 Converter

Der *Converter* basiert auf der Arbeit der Praktikumsgruppe der Universität Leipzig aus dem Jahr 2013 (<https://github.com/ChewieSC/swp13-sc>). Das Paket wurde zusätzlich bis 2015 von Felix Conrads unter <https://github.com/dice-group/cacadus> erweitert. Die Aufgabe des *Converters* besteht in der Konvertierung von PGN-Dateien und Informationen von Schacheröffnungen zu RDF. Die angelegten RDF-Kollektionen werden dann vom Paket *Database* in die Datenbank geladen.

Der gesamte Konvertierungsprozess besteht aus drei Phasen:

1. Umwandlung der Partien
2. Umwandlung der Eröffnungen
3. Zuordnung der Eröffnungen zu den Partien

Das Paket des *Converters* besteht aus zwei Klassen (*EcoToRdf* und *PgnToRdf*), sowie Hilfsklassen im Unterpaket *Utils*. *PgnToRdf* ist für die PGN-Dateien zuständig. Mit den bereits bestehenden Klassen der Praktikumsgruppe aus dem Paket *Utils*, teilt *PgnToRdf* die PGN- in gleich große RDF-Dateien ein. Das geschieht gemäß der vorgestellten Ontologie aus Kapitel 2.7.2. *EcoToRdf* wandelt die Eröffnungen in RDF-Daten um. Es benötigt dafür eine Textdatei mit einer Liste von Schacheröffnungen (*chessopenings.txt*). Anschließend ordnet es den Partien in der Datenbank diese Eröffnungen mit Hilfe der Klasse *EcoLinker* (aus *Utils*) zu.

### 4.5 Datenbank

Die Datenbank speichert die RDF-Daten zu Schachpartien und -eröffnungen. Folgende Technologien werden verwendet:

- Docker
- Virtuoso

Docker ist eine Open-Source-Software. Die Docker-Container enthalten Anwendungen (z. B. Virtuoso) mit allen dafür nötigen Ressourcen. Container isolieren die Software, sodass sie in allen Umgebungen (Windows, Mac, Linux, etc.) gleich ausgeführt wird (vgl. [27]).

Virtuoso ist eine Software-Lösung u. a. für den Datenzugriff, die Visualisierung und das Datenbankmanagement. Dazu gehört ein *RDF Store* mit einer SPARQL-Schnittstelle, um auf die semantischen Daten zuzugreifen (vgl. [60]). Die Virtuoso-Instanz enthält sämtliche Schachpartien und -eröffnungen in RDF für *Semantic Chess*. Über eine Anfrage des Pakets *Database* mit Hilfe von Jena ARQ kann Virtuoso diese verarbeiten und anschließend das Result der SPARQL Query zurückliefern.

## 4.6 Lizenzen

*Semantic Chess* steht unter der GPL-3.0-Lizenz (*GNU General Public License v3.0*). Der Code ist frei verfügbar und kann von jedem genutzt werden. Bei den Abhängigkeiten des Systems wurde darauf geachtet, dass diese ebenfalls als Open Source zur Verfügung stehen. Die folgenden Tabellen sollen einen Überblick über die Lizenzen der verwendeten Software geben.

### Frontend

Software	Lizenz	Quelle
AngularJS	MIT License	[7]
Angular-Route.js	MIT License	[7]
Bootstrap 4	MIT license	[13]
FontAwesome	Font: SIL OFL 1.1 Code: MIT License	[33]
PGNViewer	Apache License 2.0	[47]

Tabelle 4.6: Frontend: Lizenzen der verwendeten Software

## Backend

Software	Lizenz	Quelle
commons-text	Apache License 2.0	[34]
Jena ARQ	Apache License 2.0	[44]
Spring Boot	Apache License 2.0	[54]
Stanford CoreNLP	GPL-3.0	[22]

Tabelle 4.7: Backend: Lizenzen der verwendeten Software

## Datenbank

Software	Lizenz	Quelle
Docker	Apache License 2.0	[26]
Virtuoso Open-Source Edition	GPL-2.0	[62]

Tabelle 4.8: Datenbank: Lizenzen der verwendeten Software

## 4.7 Urheberrecht

Neben diesen frei verfügbaren Technologien muss bei der Weiterentwicklung und -verwendung von *Semantic Chess* das Urheberrecht beachtet werden.

Schachpartien sind nach jetzigem Stand nicht vom Urheberrecht geschützt. Da ein Gerichtsurteil fehlt, hat der Deutsche Schachbund 1994 ein Gutachten mit mehreren Argumenten erstellt. Ein wichtiger Punkt ist die Teilnahme von zwei Spielern an einer Partie. Beide würden als Miturheber zählen. Da eine Miturheberschaft an einem Werk aber nur unter Zusammenarbeit und Verständigung entsteht, kann diese bei einer Schachpartie nicht geltend gemacht werden. Denn hier ist das Gegenteil der Fall: das Gewinnstreben und die Ideen des Gegners sollen verhindert werden (vgl. [9], S. 7f).

Allerdings kann sich laut Deutschem Schachbund "ein Schachspieler vor unberechtigter kommerzieller Ausbeutung seiner Partien gem. § 1 UWG schützen" ([9], S. 8).

Dagegen stehen die in PGN-Dateien geschriebenen Kommentare und Annotationen unter Schutz (vgl. UrhG § 2). Ihre Verwendung kann nur mit der Zustimmung des Autors erfolgen (vgl. UrhG § 12). Für Werke von Autoren, die vor 70 Jahren gestorben sind, erlischt das Urheberrecht (vgl. UrhG § 64). Um rechtlichen Bedenken aus dem Weg zu gehen, besitzt *Semantic Chess* daher zur Zeit keine Möglichkeit nach Kommentaren oder Annotationen zu fragen.

## Kapitel 5

# Evaluation und Benchmark

Um die Leistungsfähigkeit von *Semantic Chess* zu bewerten, wurde ein Benchmark erstellt. Damit kann das System mit anderen zukünftigen QAS im Bereich Schach verglichen werden. Dieses Kapitel erläutert zudem die Stärken und Schwächen von *Semantic Chess*. Den Schluss bildet eine Einschätzung, inwieweit das System die sieben Herausforderungen erfüllt, die das Kapitel "Stand der Technik" vorgestellt hat.

Der Benchmark dient als Vergleichsmaßstab zwischen verschiedenen QAS. Diese Arbeit hat sich an der *QALD challenge* orientiert (vgl. [50]). Als Trainingsdaten dienen 50 schachspezifische Fragen, die bei der Entwicklung von zukünftigen QAS in diesem Bereich unterstützen sollen. Die Testdaten umfassen zehn verschiedene Fragen, die nach der Erstellung des Systems zusammengetragen wurden. Sie stammen von Nutzern eines Schachforums (<https://www.chess.com/forum/view/scholastic-chess/chess-q-a>).

Zur Auswertung wurden jeweils *Precision*, *Recall* und *F-Measure*, sowie die entsprechenden Mikro- und Makrowerte berechnet. Da die Antwortmengen in ihrer Größe zwischen eins und 953 schwanken, sind bei der Auswertung eher die Mikrowerte zu bevorzugen. Die Makrowerte geben zwar einen schnellen Überblick über die Gesamtpformance des Systems, berücksichtigen aber nicht den Umstand, dass das System bei einigen Fragen besonders viele richtige bzw. falsche Antworten ausliefert oder viele richtige bzw. falsche nicht findet.

## 5.1 Trainingsdaten

Die 50 Trainingsdaten sollen viele verschiedene Kombinationen aus den Informationen von PGN-Dateien abdecken. So soll z. B. die Frage *“Show me games by Wilhelm Steinitz.”* einen Spieler (in der PGN-Datei als *white* oder *black*) und die Frage *“In which game was the latest castling?”* die Züge betrachten.

Die vorgestellten Funktionen *subStringEntities()*, *distanceEntities()*, *regexEntities()* und *substringWithDistanceFallback()* (vgl. Kap. 4.4.5, Abschnitt “Klasse: Allocator”) wurden mit diesen Fragen entwickelt. Die Auswertung berechnet für alle Fragen den Makro- bzw. Mikro-*Recall*, *-Precision* und *-F-Measure* (vgl. Kap. 2.6).

Die Datenbank enthielt während der Entwicklung 6464 Partien aus dem 18. und 19. Jahrhundert. Partien aus diesem Zeitraum enthalten teilweise unvollständige Daten. Aufgrund des Alters der Partien sind das bspw. fehlende Spieler, unvollständige Namen und Datums-  
werte, sowie falsche Züge. Das System wurde mit diesen Ungenauigkeiten als Grundlage entwickelt, um Wege zu implementieren auch Partien mit diesen Informationen zu finden.

### **regexEntities()**

Die Funktion *regexEntities()* liefert für die 50 Fragen 31 Antworten mit einem Makro-*F-Measure* von 59,93% und einem Mikrowert von 60,05% (vgl. Anhang B.4). Da die Funktion vor allem Fragen mit vielen Antworten nicht löst, ist die Mikro-*Precision* von allen Vergleichswerten am höchsten. Dafür besitzt der Mikro-*Recall* nur einen Wert von 43,96%. Die Funktion beantwortet, im Vergleich zu den anderen Funktionen, die wenigsten Fragen.

Der Grund ist die statische Platzierung der Filter am Ende der Templates. Bei Fragen, die eine Aufteilung der gefundenen Filter auf die Untermengen der Vereinigung erfordert, zeigt die Funktion Schwächen. Das tritt bei den Fragen auf, die eine Verlust- oder Gewinnpartie erwarten, wie z. B. die Daten mit den Indizes 2 und 31. Die Filter stehen in den Vorlagen immer am Ende. Eine Aufteilung der Filter führt das System nicht durch. Ein weiterer Nachteil der *regex*-Filter ist die Unfähigkeit Rechtschreibfehler oder alternative Schreibweisen (“Caro-Kann” statt “Caro Kann”) zu erkennen, wie in den Fragen mit den Indizes 4 und 47 zu erkennen ist.

Ein Vorteil der Funktion ist, dass sie die Datenbank erst bei der Ausführung der SPARQL

Query befragt und nicht im Vorfeld die gefundenen Entitäten abgleicht. Dadurch findet seltener eine Kommunikation zwischen System und Datenbank statt, was bei der Beantwortung der Frage etwas Zeit sparen kann.

### **distanceEntities()**

Die Funktion *distanceEntities()* beantwortet 39 Fragen mit einem Makro-*F-Measure* von 70,2% und einem leicht höheren Mikro-*F-Measure* von 76,01% (vgl. Anhang B.2).

Der größte Nachteil der Funktion ist, dass sie nur die ähnlichste Information nach dem Abgleich mit der Datenbank in den jeweiligen Platzhalter des Templates schreibt. Das tritt z. B. bei der Frage 19 auf: *“What openings were played at the World Championship 1894?”*. Das System erkennt *“World Championship”* als Entität. Durch den Abgleich findet das System die identische Information *“World Championship”*. Diese bezieht sich allerdings auf eine inoffizielle Weltmeisterschaft einige Jahre vor 1894. Die eigentliche Weltmeisterschaft von 1894 wird in den PGN-Dateien als *“1. World Championship”* betitelt. Dadurch ersetzt das System den entsprechenden Platzhalter mit der falschen Entität und liefert keine Antworten zurück, da 1894 keine Partien mit der exakten Event-Bezeichnung *“World Championship”* existieren.

Vorteilhaft an der Funktion ist die Möglichkeit Rechtschreibfehler zu umgehen oder alternative Schreibweisen zu finden. Da die Funktion dadurch ein Ergebnis (die Ähnlichste zur gefundenen Entität) zurückliefert, können die Platzhalter auch immer ersetzt werden. Sie bleiben nicht leer.

### **subStringEntities()**

Die Funktion *subStringEntities()* findet für 40 Fragen Antworten und weist einen Makro-*F-Measure* von 76,33% und einen Mikrowert von 65,75% auf (vgl. Anhang B.3).

*subStringEntities()* besitzt nicht die Möglichkeit Rechtschreibfehler zu ermitteln. Es erstellt eine Liste mit allen Informationen aus der Datenbank, die die exakte Zeichenkette der gefundenen Entität enthalten. Dadurch umgeht die Funktion den Nachteil von *distanceEntities()* nur die ähnlichste Information zu verwenden. Sie listet alle passenden Informationen in einer *VALUES*-Klausel auf. Für das Beispiel *“What openings were played at the World Championship 1894?”* sind dann die verschiedenen Event-Bezeichnungen vor-



handen, die “World Championship” enthalten (vgl. Listing 5.1). Das System kann damit eine Antwort finden.

```
1 SELECT DISTINCT ?opening
2 WHERE {
3     VALUES (?value1 ) {
4         ('1. World Championship')('2. World Championship')
5         ('3. World Championship')('4. World Championship')
6         ('World Championship')('5. World Championship')
7         ('6. World Championship')
8     }
9     ?game prop:event ?value1 .
10    ?contEco cont:openingName ?opening .
11    ?game cont:eco ?contEco .
12    ?game prop:date ?date .
13    FILTER ( regex( ?date , '1894' ))
14 }
15 LIMIT 2000 OFFSET 0
```

Listing 5.1: Beispiel VALUES-Klausel: “World Championship”

Der Nachteil an diesem Verfahren ist allerdings, dass die *VALUES*-Klauseln umfangreich werden können. Vor allem bei Eröffnungsnamen tritt dieser Fall auf. Fragt der Nutzer nach der “Caro-Kann-Eröffnung” listet das System die über 90 gespeicherten Varianten dieser Eröffnung auf. Ein weiterer Schwachpunkt ist, dass durch *VALUES*-Klauseln zu viele und nicht nötige Antworten mit ausgegeben werden. Ein Beispiel hierfür ist Frage 39. Der Nutzer wünscht sich Partien zwischen Steinitz und Lasker und hat höchstwahrscheinlich die Intention, Spiele zwischen dem ersten (Steinitz) und zweiten (Lasker) Weltmeister zu sehen. Da es aber Spieler gibt, die die gleichen Nachnamen besitzen, listet das System auch Spiele zwischen diesen Personen auf.

### **substringWithDistanceFallback()**

Um die Nachteile der beiden vorangehenden Funktionen etwas zu neutralisieren, wurden sie kombiniert. Die Funktion *substringWithDistanceFallback()* kann somit 44 von 50 Fragen der Trainingsdaten beantworten und erreicht einen Makro-*F-Measure* von 83,47% und einen Mikro-*F-Measure* von 80,17% (vgl. Anhang B.1).

Zunächst versucht das System mit *subStringEntities()* die Informationen zu sammeln. Findet das System in der Datenbank nichts, durchläuft es den Prozess ein zweites Mal

mit der Funktion *distanceEntities()*.

Die Funktion ist somit am zuverlässigsten und dient zur Beantwortung der Testfragen (vgl. Kap. 5.2).

## Unbeantwortete Fragen und ungenaue Antworten

Sechs Fragen bleiben nach der Implementierung für das System nicht zu beantworten. Dazu zählen die Fragen mit den Indizes 17, 23, 29, 39, 40 und 48 (vgl. Anhang B.1).

Die Frage 29: *"Show me any games with three pawns on the 2nd rank."* und *"Games with doubling rooks on the 7th rank."* verlangen Antworten, die die zweite bzw. siebte Reihe (*"rank"*) mit einfließen lassen. Das System erstellt bei positionsbezogenen Fragen einen *regex*-Filter, wie z. B. "Turm gegen Läufer". Für Fragen, die sich auf Reihen, Linien und Diagonalen beziehen, ist im System keine Funktion hinterlegt, die diese Informationen in einen Filter umwandeln. Für Reihen ist der Aufwand überschaubar, da der reguläre Ausdruck bspw. die Schrägstriche "/" in der FEN-Zeichenkette abzählen kann ("/" trennen die Reihen voneinander). Innerhalb der Schrägstriche befinden sich ein bis acht Zeichen. Dieser Umstand erschwert eine gezielte Abfrage der Linien. Die gleichen Schwierigkeiten treten bei den Diagonalen auf. Eine Abfrage von bestimmten Feldern ist derzeit auch nicht möglich, wie z. B. *"Show me games with white queen on a8 and white rook on h1."*. Die *regex*-Filter würden dadurch einen langen regulären Ausdruck enthalten. Bei einer Datenbank mit Millionen Partien mit jeweils ca. 50 bis 100 verschiedenen Positionen würde ein solcher String-Vergleich zu viel Zeit kosten.

Ein fehlender regulärer Ausdruck ist der Grund für eine ungenaue Beantwortung der Frage 32: *"Has Wilhelm Steinitz ever played a fianchetto?"*. Das System besitzt keine Funktion um den Begriff "Fianchetto" umzuwandeln. Es liefert alle Partien vom Spieler zurück. Ein regulärere Ausdruck müsste die Position eines Fianzettos abfragen. Dieses Stellungs-muster kann an mehreren Positionen des Brettes auftreten, sodass auch hier der reguläre Ausdruck lang wird.

Frage 17: *"Show me all the games of the tournament with the highest average rating."* und 39 *"What is the lifetime record between Wilhelm Steinitz and Emanuel Lasker?"* benötigen andere als die im System gespeicherten Templates. Die erste von beiden benötigt eine verschachtelte Query. Zunächst muss eine Anfrage für jedes Turnier den Durchschnitt der Elozahlen von allen Partien berechnen. Eine zweite Anfrage sammelt dann die Partien

von dem Turnier mit dem höchsten Eloschnitt. Frage 39 benötigt fünf Untermengen, die jeweils vereinigt werden müssen. Eine Untermenge zählt die Partien mit den Remis, zwei sind für die Niederlagen und die letzten beiden für die Siege der Spieler zuständig. Dadurch können die Punkte für jeden Spieler zusammengerechnet werden.

Bei den Fragen 23 (*"Who was the 5th world champion?"*) und 48 (*"What is played against the Italian Game"*) identifiziert das System die Entitäten nicht. Das QAS erkennt nicht, dass der fünfte Weltmeister (*"world champion"*) dem Event *"5. World Championship"* zugeordnet werden muss. Bei Frage 48 hat das System Probleme beim Erkennen der Entität *"Italian Game"*. Hier fehlt eines der vordefinierten Schlüsselwörter, wie z. B. *"opening"* oder *"system"*.

Neben Frage 32 besitzt auch Frage 44 einen *F-Measure* von unter 50%. Diese Frage enthält eine Zeitspanne, was zur Zeit nicht im System implementiert ist, um daraus einen Filter zu formen. Das System gibt nur die Antworten für die obere Grenze der Zeitspanne zurück.

## 5.2 Testdaten

Die Testdaten umfassen zehn Fragen, was einen Anteil von 20% zu den Trainingsdaten bedeutet. Die Fragen wurden von Nutzern des chess.com-Forums gestellt (vgl. [14]). Auf einen Forumseintrag antworteten zwei Nutzern mit jeweils zehn Fragen. Zum Erreichen der Testabdeckung von 20% wurden jeweils die ersten fünf der gestellten Fragen verwendet. Da die Datenbank Partien aus dem 18. und 19. Jahrhundert enthält, wurden die Fragen dementsprechend umformuliert (vgl. Tab. 5.1). Beispielweise ist in der Frage 9: *"What is the most played opening in the World Chess Championship overall beginning after year 1886?"* die Jahreszahl ausgetauscht. Bei den übrigen Fragen betrifft das meistens die verwendeten Spieler, Events und Eröffnungen, die in das 19. Jahrhundert übertragen wurden. Anschließend sollte das System die Fragen beantworten.

Von den zehn Fragen konnte *Semantic Chess* sechs beantworten und erreichte damit einen Makro-*F-Measure* von 57,39% und einen Mikro-*F-Measure* von 66,67%. Meistens verlangen die Nutzer nach Antworten, die nur eine Tabellenzeile benötigen, wie z. B. Ja-Oder-Nein-Fragen oder Fragen nach der Häufigkeit (*"how many"*) oder dem Häufigsten (*"most"*).

Die Fragen 2, 4, 9 und 10 konnte es nicht beantworten. Nummer 2: *"What was the lifetime*

Id	Question	Precision	Recall	F-Measure
1	Has the French Defense ever been played in a World Championship match?	100.00%	100.00%	100.00%
2	What was the lifetime score between Steinitz and Lasker?	0.00%	0.00%	0.00%
3	How many games did Wilhelm Steinitz win when employing the Steinitz Gambit?	100.00%	100.00%	100.00%
4	Who has won the DSB Kongress most often?	0.00%	0.00%	0.00%
5	Did Emanuel Lasker ever play Bertholt Lasker?	50.00%	100.00%	66.67%
6	How often won Adolf Anderssen in the English Opening?	100.00%	100.00%	100.00%
7	Games in the Lowenthal Variation with an ELO over 2300.	100.00%	100.00%	100.00%
8	What ist the most played opening by Adolf Anderssen?	100.00%	100.00%	100.00%
9	What is the most played opening in the World Chess Championship overall beginning after year 1886?	0.00%	0.00%	0.00%
10	Which World Chess Championship had the most games ever played?	0.00%	0.00%	0.00%
	<b>Number of answers</b>			<b>6</b>
	<b>Macro Precision, Recall, F-Measure</b>	<b>55.00%</b>	<b>60.00%</b>	<b>57.39%</b>
	<b>Micro Precision, Recall, F-Measure</b>	<b>75.00%</b>	<b>60.00%</b>	<b>66.67%</b>

Tabelle 5.1: Benchmark der Testdaten

*score between Steinitz and Lasker?*“ tauchte schon in den Trainingsdaten auf (vgl. Kap. 5.1). Das System benötigt zum Lösen der Frage mehrere Vereinigungen, um die Gewinn-, Verlust und Remispartien der Spieler in einer Tabellenzeile darstellen zu können. Dafür wurde allerdings kein Template hinterlegt. Alternativ könnte der Nutzer derzeit einzeln fragen, wie oft ein Spieler gegen den anderen gewonnen, verloren oder Unentschieden gespielt hat.

Frage 4 *“Who has won the DSB Kongress most often?”* benötigt den Einsatz von verschachtelten SPARQL Queries oder einer Vielzahl von Vereinigungen. Da der *DSB Kongress* häufiger stattfand, müssten von jedem Kongress die Spieler mit den meisten Punkten wiedergegeben werden. Für ein Einzeltournament ist die Wiedergabe des Siegers auch nicht gewährleistet, da wie bei der Frage 2 mehrere Vereinigungen nötig sind, um die erzielten Punkte jedes Spielers zu berechnen und anschließend den Sieger zu ermitteln.

Die Frage 9 verlangt eine Zeitspanne: *“What is the most played opening in the World Chess Championship overall beginning after year 1886?”*. Das System nutzt nur das Jahr 1886, findet in diesem allerdings keine Weltmeisterschaft und liefert kein Ergebnis zurück. Um die Antwort auf Frage 10 (*“Which World Chess Championship had the most games ever played?”*) zu erhalten, reicht es die Anzahl der Runden der Weltmeisterschaften zu zählen und absteigend zu sortieren. Das System erkennt die Verbindung zwischen *“most games”* und *“Anzahl der Runden”* nicht. Das System müsste bei einer Kombination aus der Frage nach einem Event und der Anzahl der Partien auf die Rundenzahl schließen.

Die Bearbeitungszeit schwankt je nach Größe der SPARQL Queries (vgl. Tab. 5.2).

Id	Question	Total Time in s	Parser Time in s
1	Has the French Defense ever been played in a World Championship match?	3,6	0,2
2	What was the lifetime score between Steinitz and Lasker?	-	-
3	How many games did Wilhelm Steinitz win when employing the Steinitz Gambit?	1,4	0,4
4	Who has won the DSB Kongress most often?	-	-
5	Did Emanuel Lasker ever play Bertholt Lasker?	0,4	0,2
6	How often won Adolf Anderssen in the English Opening?	11,8	0,2
7	Games in the Lowenthal Variation with an ELO over 2300.	1,0	0,3
8	What ist the most played opening by Adolf Anderssen?	0,4	0,2
9	What is the most played opening in the World Chess Championship overall beginning after year 1886?	-	-
10	Which World Chess Championship had the most games ever played?	-	-

Tabelle 5.2: Bearbeitungszeiten der Testfragen

Benötigt das System eine längere *VALUES*-Klausel, braucht die Abfrage der Datenbank mehr Zeit. Vor allem bei Frage 6 wird das deutlich. Das System sammelt die über 100 Varianten der Englischen Eröffnung. Die folgende Abfrage dauert dementsprechend länger. Ähnliches tritt bei der Frage 1 auf, wo das System eine längere Liste mit allen Events anlegt, die *“World Championship”* enthalten.

Der Prozess des Parsers dauert bei allen Fragen ungefähr gleich lang (ca. 0,2s bis 0,4s). Nur bei Fragen mit Schlüsselwörtern, die auf eine Zeit hindeuten, dauert die Analyse etwas länger. Hier muss das entsprechende Paket des *Stanford coreNLP* zunächst geladen werden, da der bereitgestellte Webserver diese Funktion nicht unterstützt. Als Beispiel kann hier Frage 3 herangezogen werden. Es enthält das Schlüsselwort *“when”*. Auch wenn das Wort in der Antwort keine Rolle spielt, lädt das System das Paket zunächst.

Eine Frage, die einen *regex*-Filter enthält, wie z. B. *“Show me games with rook against rook and pawn.”* benötigt für die Analyse der Frage ebenfalls nur ca. 0,2 Sekunden. Die Abfrage der Datenbank dauert hier aber bereits 17 Sekunden. Das zeigt die Notwendigkeit, bei einer Weiterentwicklung den *regex*-Filter zu umgehen und mit RDF-Daten zu ersetzen, wie z. B. mit Indizes für Stellungsmuster und Materialverhältnis.

## Zusammenfassung

Zusammenfassend zeigt der Benchmark, dass das System den Großteil der Fragen beantworten kann, aber noch Lücken enthält. Fragen, die sich nahe an den enthaltenen Informationen aus einer PGN-Datei orientieren, kann das System größtenteils beantworten. Für Fragen, die sich nicht sofort aus einer PGN erschließen, sondern sich erst nach weiteren Berechnungen oder verschachtelten Anfragen beantworten, fehlen dem System die entsprechenden Templates.

Das System kann die Fragen schnell beantworten. Je komplexer die SPARQL Queries und umfangreicher die *VALUES*-Klauseln werden, umso länger braucht das System, um eine Antwort zu formulieren.

## 5.3 Evaluierung der Herausforderungen im Schach-Question-Answering

Die im Kapitel "Stand der Technik" erläuterten Herausforderungen bei der Entwicklung eines QAS sollen hier abschließend für *Semantic Chess* betrachtet werden.

### Lexikalische Lücke

Die Entitätenerkennung vom *Stanford coreNLP* versucht Wörtern bestimmte Klassen zuzuordnen, wie z. B. Personennamen auf die Klasse "*PERSON*". Den gleichen Ansatz verfolgt *Semantic Chess* mit schachspezifischen Klassen, die sich auf die Metainformationen einer PGN-Datei konzentrieren. So wird für eine Eröffnung die Klasse "*OPENING*" ermittelt, wenn sie bestimmte Schlüsselwörter enthält. Im System ist dafür ein Vokabular enthalten, das versucht möglichst viele Wörter zu sammeln und sinnverwandte zu gruppieren. Durch Erfahrungswerte, wie den von Nutzern gestellten Fragen, kann diese Sammlung weiter wachsen und versuchen die lexikalische Lücke zu schließen. Der jetzige Stand weist noch Lücken auf, vor allem im Bereich von stellungsbezogenen Begriffen, wie Fianchetto oder Epaulettenmatt. Sie erfordern zusätzliche Informationen in den RDF-Daten, die noch fehlen, aber bei einer Weiterentwicklung des Systems in das Vokabular eingetragen und dort in Klassen gruppiert werden können. Zusätzlich müsste der Parser Funktionen enthalten, wie er diese Klassen interpretieren soll.

## Ambiguität

Die Herausforderung der Mehrdeutigkeit soll hauptsächlich vom Schachvokabular übernommen werden. Durch die Gruppierung der Wörter in Klassen erfasst das System schon einen Teil der Synonyme. Dadurch muss es nur noch die Klassen betrachten und nicht das einzelne Wort auf die RDF-Daten zuordnen. Für sinnverwandte Wörter und Wortgruppen, wie *“exchange”* und *“the exchange”* (vgl. Kap. 3.2), benötigt der Parser zusätzliche Informationen, die in das System eingearbeitet werden müssen.

## Mehrsprachigkeit

Das System *Semantic Chess* konzentriert sich auf das Englische. Fragen, in einer anderen Sprache versteht es nicht. Ein Weg diese Herausforderung zu bewältigen, ist eine Übersetzung des Schachvokabulars anzufertigen. Mit diesen Begriffen und Klassen kann der Parser den Wörtern einer Frage dann die RDF-Daten zuordnen und für das Template vorbereiten.

## Komplexe Anfragen

Mit den Templates kann das System einen Großteil der Fragen beantworten. Für eine komplette Abdeckung fehlen in der Sammlung weitere Vorlagen, die dann z. B. Fragen nach dem *“lifetime score”* und nach Zeitspannen beantworten oder verschachtelte Anfragen verwenden, wie z. B. für die Frage: *“Show me all the games of the tournament with the highest average rating”*. Da diese Fälle zum Teil durch den Benchmark identifiziert sind, kann bei einer Weiterentwicklung das System darauf hingewiesen werden, wann eine entsprechende Vorlage verwendet werden soll.

## Verteiltes Wissen

Schachpartien liegen nicht in verschiedenen Datenbanken im RDF-Format bereit. Dadurch kann das System nur auf die Daten zurückgreifen, die in der eigenen Datenbank vorhanden sind. Falls in Zukunft eine zusätzliche RDF-Sammlung existiert, müssen zusätzliche Klassen entwickelt werden, wie diese Daten abzufragen sind und wie sie der eigenen Ontologie zugeordnet werden können.

## Prozedurale, zeitliche und räumliche Fragen

Einige Lösungsansätze für diese Herausforderung bietet bereits das Kapitel 3.6. Prozedurale Fragen wurden in dem implementierten Ansatz von *Semantic Chess* nicht berücksichtigt, da die Antworten meistens über die in den PGN-Dateien enthaltenen Informationen hinausgehen.

Zeitliche und räumliche Fragen sind gut zu beantworten, wenn sie sich auf die Metainformationen (“*site*” und “*date*”) einer PGN-Datei beziehen. Wenn sie sich auf Stellungen beziehen, können sie nicht beantwortet werden, z. B. bei den Fragen “*Show me games where the queen attacks a rook and the king at the same time.*” oder “*On which move occurred the most adjacent pawns on 7th rank in the game between Lee and Shoosmith in 1904?*” .

## Templates

Die Templates wurden manuell im System angelegt. Durch Erfahrungswerte, die durch die Benutzung des QAS entstehen, können bei der Weiterentwicklung die Vorlagen verfeinert werden. Dadurch kann das System zukünftig weitere Fragen abdecken. Bei der Erstellung von neuen Templates ist darauf zu achten, dass der *Allocator* sie mit Hilfe des Sequenzcodes des *Parsers* eindeutig identifizieren kann.

Die Tabelle 5.3 fasst den Einfluss der Herausforderung auf das System *Semantic Chess* zusammen.



<b>Herausforderung</b>	<b>Einfluss auf Semantic Chess</b>	<b>Wie gelöst?</b>
Lexikalische Lücke	ja	manuelle Erstellung eines Schachvokabular
Ambiguität	ja	manuelle Erstellung eines Schachvokabular
Mehrsprachigkeit	nein	Englisch als Hauptsprache
Komplexe Anfragen	teilweise	durch Templates, aber keine komplette Abdeckung
Verteiltes Wissen	nein	Schachpartien liegen zur Zeit online nicht frei verfügbar im RDF-Format vor
Prozedurale, zeitliche und räumliche Fragen	teilweise	bezogen auf die Metainformationen der PGN-Dateien, nicht aber auf Stellungsmuster
Templates	ja	manuelle Erstellung von Templates

Tabelle 5.3: Einfluss der Herausforderungen auf Semantic Chess

## Kapitel 6

# Hinweise zur Weiterentwicklung

Front-, Backend und Datenbank enthalten jeweils Vor- und Nachteile. Die folgenden Abschnitte sollen diese Stärken und Schwächen nachzeichnen, um eine einfachere Weiterentwicklung zu ermöglichen.

### 6.1 Frontend

AngularJS ist im Frontend implementiert. Es bietet zwar die Möglichkeit eine *Single Page*-Applikation zu erstellen und (nach einer Einarbeitungsphase) das Projekt schnell weiterzuentwickeln, allerdings zeigt es bei größeren Datenmengen *Performance*-Probleme. Je mehr Antworten das System findet, umso mehr Elemente muss das Frontend bearbeiten. Eine Tabelle der Weboberfläche ist daher auf 2000 Zeilen begrenzt, um die Ladezeit nicht zu verlängern.

Bei einer Weiterentwicklung des Frontends könnte daher AngularJS gegen die Nachfolgerversion (derzeit Angular 5) oder gegen ein anderes *Framework* ausgetauscht werden, das größere Datenmengen schneller bearbeitet. Auch in AngularJS kann noch optimiert werden, indem bspw. die Tabelle auf mehrere Seiten aufgeteilt wird. Die Paginierung zeigt dann z. B. nur die ersten 10 Ergebnisse. Durch die weiteren kann der Nutzer anhand der Seitenzahlen navigieren.

Das Frontend soll eine einfache Bedienung aufweisen. Der Nutzer erhält nur die Möglichkeit mit der natürlichen Sprache eine Frage zu formulieren (bzw. SPARQL Queries für Nutzer mit dem nötigen Hintergrundwissen). Zusätzliche Filter werden vermieden. Durch die Darstellung der Antwort in einer Tabelle soll der Nutzer sofort entscheiden können, ob

das System die richtigen Ergebnisse anzeigt. Bei der Wiedergabe von Partien erschließt sich erst nach dem Laden des Spiels, ob das System die korrekte Antwort gegeben hat. Hier könnte z. B. eine Vorschau implementiert werden, damit der Nutzer einen Eindruck von der Partie erhält.

## 6.2 Backend

Die einzelnen Pakete des Backends sollen untereinander so unabhängig wie möglich voneinander agieren. Dadurch sollen sie in Zukunft leichter auszutauschen sein. Für die Klassen gilt ähnliches. Hier gibt es allerdings größere Abhängigkeiten, wie z. B. zwischen den Klassen *Allocator* und *Parser*.

Bei der Weiterentwicklung des Systems sollte außerdem beachtet werden, dass die Ontologie und die Konvertierung der Schach- in RDF-Daten aus dem Jahr 2013 stammt. Einige Abhängigkeiten aus der damaligen *pom.xml* der Konvertierung (Datei mit Informationen des Projekts und seinen Abhängigkeiten) sind veraltet. Beispielsweise ist das Artefakt von Jena ARQ von `com.hp.hp1.jena` zu `org.apache.jena` umgezogen. Die Notwendigkeit einer Aktualisierung muss daher beobachtet werden.

Eine Aktualisierung und Erweiterung benötigt auch das Schachvokabular. Nicht nur um kommende Entwicklungen bei Schachbegriffen und neuen oder veränderten Bezeichnungen der Eröffnungen entgegenzukommen, sondern auch um das System um noch nicht implementierte Begriffe (z. B. Fianchetto) zu erweitern (vgl. Kap. 5.2).

Außerdem benötigen die Schachpositionen und deren Merkmale einen Index oder zusätzliche Informationen in den RDF-Daten, um sie schneller durchsuchen zu können. Bereits bei den 6464 Partien der Testdatenbank benötigt dieser Vergleich mehrere Sekunden, um alle Stellungen zu durchlaufen.

Für die zukünftige Weiterentwicklung muss das System zudem für Fragen sensibilisiert werden, die Zeitspannen erfordern. Es muss Filter anlegen können, die Zeitspannen umsetzen und Schlüsselwörter wie *“between”*, *“after”* und *“before”* erkennen.

## 6.3 Datenbank

Virtuoso bietet eine effiziente Möglichkeit Millionen von Tripel zu speichern und zu organisieren. Allerdings stürzt es häufig bei der Zuordnung von Schacheröffnungen zu Partien (vgl. Kap. 4.4.6) an zufälligen Punkten ab. Während der Zuordnung befragt das System die Datenbank, welche Partien mit der Zugreihenfolge in den Eröffnungen übereinstimmen. Dieser Fehler betrifft deshalb ausschließlich den Verwalter des Systems, wenn die RDF-Kollektionen erstellt werden. Um das Problem zu umgehen, besitzt das System die Möglichkeit die Zuordnung manuell in kleinere Gruppen zu teilen. Dadurch kann das System die Eröffnungen der ECO-Gruppen von A bis E einzeln betrachten. Das Risiko eines Absturzes der Datenbank wird damit vermieden.

## Kapitel 7

# Zusammenfassung

*Semantic Chess* hat das Ziel Fragen zum Thema Schach zu beantworten. Es ist ein Schach-*Question-Answering-System*, das interessierten Nutzern bei der Suche nach Informationen in den zahlreichen PGN-Dateien helfen kann. Nutzer können mit der natürlichen Sprache das System befragen und erhalten von ihm eine Antwort in einer Tabelle. Mit einem Mikro-*F-Measure* von 80,17% und einem Makro-*F-Measure* von 83,47% ist es in der Lage die meisten Trainingsfragen zu beantworten. Auf die Testfragen findet das System mehrheitlich Antworten und erreicht dabei beim *F-Measure* einen Mikrowert von 66,67% und einen Makrowert von 57,39%.

Um ein effektives System zu entwickeln, wurden zunächst der Stand der Technik und die verschiedenen Ansätze im Bereich des *Question Answering* betrachtet. Es wurden die verschiedenen Herausforderungen erläutert, die bei der Entwicklung von QAS auftreten. Anschließend hat diese Arbeit sie auf das Thema Schach übertragen und erklärt, welche Ansätze möglich sind, um die auftretenden Schwierigkeiten so gut wie möglich zu bewältigen. Die lexikalische Lücke und die Ambiguität spielen eine größere Rolle, denn Schach besitzt ein umfangreiches Vokabular. Vor allem bei den Eröffnungen und bei Stellungsmustern treten zahlreiche Begriffe auf. Das System muss sie gruppieren und bestimmte Schlüsselwörter wiedererkennen. Dabei treten auch, durch die Geschichte des Schachs bedingt, vereinzelt deutsche, französische und italienische Wörter auf, die in das englische Vokabular übertragen wurden und die das QAS korrekt interpretieren muss. Im System befinden sich manuell angelegte Templates, durch die es möglich sein soll auch komplexe Anfragen zu beantworten. Es existieren allerdings noch nicht genügend unterschiedliche Vorlagen, um eine komplette Abdeckung der Fragen zu ermöglichen. Die Herausforderungen zum verteilten Wissen, sowie zu prozeduralen, zeitlichen und räumlichen Fragen sind

kaum in das finale System mit eingeflossen. Schachpartien im RDF-Format liegen zur Zeit nicht im Web vor. Das System kann daher nicht auf verschiedene Quellen zurückgreifen, um Wissen zu sammeln und zusammenzufassen. Prozedurale, zeitliche und räumliche Fragen kann *Semantic Chess* nur beantworten, wenn sie sich auf die Metadaten und Züge aus PGN-Dateien beziehen. Fragen, die sich auf Stellungsmuster beziehen, benötigen andere Lösungsansätze und komplexere RDF-Daten, um den Nutzer Antworten zu liefern.

*Semantic Chess* ist ein Template-basiertes QAS und gehört der Gruppe der *closed domains* an. Es besteht aus einem Front-, Backend und einer Datenbank. Das Frontend dient zur Kommunikation mit dem Nutzer und zur Präsentation der Antworten. Das Backend kann das Dateiformat PGN in RDF-Daten umwandeln und besitzt einen Aufbau, der vier Zustände und sieben Prozesse enthält, um natürlichsprachige Fragen in ein SPARQL Query umzuwandeln. Zunächst verschlagwortet das System die Frage. Mit den getaggten Wörtern versucht es allgemeine und schachspezifische Entitäten zu identifizieren. Daraus erstellt das System eine semantische Repräsentation, die es dann mit einem Sequenzcode einem Template zuordnet. Nachdem *Semantic Chess* die Platzhalter in der ausgewählten Vorlage austauscht, kann das System die Datenbank mit dem SPARQL Query befragen und die Antworten dem Nutzer auf einer Weboberfläche präsentieren. Die Datenbank dient zur Speicherung der RDF-Daten.

Um *Semantic Chess* zu evaluieren, wurde ein Benchmark entwickelt, das sich an der *QALD challenge* orientiert. Es enthält 50 Trainings- und zehn Testfragen. Um möglichst viele Fragen zu beantworten, wurden im System drei verschiedene Funktionen getestet, die verschiedene Verfahren besitzen, um die Templates zu befüllen. Die Funktion *regexEntities()* erstellt für jede gefundene Entität einen *regex*-Filter. Es besitzt einen Mikro-*F-Measure* von 60,05%. *distanceEntities()* vergleicht jede Entität mit den Informationen in der Datenbank und schreibt mit Hilfe der Jaccard-Distanz die passendste in den entsprechenden Platzhalter der Vorlage. Dabei kann sie Rechtschreibfehler korrigieren und alternative Schreibweisen einer Entität finden. Durch diese Ersetzung können dem System Antworten entgehen, da auch andere Informationen dazu beitragen können bzw. bessere Antworten erzeugen. Mit einem Mikro-*F-Measure* von 76,01% besitzt *distanceEntities()* den besten Wert. Die Funktion *subStringEntities()* sammelt alle Informationen aus der Datenbank, die die Entität als Zeichenkette enthalten und listet sie in einer *VALUES*-Klausel auf. Im Gegensatz zur davor genannten Funktion entgehen dem System hier keine Informationen. Die Funktion kann allerdings keine Rechtschreibfehler korrigieren oder alternative Schreibweisen finden. Außerdem können aufgrund der Masse an Informationen die

*VALUES*-Klauseln stark anwachsen und viele Entitäten enthalten. Es besitzt einen Mikro-*F-Measure* von 65,75%. Die beiden zuletzt genannten Funktionen besitzen insgesamt die besten Werte bei der *Precision*, dem *Recall* und *F-Measure*. Damit sich ihre Vorteile ergänzen und sich ihre Nachteile abschwächen, wurden sie kombiniert. Die daraus hervorgegangene Funktion *substringWithDistanceFallback()* besitzt daher die Möglichkeit noch mehr Fragen zu beantworten. Das System braucht dafür zwischen weniger als einer und mehreren Sekunden Zeit. Die Analyse der Frage mit dem *Parser* und die Zuordnung des Templates, sowie die Ersetzung der Platzhalter nimmt nur etwa 0,2 bis 0,4 Sekunden in Anspruch. Je nach Komplexität der SPARQL Query (z. B. Anzahl der Entitäten in der *VALUES*-Klausel oder *regex*-Filter) braucht die Abfrage der Datenbank dann mehrere Sekunden, um eine Antwort zu finden.

Der Benchmark zeigt, dass *Semantic Chess* Fragen beantworten kann, die sich an den Metainformationen und Zügen der PGN-Dateien orientieren. Es zeigt aber noch Lücken, wenn Fragen verschachtelte Queries, mehrere Vereinigungen (*UNION*-Klauseln) oder Zeitspannen benötigen. Eine Weiterentwicklung des Systems kann die Lücken angehen und Funktionalitäten ergänzen. Allgemein gilt, dass die Ontologie und Konvertierung von PGN- in RDF-Daten aus dem Jahr 2013 und das Schachvokabular weiterhin auf Aktualisierungen geprüft werden müssen, damit *Semantic Chess* auch in Zukunft einsatzbereit bleibt und leicht weiterzuentwickeln ist.

# Literatur

- [1] André Freitas; et al. *Treo: Combining Entity-Search, Spreading Activation and Semantic Relatedness for Querying Linked Data*. report. Digital Enterprise Research Institute (DERI), National University of Ireland, Galway, 2011. url: [http://andrefreitas.org/papers/qald2011\\_preprint.pdf](http://andrefreitas.org/papers/qald2011_preprint.pdf) (besucht am 17. 11. 2017).
- [2] Christopher D. Manning; et al. *Evaluation of text classification*. 2009. url: <https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-text-classification-1.html> (besucht am 05. 01. 2018).
- [3] Gérard Treptow; et al. *Semantic Chess - Handbuch*. 2013. url: <https://github.com/semanticchess/semanticchess/blob/master/documents/Handbuch-swp-sc13.pdf> (besucht am 17. 11. 2017).
- [4] Jonas Herig; et al. *Entwurfsbeschreibung*. 2015. url: <https://github.com/semanticchess/semanticchess/blob/master/documents/Entwurfsbeschreibung-swp-sc15.pdf> (besucht am 18. 11. 2017).
- [5] Qingyun Wu; et al. al. *A Combinatorial Game Theoretic Analysis of Chess Endgames*. report. Stanford University, Department of Management Science und Engineering, 2010. url: <http://web.stanford.edu/~wqy/research/Math%5C%20191-%5C%20Chess.pdf> (besucht am 28. 11. 2017).
- [6] Corey Allen. *Star Trek - Encounter at Farpoint (episode)*. 1987. url: <https://www.youtube.com/watch?v=MA1hD3XR1h0> (besucht am 30. 12. 2017).
- [7] Angularjs.org. *AngularJS*. 2010. url: <https://angularjs.org/> (besucht am 21. 12. 2017).
- [8] ARVES Chess Endgamestudy Association. *The GBR-code*. 2016. url: <http://www.arves.org/arves/index.php/en/endgamestudies/theory/gbrcode> (besucht am 18. 11. 2017).



- [9] Wolfgang Unzicker; Ernst Bedau. *Gibt es ein Urheberrecht an Schachpartien?* 1994. url: <http://recht.schachbund.de/extras.html?file=files/recht/downloads/extras/gutachten.pdf> (besucht am 20.12.2017).
- [10] Tim Berners-Lee. *Linked Data - Design Issues*. 2006. url: <https://www.w3.org/DesignIssues/LinkedData> (besucht am 11.11.2017).
- [11] BioASQ. *BioASQ - The Challenge. A challenge on large-scale biomedical semantic indexing and question answering*. 2017. url: <http://bioasq.org/> (besucht am 19.11.2017).
- [12] Bootstrap. *Icons*. 2017. url: <https://getbootstrap.com/docs/4.0/extend/icons/> (besucht am 04.12.2017).
- [13] Bootstrap. *License FAQs*. 2017. url: <https://v4-alpha.getbootstrap.com/about/license/> (besucht am 21.12.2017).
- [14] chess.com. *Chess Q&A*. 2017. url: <https://www.chess.com/forum/view/scholastic-chess/chess-q-a> (besucht am 28.12.2017).
- [15] Chess-db.com. *About Chess-db.com*. 2017. url: <https://chess-db.com/public/about.html> (besucht am 17.11.2017).
- [16] chessok.com. *Lomonosov Endgame Tablebases*. 2013. url: [http://chessok.com/?page\\_id=27966](http://chessok.com/?page_id=27966) (besucht am 28.11.2017).
- [17] Christina Unger; André Freitas; Philipp Cimiano. „An introduction to Question Answering over Linked Data“. In: *Reasoning on the Web in the Big Data Era: 10th International Summer School 2014*. 2014. url: [http://andrefreitas.org/papers/preprint\\_introduction\\_qa\\_linked\\_data.pdf](http://andrefreitas.org/papers/preprint_introduction_qa_linked_data.pdf) (besucht am 10.11.2017).
- [18] Christina Unger; Lorenz Bühmann; Jens Lehmann; Axel-Cyrille Ngonga Ngomo; Daniel Gerber; Philipp Cimiano. „Template-based Question Answering over RDF Data“. In: *WWW 2012 – Session: Ontology Representation and Querying: RDF and SPARQL*. 2012. url: [http://liris.cnrs.fr/~pchampin/enseignement/semweb/\\_static/articles/unger\\_2012.pdf](http://liris.cnrs.fr/~pchampin/enseignement/semweb/_static/articles/unger_2012.pdf) (besucht am 10.11.2017).
- [19] Philipp Cimiano u. a. „Towards portable natural language interfaces to knowledge bases - The case of the ORAKEL system“. In: *Data Knowl. Eng.* 65.2 (2008), S. 325–354. doi: 10.1016/j.datak.2007.10.007. url: <https://doi.org/10.1016/j.datak.2007.10.007>.
- [20] Baku Chess Olympiad Operating Committee. *Teams - Open*. 2016. url: <http://www.bakuchessolympiad.com/content/53> (besucht am 28.11.2017).

- [21] Apache Commons. *Class JaccardDistance* - *org.apache.commons.text.similarity*. 2014. url: <https://commons.apache.org/proper/commons-text/apidocs/org/apache/commons/text/similarity/JaccardDistance.html> (besucht am 10.12.2017).
- [22] Stanford CoreNLP. *License*. 2014. url: <https://stanfordnlp.github.io/CoreNLP/#license> (besucht am 21.12.2017).
- [23] Gady Costeff. *CQL: The Chess Query Language*. 2003. url: <http://gadycosteff.com/cql/> (besucht am 18.11.2017).
- [24] Harriet Dennys. *AGON releases new chess player statistics from YouGov*. 2012. url: <https://www.fide.com/component/content/article/1-fide-news/6376-agon-releases-new-chess-player-statistics-from-yougov.html> (besucht am 30.12.2017).
- [25] Treo Deri. *Introducing Treo: Talk to your Data*. 2013. url: <https://youtu.be/Zor2X0uoKsM?t=1m2s> (besucht am 17.11.2017).
- [26] Docker. *Docker Legal Terms - Components & Licenses*. 2017. url: <https://www.docker.com/components/licenses> (besucht am 21.12.2017).
- [27] Docker. *What is a Container*. 2017. url: <https://www.docker.com/what-container> (besucht am 18.12.2017).
- [28] Steven J. Edwards. *ICC Help: PGN-spec*. 1994. url: <https://www.chessclub.com/user/help/PGN-spec> (besucht am 18.11.2017).
- [29] David A. Ferrucci u.a. „Building Watson: An Overview of the DeepQA Project“. In: *AI Magazine* 31.3 (2010), S. 59–79. url: <http://www.aaai.org/ojs/index.php/aimagazine/article/view/2303>.
- [30] David A. Ferucci. *The DeepQA Project*. 2010. url: <https://www.research.ibm.com/deepqa/deepqa.shtml> (besucht am 30.12.2017).
- [31] Fide. *Participants of the World Cup 2017*. 2017. url: [http://www.fide.com/images/stories/NEWS\\_2017/FIDE\\_News/World\\_Cup\\_2017/World\\_Cup\\_Participants.pdf](http://www.fide.com/images/stories/NEWS_2017/FIDE_News/World_Cup_2017/World_Cup_Participants.pdf) (besucht am 28.11.2017).
- [32] Johannes Fischer. *Als Deep Blue das Genie Garri Kasparow schlug*. 2016. url: <http://blog.zeit.de/schach/als-deep-blue-das-genie-garry-kasparow-schlug/> (besucht am 17.11.2017).
- [33] FontAwesome. *License - The full details of how Font Awesome is licensed*. 2017. url: <http://fontawesome.io/license/> (besucht am 21.12.2017).

- [34] The Apache Software Foundation. *Licensing of Distribution*. 2017. url: <http://www.apache.org/licenses/> (besucht am 21.12.2017).
- [35] Deutsche Presse-Agentur; Agence France-Presse. *“Watson” weiß die Antwort*. 2011. url: <http://www.zeit.de/digital/internet/supercomputer-watson-jeopardy> (besucht am 30.12.2017).
- [36] Debasis Ganguly, Johannes Leveling und Gareth J. F. Jones. „Retrieval of similar chess positions“. In: *The 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '14, Gold Coast , QLD, Australia - July 06 - 11, 2014*. 2014, S. 687–696. doi: 10.1145/2600428.2609605. url: <http://doi.acm.org/10.1145/2600428.2609605>.
- [37] RDF Working Group. *RDF 1.1 Primer*. 2014. url: <https://www.w3.org/TR/rdf11-primer/> (besucht am 11.11.2017).
- [38] Poonam Gupta; Vishal Gupta. „A Survey of Text Question Answering Techniques“. In: *International Journal of Computer Applications* (2012). url: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.258.7801&rep=rep1&type=pdf> (besucht am 11.11.2017).
- [39] Steven Hewitt. *Question answering with TensorFlow*. 2017. url: <https://www.oreilly.com/ideas/question-answering-with-tensorflow> (besucht am 10.11.2017).
- [40] Konrad Höffner u. a. „Survey on challenges of Question Answering in the Semantic Web“. In: *Semantic Web* 8.6 (2017), S. 895–920. doi: 10.3233/SW-160247. url: <https://doi.org/10.3233/SW-160247>.
- [41] Volkan Cirik; Louis-Philippe Morency; Eduard Hovy. *Chess Q&A : Question Answering on Chess Games*. report. Pittsburgh USA: Department of Computer Science, Language Technologies Institute, Carnegie Mellon University, 2015. url: [http://www.thespermwhale.com/jaseweston/ram/papers/paper\\_4.pdf](http://www.thespermwhale.com/jaseweston/ram/papers/paper_4.pdf) (besucht am 18.11.2017).
- [42] Volkan Cirik; Louis-Philippe Morency; Eduard Hovy. *Chess Q&A : Question Answering on Chess Games*. 2015. url: [http://www.thespermwhale.com/jaseweston/ram/slides/session3/chessQA\\_volkanCirik.pdf](http://www.thespermwhale.com/jaseweston/ram/slides/session3/chessQA_volkanCirik.pdf) (besucht am 18.11.2017).
- [43] Gerd Isenberg. *Extended Position Description*. 2008. url: <https://chessprogramming.wikispaces.com/Extended%20Position%20Description> (besucht am 18.11.2017).
- [44] Apache Jena. *Jena ARQ*. 2011. url: <https://jena.apache.org/> (besucht am 21.12.2017).

- [45] Abraham Bernstein; Esther Kaufmann; Christian Kaiser; Christoph Kiefer. „Ginseng: A Guided Input Natural Language Search Engine for Querying Ontologies“. In: *2006 Jena User Conference, Bristol, UK*. 2006. url: <https://www.merlin.uzh.ch/contributionDocument/download/2188> (besucht am 18.11.2017).
- [46] Adila Krishnathi u. a. „An Ontology Design Pattern for Chess Games“. In: *Proceedings of the 6th Workshop on Ontology and Semantic Web Patterns (WOP 2015) co-located with the 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pennsylvania, USA, October 11, 2015*. 2015. url: <https://delicias.dia.fi.upm.es/members/vrodriguez/pdf/2015.09.wop.pdf>.
- [47] Markus Liebelt. *License*. 2014. url: <https://github.com/mliebelt/PgnViewerJS/blob/master/LICENSE.md> (besucht am 21.12.2017).
- [48] Vanessa López u. a. „PowerAqua: Supporting users in querying and exploring the Semantic Web“. In: *Semantic Web 3.3 (2012)*, S. 249–265. doi: 10.3233/SW-2011-0030. url: [http://www.semantic-web-journal.net/sites/default/files/swj102\\_2.pdf](http://www.semantic-web-journal.net/sites/default/files/swj102_2.pdf).
- [49] Ajitkumar M. Pundge; Khillare S.A.; C. Namrata Mahender. „Question Answering System, Approaches and Techniques: A Review“. In: *International Journal of Computer Applications* (2016). url: <https://pdfs.semanticscholar.org/c9fa/0d16384ef537077185b659b0e0e7adee61fb.pdf> (besucht am 16.11.2017).
- [50] Ricardo Usbeck; Axel-Cyrille Ngonga Ngomo; Bastian Haarmann; Anastasia Krithara; Michael Röder; Giulio Napolitano. „7th Open Challenge on Question Answering over Linked Data (QALD-7)“. In: *ESWC Conference Series*. 2017. url: [https://svn.aksw.org/papers/2017/ESWC\\_2017\\_QALD/public.pdf](https://svn.aksw.org/papers/2017/ESWC_2017_QALD/public.pdf) (besucht am 17.11.2017).
- [51] Axel-Cyrille Ngonga Ngomo u. a. „Holistic and scalable ranking of RDF data“. In: *2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017*. 2017, S. 746–755. doi: 10.1109/BigData.2017.8257990. url: [https://svn.aksw.org/papers/2017/ESWC\\_HARE/public.pdf](https://svn.aksw.org/papers/2017/ESWC_HARE/public.pdf).
- [52] Pete. *Chess.com Announces Computer Chess Championship*. 2017. url: <https://www.chess.com/article/view/chess-com-announces-computer-chess-championship> (besucht am 17.11.2017).
- [53] Spring Projects. *Serving Web Content with Spring MVC*. 2017. url: <https://spring.io/guides/gs/serving-web-content/> (besucht am 24.01.2018).

- [54] Spring Projects. *Spring Boot - License*. 2014. url: <https://github.com/spring-projects/spring-boot/blob/master/LICENSE.txt> (besucht am 21.12.2017).
- [55] Pranav Rajpurkar. *SQuAD. The Stanford Question Answering Dataset*. 2016. url: <https://rajpurkar.github.io/SQuAD-explorer/> (besucht am 19.11.2017).
- [56] Víctor Rodríguez-Doncel u. a. „Pattern-Based Linked Data Publication: The Linked Chess Dataset Case“. In: *Proceedings of the 6th International Workshop on Consuming Linked Data co-located with 14th International Semantic Web Conference (ISWC 2105), Bethlehem, Pennsylvania, US, October 12th, 2015*. 2015. url: <https://delicias.dia.fi.upm.es/members/vrodriguez/pdf/2015.10.cold.pdf>.
- [57] Andreas Saremba. *ChessGML: The Why and Wherefore*. 2000. url: <http://www.saremba.de/chessgml/why.htm> (besucht am 18.11.2017).
- [58] Albert Silver. *Introducing the online ChessBase database*. 2014. url: <http://en.chessbase.com/post/introducing-the-online-chessbase-database> (besucht am 17.11.2017).
- [59] Robert F. Simmons. „Answering English questions by computer: a survey“. In: *Commun. ACM* 8.1 (1965), S. 53–70. doi: 10.1145/363707.363732. url: <http://doi.acm.org/10.1145/363707.363732>.
- [60] OpenLink Software. *About OpenLink Virtuoso*. 2015. url: <https://virtuoso.openlinksw.com/> (besucht am 09.12.2017).
- [61] OpenLink Software. *How can I Load Data into Virtuoso in Transaction Mode (ie auto commit OFF)*. 2009. url: <https://www.openlinksw.com/vos/main/Main/VirtTipsAndTricksLoadDataInTransactionMode> (besucht am 09.12.2017).
- [62] OpenLink Software. *License Terms for the Virtuoso Open-Source Edition (VOS)*. 1998. url: <http://vos.openlinksw.com/owiki/wiki/VOS/VOSLicense> (besucht am 21.12.2017).
- [63] W3C. *Linked Data*. 2014. url: <https://www.w3.org/standards/semanticweb/data> (besucht am 11.11.2017).
- [64] W3C. *SPARQL 1.1 Query Language*. 2013. url: <https://www.w3.org/standards/semanticweb/data> (besucht am 11.11.2017).
- [65] Wikipedia. *Glossary of chess*. 2004. url: [https://en.wikipedia.org/wiki/Glossary\\_of\\_chess](https://en.wikipedia.org/wiki/Glossary_of_chess) (besucht am 28.11.2017).

**Anhang A**

**UML Diagramm**

**Softwaretechnik-Praktikum 2015**

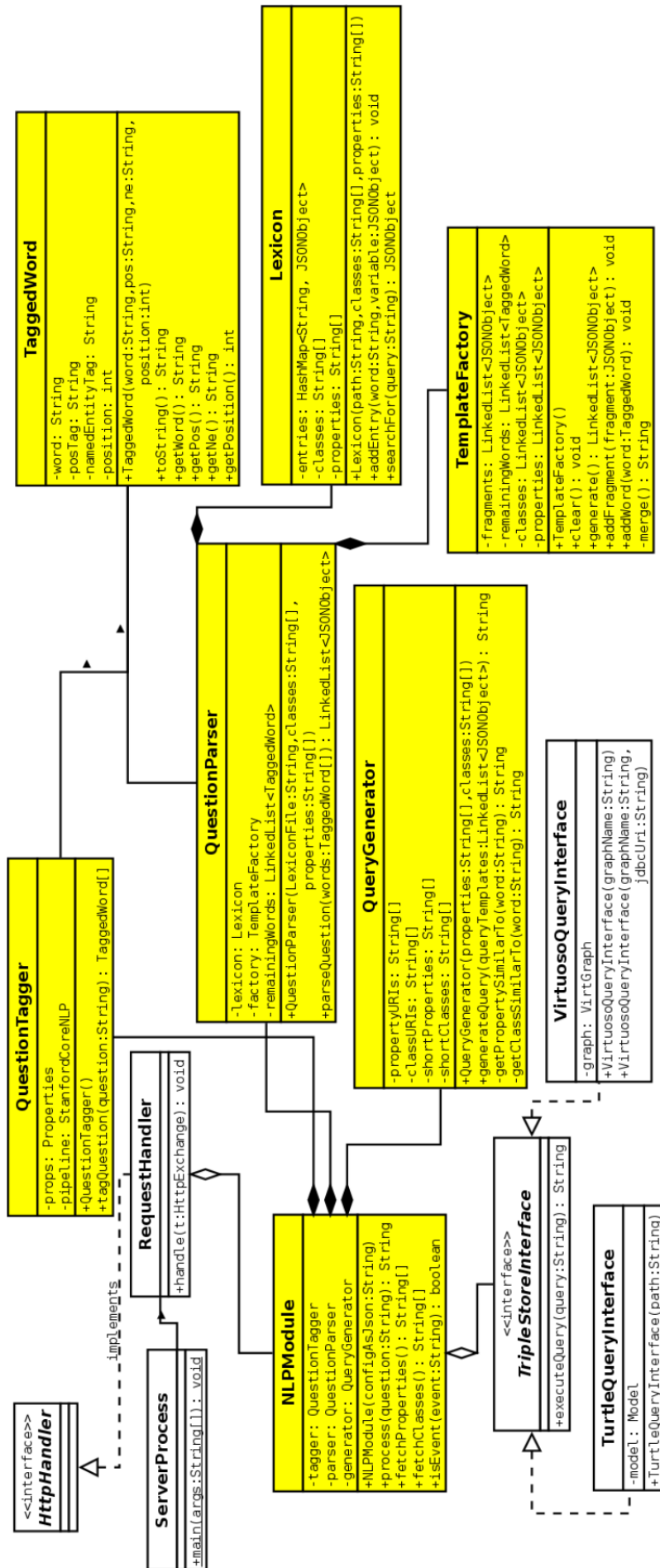


Abbildung A.1: UML Diagramm Softwaretechnik-Praktikum 2015 - Semantic Chess (Quelle: [4], S.5)

## **Anhang B**

### **Benchmark - Auswertungen**



Id	Question	Precision	Recall	F-Measure
1	Show me games by Wilhelm Steinitz.	100.00%	100.00%	100.00%
2	Show me all won games of Emanuel Lasker against Wilhelm Steinitz.	100.00%	100.00%	100.00%
6	Show me endgames with rook against rook and pawn.	100.00%	100.00%	100.00%
7	In which tournaments did Adolf Anderssen and Wilhelm Steinitz play against each other?	100.00%	100.00%	100.00%
8	All losing games of Wilhelm Steinitz from London.	100.00%	100.00%	100.00%
9	Give me 5 games with an underpromotion.	100.00%	100.00%	100.00%
11	What is the longest game?	100.00%	100.00%	100.00%
12	Give me games where Adolf Anderssen defeats Howard Staunton with black.	100.00%	100.00%	100.00%
13	What is the first game of Adolf Anderssen?	100.00%	100.00%	100.00%
16	In which game did Wilhelm Steinitz have his highest rating?	100.00%	100.00%	100.00%
20	How many times has Adolf Anderssen played draw?	100.00%	100.00%	100.00%
21	Which game was played exactly 150 years ago?	100.00%	100.00%	100.00%
24	What was the last game of Wilhelm Steinitz?	100.00%	100.00%	100.00%
27	Which players with black defeated Howard Staunton?	100.00%	100.00%	100.00%
28	All games with more than 100 moves.	100.00%	100.00%	100.00%
30	Show me positions with rook against bishop pair.	100.00%	100.00%	100.00%
31	Show me wins in C33.	100.00%	100.00%	100.00%
33	Are there games with f4 d5 Nf3?	100.00%	100.00%	100.00%
34	Wins of white in C52 and an average ELO of over 2400.	100.00%	100.00%	100.00%
37	Show me the 3rd game between Wilhelm Steinitz and Emanuel Lasker.	100.00%	100.00%	100.00%
42	Show me 10 games with a long castling.	100.00%	100.00%	100.00%
43	In which game was the latest castling?	100.00%	100.00%	100.00%
45	Show me all games from Amsterdam.	100.00%	100.00%	100.00%
46	In which game did the latest exchange take place?	100.00%	100.00%	100.00%
47	All games by William Steinitz.	100.00%	100.00%	100.00%
14	Show me games of white in the Evans Gambit.	100.00%	100.00%	100.00%
5	Who did play blind games?	100.00%	100.00%	100.00%
19	What openings were played at the World Championship 1894?	100.00%	100.00%	100.00%
25	How often is Evans Gambit played?	100.00%	100.00%	100.00%
26	When did Wilhelm Steinitz played his last World Championship?	100.00%	100.00%	100.00%
36	What was the first game with a Ruy Lopez Opening?	100.00%	100.00%	100.00%
50	What is the main variation in the Ruy Lopez Opening?	100.00%	100.00%	100.00%
3	Which player with an ELO above 2500 has won most often in 1899?	100.00%	100.00%	100.00%
49	In which variation in the Sicilian Defense does black win the most?	100.00%	100.00%	100.00%
15	Which opening has Wilhelm Steinitz played most frequently?	100.00%	100.00%	100.00%
35	Which player loses the most with white.	100.00%	100.00%	100.00%
38	Show me all won games of Lasker against Steinitz.	97.37%	100.00%	98.67%
10	Show me all games of the world championship 1886.	69.70%	100.00%	82.14%
22	What does Emanuel Lasker play against the Sicilian Defense?	62.22%	100.00%	76.71%
41	Examples of games with bishop against knight and 2 pawns.	100.00%	50.00%	66.67%
18	Give me the games in which Wilhelm Steinitz lost against a much weaker player.	42.03%	100.00%	59.18%
4	Did Emanuel Lasker play the opening King's Gambit?	16.67%	100.00%	28.57%
44	All games of Wilhelm Steinitz between 1855 and 1870.	100.00%	12.17%	21.71%
32	Has Wilhelm Steinitz ever played a fianchetto?	0.10%	100.00%	0.21%
17	Show me all the games of the tournament with the highest average rating.	0.00%	0.00%	0.00%
23	Who was the 5th world champion?	0.00%	0.00%	0.00%
29	Show me any games with three pawns on the 2nd rank?	0.00%	0.00%	0.00%
39	What is the lifetime record between Wilhelm Steinitz and Emanuel Lasker.	0.00%	0.00%	0.00%
40	Games with doubling rooks on the 7th rank.	0.00%	0.00%	0.00%
48	What is played against the Italian Game?	0.00%	0.00%	0.00%
	<b>Number of answers</b>			<b>44</b>
	<b>Macro Precision, Recall, F-Measure</b>	<b>81.76%</b>	<b>85.24%</b>	<b>83.47%</b>
	<b>Micro Precision, Recall, F-Measure</b>	<b>75.84%</b>	<b>85.02%</b>	<b>80.17%</b>

Tabelle B.1: Benchmark für Trainingsdaten mit *substringWithDistanceFallback()*

Id	Question	Precision	Recall	F-Measure
1	Show me games by Wilhelm Steinitz.	100,00%	100,00%	100,00%
2	Show me all won games of Emanuel Lasker against Wilhelm Steinitz.	100,00%	100,00%	100,00%
6	Show me endgames with rook against rook and pawn.	100,00%	100,00%	100,00%
7	In which tournaments did Adolf Anderssen and Wilhelm Steinitz play against each other?	100,00%	100,00%	100,00%
8	All losing games of Wilhelm Steinitz from London.	100,00%	100,00%	100,00%
9	Give me 5 games with an underpromotion.	100,00%	100,00%	100,00%
11	What is the longest game?	100,00%	100,00%	100,00%
12	Give me games where Adolf Anderssen defeats Howard Staunton with black.	100,00%	100,00%	100,00%
13	What is the first game of Adolf Anderssen?	100,00%	100,00%	100,00%
16	In which game did Wilhelm Steinitz have his highest rating?	100,00%	100,00%	100,00%
20	How many times has Adolf Anderssen played draw?	100,00%	100,00%	100,00%
21	Which game was played exactly 150 years ago?	100,00%	100,00%	100,00%
24	What was the last game of Wilhelm Steinitz?	100,00%	100,00%	100,00%
27	Which players with black defeated Howard Staunton?	100,00%	100,00%	100,00%
28	All games with more than 100 moves.	100,00%	100,00%	100,00%
30	Show me positions with rook against bishop pair.	100,00%	100,00%	100,00%
31	Show me wins in C33.	100,00%	100,00%	100,00%
33	Are there games with f4 d5 Nf3?	100,00%	100,00%	100,00%
34	Wins of white in C52 and an average ELO of over 2400.	100,00%	100,00%	100,00%
37	Show me the 3rd game between Wilhelm Steinitz and Emanuel Lasker.	100,00%	100,00%	100,00%
42	Show me 10 games with a long castling.	100,00%	100,00%	100,00%
43	In which game was the latest castling?	100,00%	100,00%	100,00%
45	Show me all games from Amsterdam.	100,00%	100,00%	100,00%
46	In which game did the latest exchange take place?	100,00%	100,00%	100,00%
47	All games by William Steinitz.	100,00%	100,00%	100,00%
14	Show me games of white in the Evans Gambit.	100,00%	60,04%	75,03%
5	Who did play blind games?	100,00%	24,24%	39,02%
19	What openings were played at the World Championship 1894?	0,00%	0,00%	0,00%
25	How often is Evans Gambit played?	0,00%	0,00%	0,00%
26	When did Wilhelm Steinitz played his last World Championship?	0,00%	0,00%	0,00%
36	What was the first game with a Ruy Lopez Opening?	100,00%	100,00%	100,00%
50	What is the main variation in the Ruy Lopez Opening?	100,00%	100,00%	100,00%
3	Which player with an ELO above 2500 has won most often in 1899?	100,00%	100,00%	100,00%
49	In which variation in the Sicilian Defense does black win the most?	100,00%	100,00%	100,00%
15	Which opening has Wilhelm Steinitz played most frequently?	100,00%	100,00%	100,00%
35	Which player loses the most with white.	100,00%	100,00%	100,00%
38	Show me all won games of Lasker against Steinitz.	0,00%	0,00%	0,00%
10	Show me all games of the world championship 1886.	69,70%	100,00%	82,14%
22	What does Emanuel Lasker play against the Sicilian Defense?	0,00%	0,00%	0,00%
41	Examples of games with bishop against knight and 2 pawns.	100,00%	100,00%	100,00%
18	Give me the games in which Wilhelm Steinitz lost against a much weaker player.	42,03%	100,00%	59,18%
4	Did Emanuel Lasker play the opening King's Gambit?	16,67%	100,00%	28,57%
44	All games of Wilhelm Steinitz between 1855 and 1870.	100,00%	12,17%	21,70%
32	Has Wilhelm Steinitz ever played a fianchetto?	0,10%	100,00%	0,21%
17	Show me all the games of the tournament with the highest average rating.	0,00%	0,00%	0,00%
23	Who was the 5th world champion?	0,00%	0,00%	0,00%
29	Show me any games with three pawns on the 2nd rank?	0,00%	0,00%	0,00%
39	What is the lifetime record between Wilhelm Steinitz and Emanuel Lasker.	0,00%	0,00%	0,00%
40	Games with doubling rooks on the 7th rank.	0,00%	0,00%	0,00%
48	What is played against the Italian Game?	0,00%	0,00%	0,00%
	<b>Number of answers</b>			<b>39</b>
	<b>Macro Precision, Recall and F-Measure</b>	<b>72,57%</b>	<b>73,93%</b>	<b>73,24%</b>
	<b>Micro Precision, Recall and F-Measure</b>	<b>74,02%</b>	<b>78,11%</b>	<b>76,01%</b>

Tabelle B.2: Benchmark für Trainingsdaten mit *distanceEntities()*

Id	Question	Precision	Recall	F-Measure
1	Show me games by Wilhelm Steinitz.	100,00%	100,00%	100,00%
2	Show me all won games of Emanuel Lasker against Wilhelm Steinitz.	100,00%	100,00%	100,00%
6	Show me endgames with rook against rook and pawn.	100,00%	100,00%	100,00%
7	In which tournaments did Adolf Anderssen and Wilhelm Steinitz play against each other?	100,00%	100,00%	100,00%
8	All losing games of Wilhelm Steinitz from London.	100,00%	100,00%	100,00%
9	Give me 5 games with an underpromotion.	100,00%	100,00%	100,00%
11	What is the longest game?	100,00%	100,00%	100,00%
12	Give me games where Adolf Anderssen defeats Howard Staunton with black.	100,00%	100,00%	100,00%
13	What is the first game of Adolf Anderssen?	100,00%	100,00%	100,00%
16	In which game did Wilhelm Steinitz have his highest rating?	100,00%	100,00%	100,00%
20	How many times has Adolf Anderssen played draw?	100,00%	100,00%	100,00%
21	Which game was played exactly 150 years ago?	100,00%	100,00%	100,00%
24	What was the last game of Wilhelm Steinitz?	100,00%	100,00%	100,00%
27	Which players with black defeated Howard Staunton?	100,00%	100,00%	100,00%
28	All games with more than 100 moves.	100,00%	100,00%	100,00%
30	Show me positions with rook against bishop pair.	100,00%	100,00%	100,00%
31	Show me wins in C33.	100,00%	100,00%	100,00%
33	Are there games with f4 d5 Nf3?	100,00%	100,00%	100,00%
34	Wins of white in C52 and an average ELO of over 2400.	100,00%	100,00%	100,00%
37	Show me the 3rd game between Wilhelm Steinitz and Emanuel Lasker.	100,00%	100,00%	100,00%
42	Show me 10 games with a long castling.	100,00%	100,00%	100,00%
43	In which game was the latest castling?	100,00%	100,00%	100,00%
45	Show me all games from Amsterdam.	100,00%	100,00%	100,00%
46	In which game did the latest exchange take place?	100,00%	100,00%	100,00%
47	All games by William Steinitz.	0,00%	0,00%	0,00%
14	Show me games of white in the Evans Gambit.	100,00%	100,00%	100,00%
5	Who did play blind games?	100,00%	100,00%	100,00%
19	What openings were played at the World Championship 1894?	100,00%	100,00%	100,00%
25	How often is Evans Gambit played?	100,00%	100,00%	100,00%
26	When did Wilhelm Steinitz played his last World Championship?	100,00%	100,00%	100,00%
36	What was the first game with a Ruy Lopez Opening?	0,00%	0,00%	0,00%
50	What is the main variation in the Ruy Lopez Opening?	0,00%	0,00%	0,00%
3	Which player with an ELO above 2500 has won most often in 1899?	100,00%	100,00%	100,00%
49	In which variation in the Sicilian Defense does black win the most?	100,00%	100,00%	100,00%
15	Which opening has Wilhelm Steinitz played most frequently?	100,00%	100,00%	100,00%
35	Which player loses the most with white.	100,00%	100,00%	100,00%
38	Show me all won games of Lasker against Steinitz.	97,37%	100,00%	98,67%
10	Show me all games of the world championship 1886.	69,70%	100,00%	82,14%
22	What does Emanuel Lasker play against the Sicilian Defense?	62,22%	100,00%	76,71%
41	Examples of games with bishop against knight and 2 pawns.	100,00%	50,00%	66,67%
18	Give me the games in which Wilhelm Steinitz lost against a much weaker player.	42,03%	100,00%	59,18%
4	Did Emanuel Lasker play the opening King's Gambit?	0,00%	0,00%	0,00%
44	All games of Wilhelm Steinitz between 1855 and 1870.	100,00%	12,17%	21,70%
32	Has Wilhelm Steinitz ever played a fianchetto?	0,10%	100,00%	0,21%
17	Show me all the games of the tournament with the highest average rating.	0,00%	0,00%	0,00%
23	Who was the 5th world champion?	0,00%	0,00%	0,00%
29	Show me any games with three pawns on the 2nd rank?	0,00%	0,00%	0,00%
39	What is the lifetime record between Wilhelm Steinitz and Emanuel Lasker.	0,00%	0,00%	0,00%
40	Games with doubling rooks on the 7th rank.	0,00%	0,00%	0,00%
48	What is played against the Italian Game?	0,00%	0,00%	0,00%
	<b>Number of answers</b>			<b>40</b>
	<b>Macro Precision, Recall and F-Measure</b>	<b>75,43%</b>	<b>77,24%</b>	<b>76,33%</b>
	<b>Micro Precision, Recall and F-Measure</b>	<b>69,49%</b>	<b>62,40%</b>	<b>65,75%</b>

Tabelle B.3: Benchmark für Trainingsdaten mit *subStringEntities()*

Id	Question	Precision	Recall	F-Measure
1	Show me games by Wilhelm Steinitz.	100,00%	100,00%	100,00%
2	Show me all won games of Emanuel Lasker against Wilhelm Steinitz.	0,00%	0,00%	0,00%
6	Show me endgames with rook against rook and pawn.	100,00%	100,00%	100,00%
7	In which tournaments did Adolf Anderssen and Wilhelm Steinitz play against each other?	100,00%	100,00%	100,00%
8	All losing games of Wilhelm Steinitz from London.	0,00%	0,00%	0,00%
9	Give me 5 games with an underpromotion.	100,00%	100,00%	100,00%
11	What is the longest game?	100,00%	100,00%	100,00%
12	Give me games where Adolf Anderssen defeats Howard Staunton with black.	100,00%	100,00%	100,00%
13	What is the first game of Adolf Anderssen?	100,00%	100,00%	100,00%
16	In which game did Wilhelm Steinitz have his highest rating?	0,00%	0,00%	0,00%
20	How many times has Adolf Anderssen played draw?	100,00%	100,00%	100,00%
21	Which game was played exactly 150 years ago?	100,00%	100,00%	100,00%
24	What was the last game of Wilhelm Steinitz?	100,00%	100,00%	100,00%
27	Which players with black defeated Howard Staunton?	100,00%	100,00%	100,00%
28	All games with more than 100 moves.	100,00%	100,00%	100,00%
30	Show me positions with rook against bishop pair.	100,00%	100,00%	100,00%
31	Show me wins in C33.	0,00%	0,00%	0,00%
33	Are there games with f4 d5 Nf3?	0,00%	0,00%	0,00%
34	Wins of white in C52 and an average ELO of over 2400.	100,00%	100,00%	100,00%
37	Show me the 3rd game between Wilhelm Steinitz and Emanuel Lasker.	100,00%	100,00%	100,00%
42	Show me 10 games with a long castling.	100,00%	100,00%	100,00%
43	In which game was the latest castling?	100,00%	100,00%	100,00%
45	Show me all games from Amsterdam.	100,00%	100,00%	100,00%
46	In which game did the latest exchange take place?	100,00%	100,00%	100,00%
47	All games by William Steinitz.	0,00%	0,00%	0,00%
14	Show me games of white in the Evans Gambit.	100,00%	100,00%	100,00%
5	Who did play blind games?	100,00%	100,00%	100,00%
19	What openings were played at the World Championship 1894?	100,00%	100,00%	100,00%
25	How often is Evans Gambit played?	100,00%	100,00%	100,00%
26	When did Wilhelm Steinitz played his last World Championship?	100,00%	100,00%	100,00%
36	What was the first game with a Ruy Lopez Opening?	0,00%	0,00%	0,00%
50	What is the main variation in the Ruy Lopez Opening?	0,00%	0,00%	0,00%
3	Which player with an ELO above 2500 has won most often in 1899?	0,00%	0,00%	0,00%
49	In which variation in the Sicilian Defense does black win the most?	100,00%	100,00%	100,00%
15	Which opening has Wilhelm Steinitz played most frequently?	100,00%	100,00%	100,00%
35	Which player loses the most with white.	100,00%	100,00%	100,00%
38	Show me all won games of Lasker against Steinitz.	0,00%	0,00%	0,00%
10	Show me all games of the world championship 1886.	69,70%	100,00%	82,14%
22	What does Emanuel Lasker play against the Sicilian Defense?	62,22%	100,00%	76,71%
41	Examples of games with bishop against knight and 2 pawns.	100,00%	50,00%	66,67%
18	Give me the games in which Wilhelm Steinitz lost against a much weaker player.	0,00%	0,00%	0,00%
4	Did Emanuel Lasker play the opening King's Gambit?	0,00%	0,00%	0,00%
44	All games of Wilhelm Steinitz between 1855 and 1870.	100,00%	12,17%	21,70%
32	Has Wilhelm Steinitz ever played a fianchetto?	0,00%	0,00%	0,00%
17	Show me all the games of the tournament with the highest average rating.	0,00%	0,00%	0,00%
23	Who was the 5th world champion?	0,00%	0,00%	0,00%
29	Show me any games with three pawns on the 2nd rank?	0,00%	0,00%	0,00%
39	What is the lifetime record between Wilhelm Steinitz and Emanuel Lasker.	0,00%	0,00%	0,00%
40	Games with doubling rooks on the 7th rank.	0,00%	0,00%	0,00%
48	What is played against the Italian Game?	0,00%	0,00%	0,00%
	<b>Number of answers</b>			<b>31</b>
	<b>Macro Precision, Recall and F-Measure</b>	<b>60,64%</b>	<b>59,24%</b>	<b>59,93%</b>
	<b>Micro Precision, Recall and F-Measure</b>	<b>94,73%</b>	<b>43,96%</b>	<b>60,05%</b>

Tabelle B.4: Benchmark für Trainingsdaten mit *regexEntities()*

## **Anhang C**

### **Schachvokabular**

Entität	Schlüsselwörter
0-1	lose, loss
1-0	0-1, 1-0, beat, checkmate, defeat, mate, victory, win
1/2-1/2	1/2, 1/2-1/2, draw, remis, stalemate, tie, threefold
average	average
black	black
count	main, many, most, often
date	date, when
elo	elo, rate, rating
event	championship, event, tournament, tourney
eventEntity	blitz, blind, blindfolded, candidate, challenge, chess960, classics, congress, correspondence, exhibition, festival, final, knockout, kongress, masters, meeting, open, olympiad, playoff, rapid, section, simul, simultan, simultaneous, speed,
fen	capture, castle, castling, exchange, promote, promotion, underpromotion
game	encounter, endgame, game, match, matchup, pairing, position
jjr_neg	below, less, lower, shorter, smaller, under, weaker
jjr_pos	above, beyond, greater, higher, larger, longer, more, over, stronger
jjs_pos	hardest, highest, longest, strongest, toughest
moves	move
opening	attack, counterattack, countergambit, defence, defense, gambit, indian, line, opening, symmetrical, system, variation, wall
ordinal	last
person	opponent, person, player, who, whom
piece	bishop, king, knight, pawn, rook, queen
round	round
site	city, country, place, town, where
temporal	earliest, latest
white	white

Tabelle C.1: Schachvokabular, zur Beantwortung der 50 Trainingsfragen

## **Selbstständigkeitserklärung**

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

_____	_____	_____
Ort	Datum	Unterschrift