Die XXL–Suchmaschine zur ontologiebasierten Ähnlichkeitssuche in XML–Dokumenten

Anja Theobald

Dissertation

zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften
an der Naturwissenschaftlich–Technischen Fakultät I
der Universität des Saarlandes

Mai 2004

Dekan der Naturwissenschaftlich– Technischen Fakultät I:	Prof. Dr. Jörg Eschmeier

Vorsitzender der Prüfungskommission:	Prof. Dr. Gert Smolka
Erstgutachter:	Prof. Dr. Gerhard Weikum
Zweitgutachter:	Prof. Dr. Norbert Fuhr
Tag des Promotionskolloquiums:	16. Juli 2004

Inhaltsverzeichnis

Symbolverzeichnis

Kurzfassung

Die effektive und effiziente Informationssuche in großen Mengen semistrukturierter Daten im XML–Format stehen im Mittelpunkt dieser Arbeit. In dieser Arbeit wird die XXL–Suchmaschine vorgestellt. Sie wertet Anfragen aus, die in der XML–Anfragesprache XXL formuliert sind. Eine XXL–Anfrage umfasst dabei Suchbedingungen an die Struktur und an den Inhalt von XML–Dokumenten. Als Ergebnis wird eine nach ihrer Relevanz absteigend sortierte Liste von Treffern produziert, wobei ein Treffer ein relevantes XML–Dokument oder nur der relevante Teil eines XML–Dokuments sein kann. Die relevanzorientierte Auswertung von gegebenen Suchbedingungen beruht zum einen auf Verfahren aus dem Vektorraummodell und zum anderen wird semantisches Wissen einer quantifizierten Ontologie hinzugezogen. Zu diesem Zweck werden Datenbank–Technologien und Verfahren aus dem Information Retrieval kombiniert, um die Qualität der Suchergebnisse im Vergleich zur traditionellen Stichwortsuche in Textdokumenten zu verbessern. Die hier vorgestellten Konzepte wurden in einem Prototypen implementiert und umfangreich evaluiert.

Abstract

The effective and efficient information retrieval in large sets of semistructured data using the XML format is the main theme of this thesis. This thesis presents the XXL search engine, which executes queries formulated in the XML query language XXL. An XXL query consists of search conditions on the structure and search conditions on the content of XML documents. The result is a ranked result list in descending order of relevance, where a result can be a relevant XML document or only the relevant part of an XML document. The relevance–based query evaluation uses methods from the vector space model and semantic knowledge from a quantified ontology. For this purpose, we combine database technologies and methods from information retrieval to improve the quality of search results in comparison to traditional keyword–based text retrieval. The presented concepts have been implemented and exhaustively evaluated.

Zusammenfassung

Die Popularität von XML, der erweiterten Annotationssprache des World–Wide–Web–Konsortiums (W3C), hat in den letzten Jahren erheblich zugenommen. XML wird in vielen Bereichen zur Repräsentation und zum Austausch von Daten verwendet, z.B. in traditionellen Datenbankmanagementsystemen, in E–Business–Umgebungen, im E–Science–Umfeld, im World Wide Web (Web).

XML–Dokumente unterscheiden sich von traditionellen Textdokumenten, HTML–Dokumenten, etc., durch die flexible Kombination von Struktur– und Inhaltsdaten. Mit Hilfe von Elementen und Attributen wird der Inhalt von XML–Dokumenten strukturiert und durch die sorgfältige Wahl der Element– und Attributnamen zusätzlich semantisch annotiert. Dabei sind die Elemente innerhalb eines XML–Dokuments hierarchisch angeordnet und es besteht die Möglichkeit, beliebige Elemente durch Links miteinander zuverknüpfen. Insbesondere kann der Anteil an Struktur– und Inhaltsdaten von XML–Dokument zu XML–Dokument variieren, weshalb wir hier auch von semistrukturierten Dokumenten sprechen.

Eine zentrale Aufgabe, die sich im Umgang mit großen Mengen von XML–Dokumenten stellt, ist die effektive und effiziente Informationssuche in XML–Dokumenten mit dem Ziel hoher Ergebnisqualität und kurzer Antwortzeit. Dabei wird als Resultat eine Rangliste von relevanten Treffern angestrebt, die absteigend nach ihrer Relevanz sortiert sind. Die Qualität der Rangliste wird zum einen an der Ausbeute und zum anderen an der Präzision gemessen. Dabei gibt die Ausbeute den Anteil der relevanten Treffer an, die gefunden wurden. Die Präzision ergibt sich aus dem Anteil der gefundenen Treffer, die relevant sind. Dabei ist die Relevanz eines Treffers inhärent subjektiv und vage und sehr schwer automatisch abschätzbar.

In dieser Arbeit zeigen wir, das wir im Vergleich zur stichwortbasierten Informationssuche in XML–Dokumenten zum einen durch die Ausnutzung der Semistrukturiertheit bei der Formulierung der Suchanfrage die XML–Suche präzisieren und zum anderen durch den Einsatz von ontologiebasierter semantischer Ähnlichkeit die Qualität der Rangliste erheblich verbessern können.

Die XXL–Suchmaschine ist für die effektive und effiziente Informationssuche in beliebigen XML–Dokumenten konzipiert. Sie verwendet einen gerichteten XML–Graphen als Datenmodell, dessen Knoten die Namen der Elemente und Attribute beinhalten, dessen Blätter mit den Werten von Elementen und Attributen beschriftet sind und dessen gerichtete Kanten die Struktur der XML–Dokumente widerspiegeln.

Suchanfragen werden mit Hilfe der XML–Anfragesprache XXL (flexible XML search language) formuliert. Eine XXL–Anfrage beschreibt mit Hilfe von Struktur– und Inhaltsbedingungen gesuchte Teilgraphen des XML–Graphen. Insbesondere kann die Suche nach semantisch ähnlichen Namen und Werten von Elementen oder Attributen als Suchkriterium in der XXL–Anfrage formuliert werden.

Das Ergebnis einer XXL–Anfrage ist eine absteigend sortierte Rangliste von relevanten Teilgraphen des XML–Graphen. Ein zu einer gegebenen XXL–Anfrage relevanter Teilgraph wird nach dem Prinzip der Graphisomorphie im XML–Graphen gefunden (isomorphe Einbettung unter Berücksichtigung der Knotenbeschriftungen). Hierbei dürfen, bedingt durch entsprechende Ähnlichkeitssuchbedingungen, semantische Abweichungen bei den Namen und Werten des relevanten Teilgraphen auftreten. Der Grad der Abweichung wird durch lokale Relevanzwerte angegeben, aus denen der Relevanzwert des Teilgraphen hervorgeht.

Die lokale Relevanzbewertung von Namen und Werten stützt sich zum einen auf das Vektorraummodell und zum anderen auf einen probabilistischen Ansatz, bei dem eine quantifizierte Ontologie zum Einsatz kommt. Auf der Basis des Vektorraummodells (vector space model) werden Werte von Elementen/Attributen und Inhaltsbedingungen einer XXL–Anfrage durch Vektoren von Termen (Stichwörtern) repräsentiert. Die Relevanz eines Elementwertes ergibt sich dann aus der Distanz beider Vektoren. Typischer Weise werden hierzu statistische Daten über die Häufigkeit der gegebenen Terme im Elementwert (Termfrequenz TF) und die inverse Häufigkeit der Terme in allen vorliegenden Elementwerten (inverse Elementfrequenz IEF) herangezogen. Mit Hilfe eines geeigneten Distanzmaßes (z.B. Kosinusmaß) wird dann die Relevanz, also die inhaltliche Ähnlichkeit, bewertet. Zur Auswertung von semantischen Ähnlichkeitssuchbedingungen wird eine quantifizierte Ontologie verwendet. Diese beinhaltet umfangreiches Wissen über die Art und den Grad von semantischen Beziehungen zwischen Konzepten. Ein Konzept steht für eine Menge von Begriffen (Paaren von Wörtern und ihren Bedeutungen) gleicher Bedeutung. Konzepte und semantische Beziehungen werden aus einem bestehenden Thesaurus übernommen und die Quantifikation basiert auf umfangreichen Webcrawls. Der Grad der semantischen Ähnlichkeit von vorgegebenen und gefundenen Namen bzw. Stichwörtern fließt in den zugehörigen lokalen Relevanzwert ein.

Zur effizienten Auswertung von XXL–Anfragen verwendet die XXL–Suchmaschine geeignete Indexstrukturen. Die Inhaltsdaten der XML–Dokumente (die Element– und Attributwerte) werden im Elementinhaltindex gespeichert. Dieser enthält eine invertierte Liste, so dass effizient stichwortbasiert in den Inhaltsdaten gesucht werden kann. Die Strukturdaten werden im Elementpfadindex gespeichert, der neben den Informationen über die Eltern/Kind– bzw. Vorgänger/Nachfolger–Beziehungen von Elementen auch die Informationen über bestehende transitive Verbindungen zwischen beliebigen Elementen bereithält.

Die XXL–Suchmaschine wurde als Java–basierte Client–Server–Anwendung mit entsprechenden grafischen Benutzeroberflächen implementiert. Sie unterstützt drei voneinander unabhängige Aufgaben, das Crawlen und Indexieren von XML–Dokumenten, den Aufbau und die Pflege einer quantifizierten Ontologie sowie die Auswertung von XXL–Anfragen. Die XML– und die Ontologiedaten werden in einer relationalen Datenbank gespeichert. Als Beleg für die Wirksamkeit der ontologiebasierten Ähnlichkeitssuche in XML–Dokumenten haben wir zum einen eine Reihe gezielter, exemplarischer Experimente und zum anderen einen umfangreichen Benchmarktest durchgeführt. In beiden Fällen steht die Qualität von XXL–Anfrageergebnissen (die Ausbeute und die Präzision) im Vordergrund.

Summary

In the last few years, the popularity of XML, the extensible markup language of the World Wide Web Consortium (W3C), has significantly increased. XML is used to represent and exchange data in many areas, such as traditional database management systems, e–business environments, e–science environments, and World Wide Web (Web).

Compared to traditional text documents, HTML documents, etc., XML documents consist of a flexible combination of structural data and content data. Elements and attributes are used for structuring, and carefully chosen element names and attribute names for the semantic annotation of the document content. Elements are hierarchically organized within a document. In addition, arbitrary elements can be connected via links. Because of varying parts of structural data and content data per document, such XML documents are called semistructured documents.

The effective and efficient information retrieval in XML documents aiming at high quality results with short response time, is the most important aspect in handling a large set of XML documents. The result should be a ranked list of relevant results in descending order of relevance. The quality of a ranked list is measured in terms of recall and precision. Recall is the fraction of the relevant results which have been retrieved. Precision is the fraction of the retrieved results which are relevant. The relevance of a result is inherently subjective and vague, and very difficult to evaluate automatically.

Within this work, we show that compared to keyword–based information search we can increase the precision of the XML search conditions by using the semistructuredness, and we can improve the quality of the ranked result list by using ontology–based semantic similarity.

The XXL search engine is conceived for effective and efficient information search in arbitrary XML documents. It is based on a directed XML graph as the underlying data model, where the nodes contain the element and attribute names, the leaves consist of the element and attribute values, and the directed edges are used to represent the structure of the XML documents.

Search queries are formulated using the XML query language XXL (flexible XML search language). An XXL query describes the searched subgraphs of the XML graph using structural and content conditions. In addition, it is possible to search for names and values of elements or attributes with semantic similarity.

The result of an XXL query is a ranked list of relevant subgraphs of the XML graph sorted in descending order of relevance. A relevant subgraph for a given XXL query is found based on graph isomorphism. According to similarity search conditions, semantic differences in the found names and values of a relevant subgraph can occur. The degree of the differences is expressed by local relevance scores, which are used to build the relevance score for the subgraph.

The local relevance evaluation of names and values is based on the vector space model and on a probabilistic approach that uses a quantified ontology. In the vector space model, the values of elements/attributes and content conditions are represented by vectors of terms (keywords). The relevance score of an element value is based on the distance between the given vectors. For this distance we use statistical data about term frequencies within the element value (TF), and inverse term frequencies in all available element values (IEF). The relevance score of an element value is calculated by appropriate distance measures (e.g. cosinus measure). The evaluation of semantic

similarity search conditions is based on a quantified ontology. The ontology contains exhaustive knowledge about the type and the degree of semantic relationships between concepts. A concept stands for a set of terms (pairs of words and their senses) of the same sense. The concepts and their semantic relationships are generated from an existing thesaurus, and the quantification of the relationships is based on large web crawls. The similarity of given names/keywords and found names/keywords is used in the corresponding local relevance score.

The XXL search engine uses appropriate index structures for the efficient evaluation of XXL queries. The content data of the given XML documents (element and attribute values) is stored in the element content index. In order to execute keyword–based search on content data the element content index contains an inverted list. The structural data of the XML documents is stored in the element path index. This index contains information about parent/child relationships, ancestor/descendant relationships of elements, and also information about the connectivity of elements.

The XXL search engine is implemented as a Java–based client–server application with graphical user interfaces. It supports three independent tasks: crawling and indexing of XML documents, building and maintaining a quantified ontology, and the evaluation of XXL queries. The XML data and the ontology data are stored into a relational database system. As a proof of concept of the ontology–based semantic similarity search in XML documents, a number of special experiments and an exhaustive benchmark test are carried out. In both cases, the quality of XXL query results (recall and precision) is in the main measure of interest.

Kapitel 1

Einleitung

1.1 Motivation

1.1.1 Internet und Web: Status Quo

Der Begriff *Informationsgesellschaft* begründet sich in der technischen, ökonomischen, sozialen und kulturellen Veränderung der Gesellschaft. Die technologische Entwicklung vor allem im Bereich der Informations– und Kommunikationstechnologie prägt das heutige Gesellschaftsbild.

- Daten und Informationen stehen zunehmend elektronisch in großen digitalen Datenbasen zur Verfügung, z.B. in Datenbanken, in digitalen Bibliotheken, in digitalen Archiven, im Web.
- Die Bereitstellung und Verbreitung elektronischer Datenbasen erfolgt mit Hilfe geeigneter digitaler Datennetze, z.B. mit Hilfe von Intranets oder dem Internet.
- Die Extraktion, die Transformation, der Austausch und die Integration alter und neuer Daten aber vor allem die Informationssuche erfordern leistungsfähige Hilfsmittel, z.B. Suchmaschinen.

Die weltweit größte elektronische Datenbasis ist das *World–Wide Web* [WL02]. Das World–Wide Web (kurz: Web) ist ein Netzwerk von Informationsressourcen. 1992 startete CERN (European Organization for Nuclear Research) das WWW–Projekt mit dem Ziel, ein verteiltes Hypermedia–System zu bauen.

Im Web liegen die Daten und Informationen vor allem in Form von Webdokumenten auf der Basis der Hypermedia–Technologie vor. *Hypermedia* [Enc01] bezeichnet die Integration von Bildern, Audios, Videos und Animationen in Dokumenten, die miteinander verknüpft ein assoziatives Informationssystem bilden. Hypermedia–Dokumente enthalten Querverweise, *Hyperlinks* bzw. *Links*, die sie mit anderen Dokumenten verwandten Inhalts verbinden. Diese Links ermöglichen dem Benutzer, von einem Dokument zum nächsten entlang der Assoziation zu navigieren. Beinhaltet ein Hypermedia–Dokument primär textuelle Informationen, wird es auch als *Hypertext* bezeichnet.

Hypertext–Dokumente [Enc01] basieren auf Annotationssprachen (Markup Languages), speziellen Metasprachen, um Dokumenten in Netzwerken Struktur zu geben. Als internationaler Dokumentationsstandard wurde *SGML* (Standard Generalized Markup Language) [ISO8879] entwickelt. Mit Hilfe von Markups können Dokumentbereiche mit bestimmten Eigenschaften festgelegt werden. Ein *Markup* wird durch einen Namen gekennzeichnet. Im Dokument wird ein durch ein Markup markierter Bereich mit Hilfe

eines Starttags und eines Endtags gekennzeichnet. Ein *Tag* enthält dabei immer den Namen des Markups und zusätzlich Metazeichen zur Abgrenzung vom übrigen Dokument.

Hypertext–Dokumente des Webs werden mehrheitlich in einer speziellen Annotationssprache, nämlich *HTML* (Hypertext Markup Language) [LJ99], geschrieben. HTML ist eine konkrete Anwendung von SGML und dient der Präsentation von Dokumenten im Web. HTML stellt eine Menge von vordefinierten Markups zur Verfügung, die Formatierungsregeln beinhalten. Diese Markups, z.B. `HEAD`, `BODY`, `P`, `BR`, legen also fest, wie der darin eingebettete Dokumentbereich im Webbrowser angezeigt wird. Dabei wird der Beginn des markierten Bereiches mit einem entsprechenden Starttag markiert, z.B. `<HEAD>`, und das Ende des entsprechenden Dokumentbereichs durch einen Endtag gekennzeichnet, z.B. `</HEAD>`. Allerdings muss es in HTML–Dokumenten nicht zu jedem Starttag ein Endtag geben. Es gibt ein spezielles Markup `a`, das zur Darstellung von Hyperlinks (kurz: Links) verwendet wird. Dieses Markup ermöglicht den Verweis auf andere Dokumente, Bilder, Audios, Videos, Animationen, etc., wobei das Ziel des Links durch ein Attribut angegeben wird.

Beispiel 1.1 In der Abb. 1.1 sind zwei typische HTML–Dokumente dargestellt. Das linke HTML–Dokument ist eine Zusammenstellung von Informationen über Kunst und Künstler und das rechte HTML–Dokument hat das Shakespeare–Drama "Macbeth" zum Inhalt.

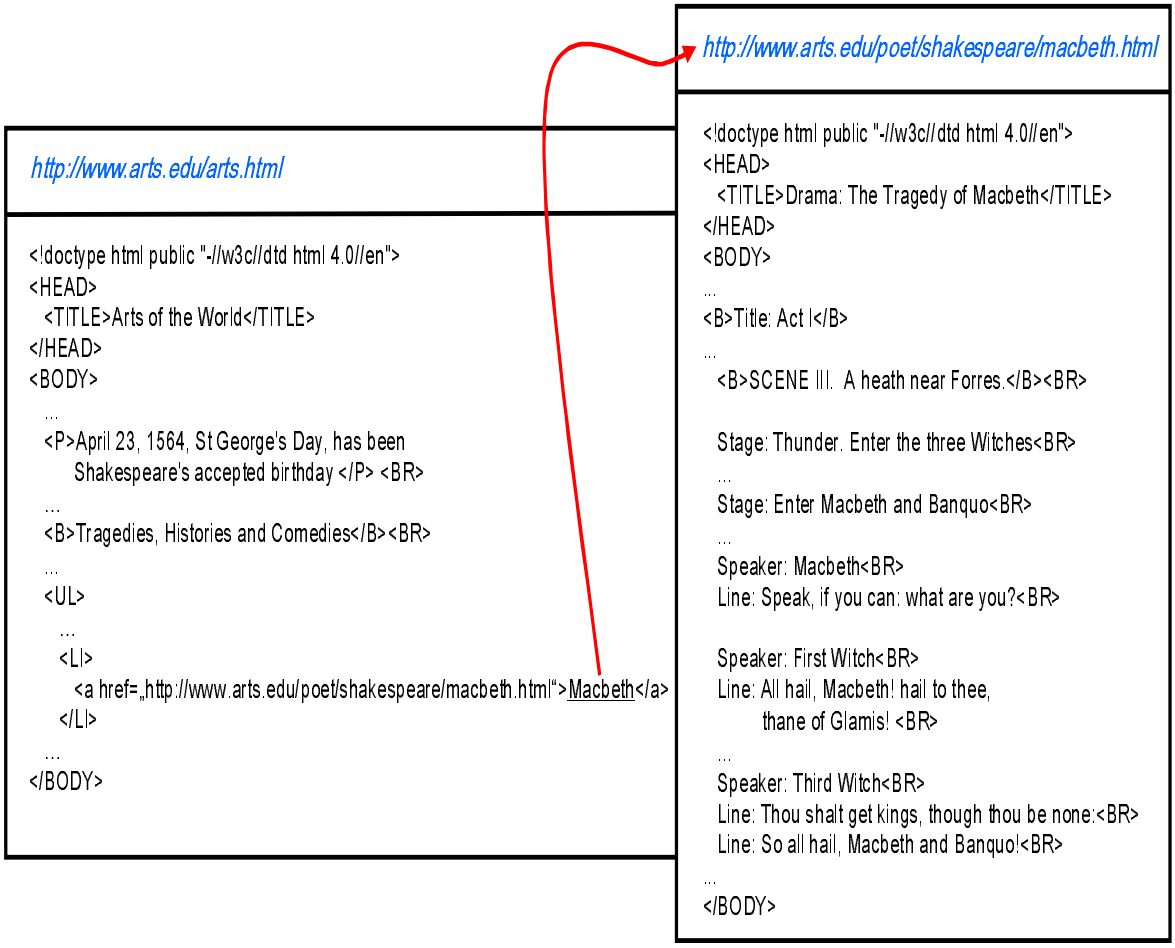


Abbildung 1.1: HTML–Dokument über Künstler (links) und das Shakespeare–Drama "Macbeth" als HTML–Dokument (rechts)

Anhand dieses Beispiels illustrieren wir den klassischen Aufbau von HTML–Dokumenten bestehend aus den Tags `HEAD` und `BODY`. Typisch sind die Formatierungstags, z.B. `<P>` für Paragraph und `<LI>` für Listeneintrag (Listitem). Zusätzlich gibt es hier einen Link vom Literatur–Dokument auf das Macbeth–Dokument, der durch das Tag `<a>` mit dem Attribut `href` gekennzeichnet ist.

Aktuell gibt es mehrere Millionen Webseiten weltweit, die mit Hilfe des Internets bereitgestellt und verbreitet werden. Das *Internet* ist ein Netz von Netzen, das Computer auf der ganzen Welt mittels verschiedenster physikalischer Netzwerktechnologien miteinander verbindet [WL02]. Diese Computer (PCs, Macs, Supercomputer, etc.) verwenden eine Menge von Regeln genannt TCP/IP (Transmission Control Protocol/Internet Protocol) zum Austausch von Datenpaketen. Zu den Diensten, die im Internet bereitgestellt sind, gehören E-Mail, File–Transfer, Newsgroups und das Web.

Im Umgang mit einer riesigen Zahl von primär HTML–basierten Webdokumenten spielen die Extraktion, die Transformation, der Austausch, die Integration alter und neuer Daten und vor allem die Informationssuche eine wichtige Rolle.

Die Informationssuche in HTML–Dokumenten des Webs wird durch die Motivation des Benutzers dominiert. Die Suche ohne genaue Zielvorstellung wird als Blättern oder Browsen bezeichnet. Zur gezielten Suche wird eine Anfragesprache zur Formulierung von adäquaten Suchbedingungen benötigt. Darüber hinaus werden leistungsfähige technische Werkzeuge zur Verarbeitung von solchen Anfragen benötigt. Im Web gibt es diverse Suchhilfen, z.B. thematische Verzeichnisse, Suchmaschinen, Suchagenten.

Definition 1.1 (Suchmaschine)
Ein Benutzer identifiziert oder beschreibt seinen Informationsbedarf durch eine Anfrage zum Beispiel mit Hilfe von Stichwörtern und Phrasen. Eine Suchmaschine ist ein Computerprogramm, das mittels vorgegebener Stichwörter und Phrasen eine Rangliste von Webseiten zurückliefert, die diesen Suchkriterien genügen. □

Die Gruppe der Suchmaschinen lässt sich in universelle Suchmaschinen, spezielle Suchmaschinen (ausgerichtet auf spezielle Themenbereiche) und Meta–Suchmaschinen (die Suchanfragen nicht selbst auswerten, sondern an andere Suchhilfen weiterleiten) unterteilen. Darüber hinaus unterscheidet man zwischen Suchmaschinen, die Indizes benutzen, und solchen, die keine benutzen.

Die gezielte Informationssuche in HTML–Dokumenten des Webs mit Hilfe von Suchmaschinen beginnt typischerweise mit der Formulierung einer Anfrage durch einen Benutzer bestehend aus logisch verknüpften Stichwörtern, die den Inhalt der gesuchten Webdokumente charakterisieren. Die meisten großen Suchmaschinen operieren auf HTML–Dokumenten. Sie parsen (analysieren) diese, speichern repräsentative Daten und Informationen auf der Basis von Stichwörtern und Linkanalysen in geeigneten Datenstrukturen (Indizes) ab. Sie werten die gestellten Anfragen auf der Basis der ihnen bekannten Daten aus und geben eine Rangliste von relevanten Dokumenten zurück, die absteigend nach der Relevanz der Dokumente sortiert ist. Die Relevanz eines Dokumentes beruht hier vor allem statistischen Informationen über Häufigkeiten, mit denen die vorgegebenen Stichwörter in den gefundenen HTML–Dokumenten auftreten.

Beispiel 1.2 Auf der Suche nach Theaterstücken (engl.: play) liefert eine traditionelle Suchmaschine, z.B. Google, zu dem Suchbegriff "`Play`" etwa 75.400.000 Treffer

(13.10.2003). Diese Rangliste ist ein Mix aus Informationen über Sport, Computer, Entertainment, Literatur und anderen Gebiete. Auf der Suche nach Theaterstücken von Shakespeare liefert Google zur Anfrage "`Shakespeare AND Play`" etwa 1.200.000 Treffer, wobei die Dramen nicht in der Trefferliste erscheinen, sondern durch weitere Navigation ausgehend von den Webseiten der Trefferliste erreicht werden.

Auf der Suche nach Theaterstücken von Shakespeare werden die HTML–Dokumente aus Abb. 1.1 ebenfalls nicht gefunden, da im linken Dokument die semantische Ähnlichkeit von "Tragödie" (engl.: tragedy) und "Theaterstück" (engl.: play) nicht erkannt wird und im rechten Dokument der Name "Shakespeare" nicht vorkommt.

Das Dilemma der Informationssuche im Web besteht vor allem in der dezentralen Administration, in der heterogenen Struktur der Webdokumente sowie in der Dynamik des Inhalts dieser Dokumente. Die stichwortbasierte Suche nach konkreten Informationen in den HTML–basierten Dokumenten im Web mit Hilfe von Suchmaschinen führt häufig zu gut bekannten Phänomenen.

1. In vielen Fällen bekommt man gar keinen oder unzählige Treffer.
2. Die einzelnen Dokumente erhalten im Allgemeinen eine unzureichende Relevanzbewertung.
3. Die Anzahl der tatsächlich relevanten Dokumente am Anfang (z.B. unter den Top 10) des Suchergebnisses (Präzision) ist häufig sehr gering.
4. Die Anzahl der tatsächlich relevanten Dokumente zu Beginn des Suchergebnisses (z.B. unter den Top 10) ist häufig im Vergleich zur Gesamtzahl aller relevanten Dokumente (Ausbeute) sehr gering.

1.1.2 Das Web der Zukunft

In den letzten Jahren erlebt die weltweit größte elektronische Datenbasis, das Web, einen Wandel in der Darstellung von Daten und Informationen. Zunehmend werden Webdokumente in der erweiterten Annotationssprache *XML* (eXtensible Markup Language) [BPS+04] statt in HTML geschrieben. Im Gegensatz zu HTML erlaubt XML die freie Definition von Markups. In XML–Dokumenten dienen die Markups zur semantische Annotation von Dokumentbereichen, das Layout wird separat durch entsprechende Stylesheets festgelegt. Ein *XML–Element* (kurz: Element) besteht aus einem Markup (Elementnamen) und dem durch ihn markierten Dokumentbereich (Elementinhalt). Ein XML–Element kann durch XML–Attribute ergänzend beschrieben werden. Ein *XML–Attribut* (kurz: Attribut) besteht aus einem Attributnamen und einem Attributwert und tritt immer zusammen mit dem zugehörigen Elementnamen auf. Zur Abgrenzung von Elementnamen und ihren Attributen vom übrigen Dokumentinhalt werden wiederum Tags definiert, die die Elementnamen, Attribute und Trennzeichen enthalten.

Beispiel 1.3 Der Inhalt der HTML–Dokumente aus Abb. 1.1 wird hier im XML–Format dargestellt. In der Abb. 1.2 ist links unten das Dokument über Künstler zu sehen und rechts werden Ausschnitte aus zwei Shakespeare–Dramen im XML–Format angegeben, die unter anderem über Links vom linken Dokument aus erreichbar sind. Zusätzlich ist links oben ein XML–Dokument angegeben, das eine Sammlung von Zitaten mit Verweisen auf die entsprechenden Quellen enthält.

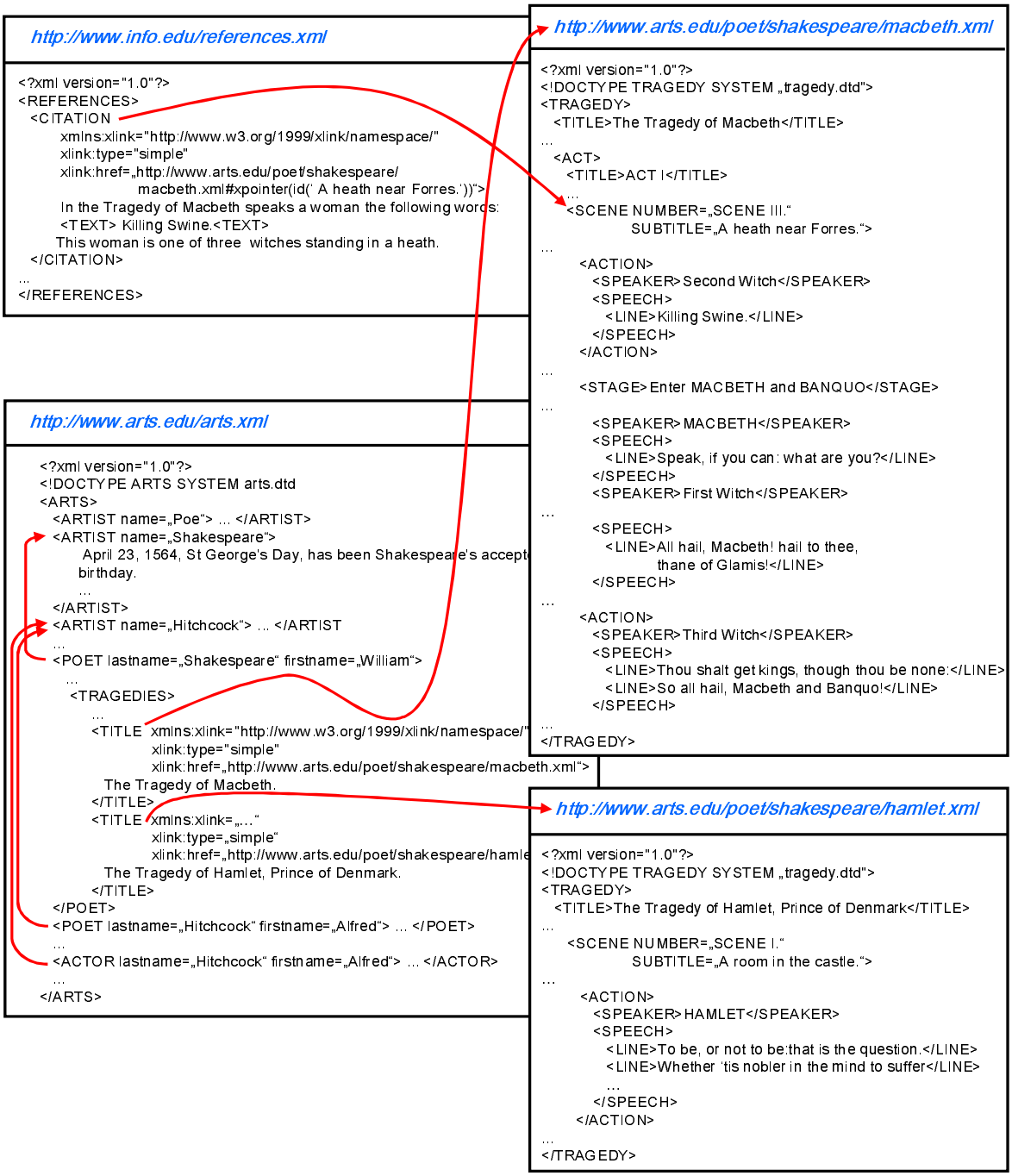


Abbildung 1.2: Eine Referenzliste und eine Kunstquelle als XML–Dokumente (links) mit Verweisen auf diverse Shakespeare–Werke als XML–Dokumente (rechts)

Der Beginn eines Elements, z.B. `ARTIST`, wird durch das Starttag `<ARTIST>` angezeigt. Das Ende eines Elements wird durch das zugehörige Endtag `</ARTIST>` gekennzeichnet. In XML–Dokumenten wird zu jedem Starttag ein Endtag verlangt. Zwischen Start– und Endtag befindet sich der Elementinhalt bestehend aus beliebigen Daten, den Elementwerten, und/oder weiteren Subelementen. Im Starttag eines Elements können außerdem Attribute enthalten sein. Ein Attribut, z.B. `name="Poe"`, besteht immer aus einem Attributnamen und einem Attributwert, der in Anführungszeichen gesetzt ist. Es gibt spezielle Attribute, die zur Realisierung von Verknüpfungen (Links) zwischen verschiedenen Elementen verwendet werden. Diese Links können Elemente eines Dokuments oder Elemente verschiedener Dokumente miteinander verknüpfen (durch Pfeile symbolisiert).

Sorgfältig gewählte Element– und Attributnamen ermöglichen also die Strukturierung und semantische Annotation des Dokumentinhalts. In einem XML–Dokument können XML–Elemente in Reihenfolge oder geschachtelt angeordnet sein. Jedes XML–Dokument wird durch ein eindeutiges *Wurzelelement*, das in keinem anderen Element enthalten ist, gekennzeichnet. Neben der Schachtelung und der sequentiellen Anordnung der Elemente innerhalb eines XML–Dokumentes gibt es Möglichkeiten, Links zwischen Elementen des gleichen oder verschiedener XML–Dokumente zu definieren.

XML–Dokumente zeichnen sich durch die flexible Kombination von Struktur– und Inhaltsdaten aus und werden daher auch als *semistrukturierte* Dokumente bezeichnet. Die *Strukturdaten* ergeben sich aus den vorkommenden Element– und Attributnamen und ihrer Anordnung und dienen der Beschreibung bzw. näheren Erläuterung des zugehörigen Elementinhalts. Die *Inhaltsdaten* setzen sich aus den zugehörigen Element– und Attributwerten der einzelnen XML–Elemente zusammen.

Nach wie vor werden die meisten Anfragen nach Informationen im Web auf der Basis von logisch verknüpften Stichwörtern gestellt und erhalten als Treffer unterschiedlich relevante HTML– und XML–Dokumente. Die besonderen Eigenschaften von XML–Dokumenten bieten dem Benutzer zusätzlich die Möglichkeit, gezielt in den Struktur– und Inhaltsdaten nach Informationen zu suchen.

Eine neue Generation von Suchmaschinen wird benötigt: *XML–fähige Suchmaschinen* zur gezielten Informationssuche in großen XML–Dokumentquellen wie dem Web (oder anderen digitalen Datenbasen). Hierzu werden neue Anfragesprachen benötigt, mit denen man gezielt Suchbedingungen an die Struktur– und die Inhaltsdaten formulieren kann und somit die ursprüngliche Anfrage präzisieren kann. Außerdem sind neue Strategien zur Verarbeitung solcher Anfragen an XML–Dokumente notwendig, bei denen die Relevanzbewertung von XML–Dokumenten unter Berücksichtigung der semantische Annotation der Inhaltsdaten durch die Strukturdaten erfolgt.

1.2 Verwandte Arbeiten

Im Mittelpunkt der vorliegenden Arbeit steht die gezielte Informationssuche (Information Retrieval) in umfangreichen XML–Datenbasen mit dem Ziel hoher Ergebnisqualität und kurzer Antwortzeit. Die neue Herausforderung liegt im effektiven und effizienten Umgang mit großen Mengen von semistrukturierten XML–Dokumenten. Schwerpunkte aktueller Forschungsarbeiten (einen guten Überblick geben [ABS00, KM03, RV03, BGS+03]) liegen vor allem in den Bereichen:

1. XML–Anfragesprachen zur Suche in Struktur– und Inhaltsdaten von XML–Dokumenten,
2. XML–Indexstrukturen zur Unterstützung der effizienten Suche in XML–Daten,
3. Ontologien zur Unterstützung der effektiven, semantischen Ähnlichkeitssuche sowie
4. Benchmarks zur Evaluation von Verfahren zur Informationssuche in XML–Dokumenten.

In den folgenden vier Abschnitten stellen wir die wichtigsten verwandten Arbeiten aus den oben genannten Bereichen vor.

1.2.1 XML–Anfragesprachen

Die Informationssuche in hochgradig strukturierten Datenbasen mit klarer Semantik der enthaltenen Daten (z.B. in relationalen und objektrelationalen Datenbanken) wird mit Hilfe geeigneter Anfragesprachen (z.B. mittels SQL [ANSI92] und OQL [CBB+97]) bewältigt. Die Informationssuche (Information Retrieval) in semistrukturierten XML–Dokumenten stellt neue Anforderungen an die Anfragesprachen.

XML–Dokumente sind durch die flexible Kombination von Struktur– und Inhaltsdaten gekennzeichnet. Sie werden typischerweise durch einen gerichteten *XML–Graphen* repräsentiert, dessen Knoten die Namen von Elementen und Attributen beinhalten und dessen Blätter mit den zugehörigen Werten beschriftet sind. Die Elementhierarchie, die Attributzugehörigkeit und die Verknüpfungen von Elementen werden durch entsprechende gerichtete Kanten ausgedrückt.

Die Hauptaufgabe einer *XML–Anfragesprache* besteht damit in der Adressierung und im Filtern von Knoten, Pfaden und Teilgraphen des XML–Graphen mittels geeigneter Suchbedingungen an die vorliegenden Struktur– und Inhaltsdaten. Unabhängig von der konkreten Notation bieten alle XML–Anfragesprachen die folgenden wichtigsten Struktur– und Inhaltsbedingungen an. Als *Strukturbedingungen* sind elementare Bedingungen an die Namen inklusive lexikalischer Wildcardzeichen, Knotenvariablen und Pfadausdrücke erlaubt. Ein Pfadausdruck beschreibt dabei die Eltern/Kind– bzw. Vorgänger/Nachfolgerbeziehung von gesuchten Knoten im XML–Graphen. *Inhaltsbedingungen* streben den Vergleich von Element– und Attributwerten mit gegebenen logisch verknüpften Konstanten oder anderen Werten durch Knotenvariablen an.

Die Unterschiede zwischen den einzelnen XML–Anfragesprachen ergeben sich aus der konkreten Semantik, die sich in der Art des unterstützten *Information Retrievals* (IR) niederschlägt.

Klassisches *Boolesches Information Retrieval* [BR99] in XML–Dokumenten produziert zu einer gegebenen Anfrage bestehend aus Struktur– und Inhaltsbedingungen eine (unsortierte) Menge von relevanten Dokumenten bzw. Elementen. Dabei ist ein Dokument/Element relevant, also zutreffend, wenn es den gegebenen Suchbedingungen genügt; andernfalls ist es nicht relevant. Die folgenden Sprachen stellen die wichtigsten Struktur– und Inhaltsbedingungen zur Suche nach relevanten Knoten, Pfaden und Teilgraphen im XML–Graphen zur Verfügung und unterstützen Boolesches Information Retrieval. Die Unterschiede bei diesen Sprachen liegen vor allem in der konkreten Syntax und dem tatsächlichen Funktionsumfang.

XPath [BBC+03] ist als Sprache zur Adressierung von Knoten und Pfaden in XML–Dokumenten des XML–Graphen konzipiert. Mit Hilfe eines XSL–Stylesheets [Ber03] wird die Darstellung einer Klasse von XML–Dokumenten spezifiziert, in dem eine Instanz dieser Klasse in ein XML–Dokument transformiert wird, das XSL–basierte Formatierungsvorgaben enthält. Diese Transformation wird mit Hilfe der Sprache XSLT [Kay03] vorgenommen, die XPath verwendet. Lorel [AQM+97] wurde zur Informationssuche in semistrukturierten Daten entwickelt und orientiert sich grundlegend in seinem Anfrageaufbau an SQL und OQL. YATL [CS00] dient der Informationssuche in XML–Dokumenten auf der Basis einfacher Struktur– und Inhaltsbedingungen. Quilt [CRF00] kombiniert grundlegende häufig genutzte Eigenschaften anderer Anfragesprachen (z.B. SQL, OQL, Lorel, YATL, XQL) in einer eigenen Anfragesprache zur Informationssuche in XML–Dokumenten. XQL [RLS98] ist eine XML–Anfragesprache, die aus der Erweiterung von XSL um Namensräume, Aggregationen, Bereichsangaben, etc. hervorgegangen ist. Die Anfragesprache XML–QL [DFF+98] orientiert sich an der

Funktionalität von XPath und in ihrer Syntax an der XML–Spezifikation. XML–GL [CCD+98] ist eine grafische XML–Anfragesprache, bei der die Suchbedingungen mit Hilfe grafischer Mittel formuliert werden. EquiX [CKK+02] verfolgt den Ansatz "Query by Example", bei dem eine Anfrage beispielhaft die Ergebnismenge beschreibt und dann mit den gegebenen XML–Dokumenten verglichen wird. XQuery [BCF+03] ist eine XML–Anfragesprache, deren Anfragen aus logisch verknüpften XPath–Ausdrücken bestehen. In [JLS+04] stellen die Autoren ein XML–Datenmodell auf der Basis von Vielfarbenbäumen (engl. multi–colored trees) zur Kennzeichnung gleicher Elemente (Knoten) in verschiedenen XML–Dokumenten vor. Zusätzlich zeigen sie die Verwendung dieses Modells zur Auswertung von Anfragen in XQuery, womit die Suche nach Pfaden über die Dokumentgrenzen hinweg möglich wird.

Insbesondere sind XPath und XQuery XML–Anfragesprachen, die vom W3C als Standard empfohlen werden.

Der Umgang mit einer sehr großen Menge von XML–Dokumenten fördert das Interesse auch an potentiell relevanten Dokumenten bzw. Elementen. *Ranglistenbasiertes Information Retrieval* [BR99] in XML–Dokumenten produziert zu einer gegebenen Anfrage bestehend aus Struktur– und Inhaltsbedingungen eine nach ihrer Relevanz absteigend sortierte Liste von relevanten Dokumenten bzw. Elementen. Dabei ergibt sich die Relevanz eines Treffers aus dem Grad der Übereinstimmung mit den vorgegebenen Suchbedingungen.

Für die Relevanzbewertung von Dokumenten bzw. Elementen gibt es verschiedene Ansätze. Häufig wird das Vektorraummodell [BR99] oder ein probabilistisches Modell [BR99] verwendet. Im *Vektorraummodell* (engl.: vector space model) werden Dokument/Element und Anfrage durch Vektoren von Termen (Stichwörtern) repräsentiert. Die Relevanz eines Dokuments/Elements ergibt sich dann aus der Distanz (inhaltlichen Übereinstimmung) beider Vektoren. Typischer Weise werden hierzu statistische Daten über die Häufigkeit der gegebenen Terme im Dokument/Element (Termfrequenz TF) und die inverse Häufigkeit der Terme in allen vorliegenden Dokumenten/Elementen (inverse Dokumentfrequenz IDF bzw. inverse Elementfrequenz IEF) herangezogen. Mit Hilfe eines geeigneten Distanzmaßes (z.B. Kosinusmaß) wird dann die Relevanz, also die inhaltliche Ähnlichkeit, bewertet. Bei *probabilistischen Modellen* wird die Relevanz eines Dokuments/Elements durch die Wahrscheinlichkeit ausgedrückt, dass das Dokument/Element tatsächlich als Ergebnis von Interesse ist.

Die folgenden Sprachen stellen die wichtigsten Struktur– und Inhaltsbedingungen zur Suche nach relevanten Knoten, Pfaden und Teilgraphen im XML–Graphen zur Verfügung und unterstützen ranglistenbasiertes Information Retrieval.

Bei der ersten Gruppe von Sprachen steht die inhaltsorientierte Relevanzbewertung von Treffern auf der Basis des Vektorraummodells (kurz: VRM) im Vordergrund, wobei hier nicht mehr nur ganze Dokumente, sondern vor allem einzelne Elemente und ihre Elementwerte betrachtet werden. Die gegebenen Strukturbedingungen müssen von den Treffern genau erfüllt werden. Die Unterschiede bei diesen Sprachen liegen vor allem in der konkreten Anwendung des VRMs.

XRANK [GSBS03] unterstützt die VRM–basierte Stichwortsuche in den Inhaltsdaten von XML–Dokumenten, wobei der Abstand zwischen den einzelnen Stichwörtern und die Vorgänger/Nachfolger–Distanz zwischen den zugehörigen Elementen in die Relevanzbewertung einfließt. Die Sprache ELIXIR [CK02] basiert auf einer Erweiterung von XML–QL um inhaltsorientierte Relevanzbewertungsverfahren auf der Basis des VRMs, wobei zur Relevanzbewertung WHIRL verwendet wird. WHIRL (word–based heterogeneous information representation language) [Coh00] ist für die relevanzbasierte

Stichwortsuche in beliebigen, unstrukturierten Texten auf der Basis des VRMs konzipiert. XSEarch [CMKS03] bietet die Möglichkeit, durch eine Liste von Stichwörtern nach relevanten Elementen und relevanten Elementwerten auf der Basis des VRMs zu suchen. TeXQuery [ABS04] ist eine Erweiterung von XQuery um umfangreiche stichwortbasierte Volltextsuchmöglichkeiten in den Inhaltsdaten von XML–Dokumenten. XIRQL [FG01] ist eine Erweiterung von XQL um Methoden zur inhaltsorientierten Relevanzbewertung von Treffern, wobei Verfahren von logikbasierten IR–Modellen und Konzepte aus dem Datenbankbereich verwendet werden.

Die zweite Gruppe von Sprachen berücksichtigt neben der inhaltsorientierten Relevanzbewertung von Treffern auf der Basis des VRMs auch Abweichungen in der Struktur der Treffer.

ApproXQL [SM02, Sch01] unterstützt zum einen die inhaltsorientierte Relevanzbewertung auf der Basis des VRMs und zum anderen die strukturorientierte Relevanzbewertung von Treffern auf der Basis der Baumeinbettung (engl.: tree matching). Die Autoren von [ACS02] beschreiben die Informationssuche in XML–Daten auf der Basis von Baummustererweiterungen (engl.: tree pattern relaxations), wobei sowohl strukturelle als auch inhaltsbasierte Erweiterungen unterstützt werden und in der Berechnung der relevanzbasierten Rangliste einfliessen. SearX [Flo03] erlaubt im Vorfeld die Definition von Rollen, wobei eine Rolle Elementnamen gleicher Semantik zusammenfasst. In [GJK+02] wird die Ähnlichkeit von Struktur– und Inhaltsbedingungen mit gegebenen XML–Dokumenten auf der Basis von Distanzmaßen (z.B. Tree–Edit–Distance) bewertet.

Fazit: Im Unterschied zu den oben genannten Ansätzen kombiniert XXL die struktur– und inhaltsorientierte Relevanzbewertung von Treffern zur Berechnung einer Rangliste von relevanten Ergebnissen. Dabei wird die Relevanz von Elementwerten auf der Basis ontologiebasierte, semantischer Ähnlichkeit und dem VRM und die Relevanz von Elementnamen auf der Basis ontologiebasierter, semantischer Ähnlichkeit ermittelt.

1.2.2 Indizierung von XML–Dokumenten

Indexstrukturen sind spezielle Datenstrukturen, um die Antwortzeiten eines Datenbanksystems zu verbessern [RV03, KM03, BR99, CLR90, WMB99]. Neue Anwendungen wie der Umgang mit semistrukturierten XML–Dokumenten stellen neue Anforderungen, die von traditionellen Indexstrukturen nicht erfüllt werden. Der Entwurf von Indexstrukturen zur Indizierung von XML–Dokumenten wird von zwei Zielen geleitet.

1. Zur effizienten Auswertung von inhaltsorientierten Suchbedingungen sollen Elementwerte, Attributwerte und Dokumentinhalte durch einen geeigneten *Inhaltindex* aufbereitet werden.
2. Zur effizienten Auswertung von strukturorientierten Suchbedingungen soll die Struktur von XML–Dokumenten bzw. vom XML–Graphen bedarfsabhängig durch einen geeigneten *Strukturindex* aufbereitet werden.

Die Indizierung von unstrukturierten Texten ist ein bekanntes Problem, zu dem es eine Vielzahl guter Lösungsansätze gibt [BR99, CLR90, WMB99], z.B. Invertierte Listen, Signaturen, Tries, Suffixbäume.

Im Gegensatz dazu stellt die Indizierung der Strukturdaten von XML–Dokumenten neue Anforderungen an die Aufbereitung dieser Daten in geeigneten Indexstrukturen. Publizierte Arbeiten in diesem Bereich unterscheiden sich durch das zugrunde liegende Datenmodell und den Verwendungszweck der Indizes. Bei den betrachteten Datenmodellen werden entweder XML–Dokumente durch XML–Bäume repräsentiert oder eine Menge von XML–Dokumenten durch einen gerichteten XML–Graphen. Die Strukturindizes werden wahlweise zur Abbildung der vollständigen Strukturdaten, zur Abbildung von Pfaden oder zur Abbildung von Verbindungsinformationen konzipiert.

Der *vollständige Strukturindex* dient zur Aufbereitung der gesamten Strukturdaten eines XML–Dokuments bzw. einer Menge von XML–Dokumenten, um beliebige Teilstrukturen effizient reproduzieren zu können.

In [Gru02] wird ein vollständiger Strukturindex zur Auswertung von XPath–Anfragen vorgestellt, der auf einer speziellen PrePostOrder–Kodierung der Knoten beruht. Hierbei fließen in die Kodierung die PreOrder– und PostOrder–Nummern der Knoten und die ihrer Eltern ein. In [ZAR03] werden Baumsignaturen verwendet, um häufig wiederkehrende Teilstrukturen in XML–Dokumenten zu kodieren. In [LM01] stellen die Autoren ein Knotenmarkierungsverfahren vor, das zum einen auf der PrePostOrder–Nummerierung basiert und zum anderen als weiteren Parameter die Anzahl der Nachfolger vermerkt. In [CKM02, KMS02] werden die Eigenschaften von PrePostOrder–Knotenmarkierungen und ähnlichen Knotenmarkierungsverfahren diskutiert. Die hier genannten Arbeiten stellen vollständige Strukturindizes für XML–Daten vor, sind aber nur auf XML–Bäume, nicht auf XML–Graphen anwendbar.

Der *Pfadindex* fasst bestimmte Pfade aus XML–Dokumenten in einer geeigneten Datenstruktur zusammen, um diese effizient wiederzufinden.

DataGuides [GW97] stellen eine kompakte, akkurate Zusammenfassung der Strukturdaten von XML–Dokumenten bereit. Dabei werden XML–Dokumente gleicher Struktur durch einen gemeinsamen DataGuide repräsentiert. Index Fabric [CSF+01] verwendet Patricia Tries, um Pfade beginnend bei der Wurzel des zugehörigen XML–Dokuments als String zu speichern und effizient durch stringbasierte Lookups wiederzufinden. Der T–Index (Template Index) [MS99] repräsentiert gleiche Pfade durch einen entsprechendes Pfad–Template, das den genauen Aufbau der Pfade beginnend bei der Dokumentwurzel widerspiegelt. Hierbei wird immer nur ein Teil der vorkommenden Pfade indiziert. APEX [CMS02] fasst häufig angefragte Pfade kompakt zusammen, indem gleiche Namen identifiziert und zusammengefasst werden. Dazu werden Data–Mining–Algorithmen angewandt. Das Index Definition Scheme [KBNK02] basiert auf der "Bisimilarity" von Knoten, d.h. Knoten mit gleichartigen ein– und ausgehenden Pfaden werden zusammengefasst. Dieses Index–Schema kann zur Definition spezieller Indexstrukturen verwendet werden, z.B. 1–Index, A(k)–Index, D(k)–Index [CLO03], F&B–Index (Forward– and Backward–Index), wobei k die maximale Länge der unterstützten Pfade angibt. In [BGK03] fassen die Autoren wiederkehrende Teilbäume in XML–Dokumenten zu einem Baumskelett zusammen, in dem Knoten mit gleichen Namen vereint werden und die bestehenden Kanten nur noch einmal auftreten. In [CMPB04] stellen die Autoren ein Verfahren vor, bei dem die geordneten Elementnamen von XML–Dokumente in Intervalle gleicher maximaler Größe geteilt werden. Die dabei entstehenden Teilbäume werden in balancierte L–Bäume abgelegt, bei denen die Elementnamen in den Blättern stehen. In [KPS04] wird ein XML–Dokument im mehrdimensionalen Raum durch eine Menge von Punkten repräsentiert. Für die Indizierung solcher Daten stellen die Autoren einen BUB–Baum zur Abbildung des mehrdimensionalen Raums in einen eindimensionalen Raum durch Verwendung von ganzzahliger

Nummerierung auf der Basis der Raumkoordinaten vor. In [ESA+04] erhalten XML–Dokumente eine Pre/Postorder–Numerierung. Die Knoten werden dann in einer Liste zusammengefasst und auf der Basis der Knotenmarkierungen in Bereiche geteilt. Die folgenden Ansätze spezialisieren sich auf die Auswertung von Twig–Anfragen, wobei ein Twig ein kleiner knotenmarkierte Baum der Tiefe 1 (Wurzel und Kinder) ist. In [RM04] werden XML–Dokumente durch Prüfer–Sequenzen repräsentiert, wobei eine Prüfer–Sequenz eine eindeutige Abbildung von einem Baum auf eine Knotenfolge ist. In [JWLY03] werden XML–Dokumente mit Hilfe eines XR–Baums auf der Basis einer bereichbasierten Knotenmarkierung indexiert. Für diesen Index geben die Autoren den TSGeneric–Algorithmus zur Suche nach Twig–Mustern an. In ähnlicher Art und Weise werden in [BKS02] XML–Dokumente mit Hilfe eines XB–Baums auf der Basis einer bereichsbasierten Knotennummerierung indexiert, für die die Autoren mit TwigStack einen Algorithmus zur effizienten Berechnung von Twig–Anfragen vorstellen. Die meisten Ansätze in diesem Gebiet können dabei mit beliebigen Graphen umgehen bzw. lassen sich entsprechend erweitern. Sie unterstützen dabei vor allem die Navigation auf dem XML–Graphen über die Eltern–Kind–Beziehungen der Elemente in XML–Dokumenten. Zur Auswertung von langen Pfaden benötigen sie entweder sehr viel Speicher zur Aufbereitung dieser Pfade oder vergleichsweise lange Zugriffszeiten, um solche Pfade zu ermitteln.

Der *Verbindungsindex* dient zur Kodierung der Verbindungsinformationen, um die transitive Verbindung beliebiger Knoten in XML–Dokumenten effizient zu testen.

Die erste Gruppe von Arbeiten zielt auf die effiziente Auswertung von Anfragen nach den Vorgängern eines gegebenen Knotens in einem gegebenen XML–Dokument ab, wobei hier ein XML–Dokument durch einen XML–Baum repräsentiert wird, dessen Knoten geeignet annotiert werden. In [AR02] wird die Pre–/Post–Order–Kodierung durch den Einsatz spezieller Techniken aus den Bereichen Baumgruppierung (engl.: tree clustering) und alphabetischen Kodierungen verbessert. In [Gru02] wird ein erweitertes Pre/Post–Order–Markierungsschema vorgestellt, das zusätzlich Informationen zu den Eltern einer Knotens beinhaltet. In [AKM01, KM01] beschreiben die Autoren ein Baummarkierungsschema, das den Baum in Partitionen teilt und rekursiv die Labels mit Hilfe des "prune&contract"–Algorithmus berechnet. In [KMS02] wird ein präfix–basiertes Knotenmarkierungsverfahren vorgestellt. In [CKM02] beschreiben die Autoren ein dynamisches Knotenmarkierungsverfahren auf der Basis von Clues für XML–Bäume.

Die zweite Gruppe von Arbeiten befasst sich mit der Kodierung der Verbindungsinformationen in beliebigen XML–Graphen, nicht nur in XML–Bäumen. Eine platzsparend Kodierung der Verbindungsinformationen eines Graphen (also seiner transitiven Hülle) wird in [CHKZ02] vorgestellt. Diese 2Hop–Kodierung ist in ihrer Originalform für die Verwendung zur Kodierung von großen Dokumentmengen ungeeignet. Ein Algorithmus, der den gesamten Graphen zunächst partitioniert und die 2Hop–Kodierungen der Teilgraphen am Ende zusammensetzt, wird in [STW04] diskutiert und ist für beliebig große XML–Graphen einsetzbar.

Fazit: Für die effiziente Kodierung von Eltern–Kind–Beziehungen in XML–Dokumenten gibt es zahlreiche ausreichend gute Ansätze aus der Graphentheorie. Die neue Herausforderung bei der Indizierung von unter Umständen sehr großen XML–Graphen besteht in der effizienten Speicherung beliebig langer Pfade. Die XXL–Suchmaschine stellt einen Elementpfadindex zur Verfügung, der auf der Basis einer Pre–/Post–Order–Kodierung und einer 2Hop–Kodierung beliebig kurze und lange Pfade effizient auswerten kann.

1.2.3 Ontologien

Ursprünglich wurden logikbasierte Ontologien von Forschern der Künstlichen Intelligenz (engl.: artificial intelligence) entwickelt, um die gemeinsame Nutzung und Wiederverwendung von gesammelten Wissen zu erleichtern. Das Interesse an Ontologien ist in den letzten Jahren im Zusammenhang mit Diskussionen über das "Semantische Web" [SemWeb, MWK00, SAD+00, FHLW03] wieder aufgelebt.

Im Gegensatz zu den ambitionierten Vorhaben der KI, universelle Ontologien zu bauen [LG90, RN95], zielen aktuelle Vorschläge auf gebietsspezifische oder nutzerspezifische Ontologien mit leichter handhabbaren Logiken ab [Hor02, SAD+00]. Aktuelle Veröffentlichungen zur Formalisierung von Ontologien sind weitreichend, beginnend bei algebraischen Ansätzen [BM99], über logikbasierten Sprachen zur Modellierung von Ontologien [DEFS99, BEH+01, HPH03, Hef04, Hor02, CHH+01], bis hin zu konzeptbasierter Datenmodellierung und Dateninterpretation [Gua98, NFM00, SM00]. Im Zusammenhang mit dem Information Retrieval in Webdokumenten gibt es Vorschläge zur Definition von Ontologien auf der Basis von Klassen, Instanzen und Eigenschaften von Klassen mit Hilfe der logikbasierten Annotationssprache OWL [HPH03, Hef04] bzw. DAML+OIL [Hor02, CHH+01] zur Annotation von Webdokumenten. Ähnliche Ansätze wurden im Kontext von Multidatenbanken verfolgt. Z.B. wird in [BHP94] ein Zusammenfassungsschema modelliert, das seine semantische Leistungsstärke vom linguistischen Wissen einer Online–Taxonomie erhält. In [PM98] präsentieren die Autoren eine konzeptbasierte Organisation der Informationen. In diesen Ansätzen wird die Quantifizierung von Begriffsbeziehungen nicht berücksichtigt.

Es gibt verhältnismäßig wenige Arbeiten, [KKS01, KKC94, KI90, STW01, BHP94], die gewichtete Ontologien beschreiben. Verschiedene Ansätze zur Quantifizierung der Ähnlichkeit von Begriffen für das bestehende Thesaurus WordNet auf der Basis der Distanzen werden in [BH01] erläutert. Diese Ansätze definieren oftmals einfache Gewichte für Begriffsähnlichkeiten, die zur Anfrageauswertung ungeeignet sind.

Die automatische oder halbautomatische Ontologiekonstruktion ist Gegenstand der Betrachtungen in [MS00, KKS01, KKC94, KI90, SBB+99] und basiert mehrheitlich auf Verfahren des Text–Minings und der Informationsextraktion mittels Verarbeitung der natürlichen Sprache (engl.: natural language processing) unter Verwendung bestehender Thesauri oder Textprozessoren wie z.B. SMES [MS00] oder GATE [CMBT02]. Zahlreiche Forschungsgruppen entwickeln umfangreiche Systeme zur Verwaltung einzelner oder mehrerer Ontologien. Dabei steht der Aufbau einer Ontologie durch die Zusammenfassung verschiedener Ontologien [BM99, LSN+00] häufig im Mittelpunkt. Andere umfangreiche Systeme dienen der Entwicklung und Verwendung eigener Ontologien, z.B. OntoBroker [DEFS99], Text-To-Onto [MS00], GETESS [SBB+99], Protg 2000 [NFM00], LGAccess [AL02], KAON [BEH+01], Ontolingua [FFR96], FrameNet [BFL98], Thesus [HNVV03].

Fazit: Unserer Meinung nach, wurde der Einsatz von Ontologien zur Unterstützung

der Informationssuche in semistrukturierten Daten, z.B. in XML–Dokumenten, bisher nicht diskutiert. Die Besonderheit der XXL–Suchmaschine liegt in der Kombination von ontologischem Wissen und Information–Retrieval–Techniken zur semantischen Ähnlichkeitssuche in den Struktur– und Inhaltsdaten von XML–Dokumenten.

1.2.4 Benchmarks

Eine wichtige Aufgabe bei der Entwicklung von Systemen und Verfahren zum effektiven und effizienten Umgang mit XML–Dokumenten besteht in der Evaluation dieser Verfahren und vor allem im Vergleich mit anderen Verfahren. Zum Vergleich von Systemen und Verfahren bieten sich *Benchmarktests* [Gray91, Lan92] an. Da Verfahren und Systeme im Allgemeinen für ein spezielles Aufgabenfeld konzipiert sind und einzelne Metriken nicht ausreichen, die Leistung aller möglichen Anwendungsfälle zu berücksichtigen, werden typischerweise *aufgabenspezifische Benchmarktests* verwendet. Zur Durchführung eines Benchmarktests gehören folgende Schritte.

1. Zunächst muss das Ziel des Benchmarktests definiert werden. Dazu gehören genaue Vorgaben über Messobjekte und Messgrößen, die Aussagen liefern oder miteinander verglichen werden sollen.

2. Die Zusammenstellung des Benchmarktests umfasst die Beschreibung und Festlegung der Rahmenbedingungen. Hierbei werden zum Beispiel die verwendeten Daten und die durchzuführenden Aufgaben spezifiziert.

3. Zur eigentlichen Durchführung eines Benchmarktestlaufs gehört primär die Erfassung der Messergebnisse.

4. Die abschließende Analyse aller Ergebnisse erfolgt durch geeignete Bewertungsmethoden, so dass die Leistung der verschiedenen Systeme und Verfahren vergleichbar ist.

Dabei sollte ein Benchmarktest die folgenden vier wichtigen Kriterien erfüllen:

1. Relevanz: Die Zielsetzung und die Zusammenstellung des Benchmarktests müssen für die zu untersuchenden Leistungsparameter relevant sein.

2. Portabilität: Es sollte einfach sein, den Benchmarktest in verschiedenen Umgebungen zu implementieren

3. Skalierbarkeit: Der Benchmarktest sollte von kleinen und großen Systemen/Verfahren durchführbar sein.

4. Einfachheit: Der Benchmarktest muss verständlich sein.

Um die Effektivität und die Effizienz von Systemen und Verfahren im Umgang mit XML–Dokumenten zu bewerten, werden verschiedene XML–Benchmarks vorgeschlagen, auf die wir an dieser Stelle kurz eingehen möchten.

Die erste Gruppe von XML–Benchmarks befasst sich mit der Laufzeit– und Speicherplatz–Analyse von XML–Datenbanksystemen bzgl. der Speicherung von XML–Dokumenten und Auswertung von XML–Anfragen.

XMach–1 [BR01, BR02] dient dem Test von XML–Datenbanksystemen im Mehrbenutzerbetrieb und basiert auf einer Webanwendung für XML–Dokumente mit dem Ziel, typische Anwendungsbeispiele für XMLDBMS (XML Data Management System) zu modellieren. Der Benchmark umfasst eine XML–Datenbank, Applikationsserver und Clients. Die Daten bestehen maximal aus 1000 XML–Dokumenten, die unterschiedlich stark strukturiert sind. Die Arbeitslast (workload) umfasst acht Anfrageoperatio-

nen und drei Updateoperationen in Form von entsprechenden XQueries, die ein weites Spektrum der Anfrageverarbeitungsaufgaben abdecken, z.B. Anfragen stark strukturierter Dokumente, Volltextsuche in XML–Dokumenten/Elementen, Navigationsanfragen, Anfragen mit Sortieren/Gruppieren/Aggregation etc. Dabei wird die erfolgreiche Auswertung von XML–Anfragen pro Sekunde gemessen (Xqps).

XMark [SWK+02] unterstützt den Test von XMLDBMS im Einzelbenutzerbetrieb und basiert auf einem Internetauktionsszenario. Es werden 20 XQueries vorgegeben, um Standard– und Nichtstandardfunktionen der XML–basierten Anfrageauswertung zu testen. Hierbei steht die Laufzeit von Auswahlanfragen, Ausgabekonstruktion und Restrukturierung, Volltextsuche, strukturbasierte Suche, Funktionen wie Sortierung und Aggregation im Vordergrund. Updateoperationen werden nicht untersucht.

Der *XOO7–Benchmark* [NLB+01, NLB+02, BLL+02] geht aus dem OO7–Benchmark für objektorientierte Datenbanksysteme hervor und ist darauf ausgelegt, die Ähnlichkeit des semistrukturierten XML–Datenmodells (XML–Graph) und der objektorientierten Ansätze auszunutzen. Hierbei wird kein spezifisches Anwendungsszenario durch die verwendeten XML–Dokumente vorgegeben, die bedarfsabhängig generiert werden. Die Arbeitslast besteht aus 23 XQueries mit dem Ziel, Auswahlanfragen (strukturbasierte Anfragen, inhaltsbasierte Anfragen) und Navigationsanfragen zu testen.

Der *Michigan–Benchmark* [RPJA02, RPJA02a] ist ein Mikrobenchmark, der die Laufzeit spezieller Teilaspekte der Anfrageverarbeitung in XMLDBMS bewerten, z.B. Selektion (inhaltsorientiert, strukturbasiert), Joins, Aggregation. Auch gibt es kein fest vorgegebenes Anwendungsszenario, die XML–Dokumente werden automatisch mit unterschiedlichen Schachtelungstiefen generiert.

XBench [YOK02, YOK04] umfasst eine Familie von XML–Benchmarks, mit dem Ziel, die Laufzeit der XQuery–Auswertung in sehr großem Umfang in XMLDBMS zu testen. Dabei wird der Datenbestand aus Online–Dictionaries bzw. E–Commerce–Katalogen generiert.

Die zweite Gruppe von XML–Benchmarks befasst sich mit der inhaltlichen Qualität der Ergebnisse, die Laufzeit bei der Anfrageauswertung spielt nur eine untergeordnete Rolle.

Der *INEX–Benchmark* [INEX, INEX03] stellt mehr als 12.000 wissenschaftliche Artikel zur Verfügung, in denen inhaltsorientiert und strukturorientiert gezielt nach Informationen gesucht werden soll. Die Ergebnisse werden hinsichtlich ihrer Präzision (precision) und ihrer Ausbeute (recall) bewertet.

Fazit: Gemäß dem Schwerpunkt dieser Arbeit ist der INEX–Benchmark gegenwärtig die einzige Möglichkeit, die Qualität der XXL–Anfrageergebnisse hinsichtlich ihrer Präzision und ihrer Ausbeute zu evaluieren und mit anderen XML–basierten Information Retrieval Verfahren zu vergleichen.

1.3 Beitrag und Gliederung dieser Arbeit

Information Retrieval (kurz: IR) befasst sich mit der Repräsentation, Speicherung, Organisation und dem Zugang zu Informationseinheiten [GRGK97, BR99] mit dem Ziel, einem Benutzer möglichst viele, hochgradig relevante Informationseinheiten zu liefern, ohne das dieser dafür auch viele nutzlose Informationseinheiten durchgehen muss. Ein zentraler Begriff ist dabei die Relevanzbewertung von Informationseinheiten bzgl. einer gegebenen Anfrage.

Die *Suche nach Daten und Fakten* beruht auf der Auswertung von Anfragen in einer (künstlichen) klar definierten Sprache. Dabei wird in wohlstrukturierten Daten mit klarer Syntax und Semantik gesucht und als Ergebnis eine Menge von Antworten zurückgegeben, die exakt zutreffen.

Im Gegensatz dazu sind für die *Suche nach Informationen* vage Anfragen auf der Basis der natürlichen Sprache charakteristisch. Dabei wird in Daten gesucht, die sehr unterschiedlich stark strukturiert sind und semantische Mehrdeutigkeiten beinhalten können. Als Ergebnis wird eine Menge von Antworten zurückgegeben, die mehr oder weniger zutreffen.

Im Rahmen dieser Arbeit entwickeln wir ein *IR–System*, das die effektive *Informationssuche in XML–Dokumenten des Webs* im Sinne hoher Ergebnisqualität und kurzer Antwortzeiten unterstützt. Dabei befassen wir uns ausführlich mit den folgenden Aspekten.

- Die Informationssuche wird entscheidend durch die Granularität der Informationseinheiten beeinflusst. Das Fundament unseres IR–Systems bildet die Definition eines abstrakten Datenmodells, des *XML–Graphen*, zur Repräsentation von XML–Dokumenten aus dem Web.
- Um die Relevanz von Informationseinheiten auf der Basis ihrer Semantik bewerten zu können, ist Wissen über die Bedeutung von Wörtern sowie die semantischen Beziehungen zwischen Wörtern auf der Basis ihrer Bedeutungen erforderlich. Dieses Wissen stellen wir mit Hilfe einer *Ontologie* bereit.
- Die Vorgehensweise bei der Informationssuche wird durch die Motivation des Benutzers dominiert. Die Suche ohne genaue Zielvorstellung wird als Blättern oder Browsing bezeichnet. Zur gezielten Suche im XML–Graphen wird eine Anfragesprache zur geeigneten Formulierung von Suchbedingungen benötigt. In unserem IR–System werden Anfragen mit Hilfe der *XML–Anfragesprache XXL* (flexible XML search language) gestellt.
- Die Qualität der Ergebnisse der Informationssuche hängen entscheidend von der Art der Relevanzbewertung der vorliegenden Informationseinheiten des XML–Graphen ab. In unserem IR–System beruht die Semantik einer XXL–Anfrage auf der ontologiebasierten Relevanzbewertung der Knoten des XML–Graphen.
- Kurze Antwortzeiten im Rahmen der Möglichkeiten dürfen trotz hohem Anspruch an die Relevanz der Ergebnisse nicht vernachlässigt werden. Die Verarbeitung von XXL–Anfragen wird in unserem System durch die Verwendung des Elementpfadindizes und des Elementinhaltindizes unterstützt.

Die hier entwickelten Konzepte werden prototypisch in einer XML–Suchmaschine implementiert. Diese Suchmaschine wird gemäß der unterstützten XML–Anfragesprache als XXL–Suchmaschine bezeichnet.

Diese Arbeit unterscheidet sich von den anderen Arbeiten auf dem Gebiet des Information Retrieval in XML–Dokumenten (z.B. XQL [RLS98], XML–QL [DFF+98], Quilt [CRF00], YATL [CS00], XIRQL [FG01], XSL [Ber03], XPath [BBC+03], XQuery [BCF+03]) durch die neuartige Berechnung der Rangliste von Ergebnissen bestehend aus relevanten XML–Dokumenten bzw. relevanten Ausschnitten von XML–Dokumenten mit Hilfe von geeigneten XML–Indexstrukturen und Ontologien. Die Relevanz von Ergebnissen und damit die Rangfolge dieser Ergebnisse beruht auf der semantischen, ontologiebasierten Ähnlichkeit der Ergebnisse zur gegebenen Anfrage. Dabei wird die semantische Ähnlichkeit zu den Element– und Attributnamen (Strukturdaten), die semantische Ähnlichkeit zu Element– und Attributwerten (Inhaltsdaten)

sowie die Verteilung der gesuchten Information über verschiedene XML–Dokumente berücksichtigt.

Die hier vorliegende Arbeit ist wie folgt aufgebaut.

Kapitel 2: Das zweite Kapitel erläutert die Syntax von XML–Dokumenten. Dabei liegt der Hauptaugenmerk auf der Syntax von Elementen und Attributen sowie der Strukturierung der XML–Dokumente durch die Anordnung von Elementen bzw. durch Links zwischen Elementen. Zur Repräsentation einer beliebigen Menge von XML–Dokumenten wird der XML–Graph definiert, der die Namen und Werte in seinen Knoten ablegt und die Hierarchie bzw. Links durch entsprechende gerichtete Kanten symbolisiert.

Kapitel 3: Im Kapitel 3 erläutern wir die Zusammenhänge von Wort/Bedeutung/Begriff und Daten/Wissen/Informationen. Dabei fassen wir Begriffe gleicher Bedeutung zu einem Konzept zusammen und verwenden dieses als Grundbaustein einer Ontologie, die vielfältige semantische Beziehungen zwischen Begriffen auf der Basis ihrer Bedeutungen widerspiegelt.

Kapitel 4: Das vierte Kapitel beginnen wir mit der Definition der Syntax der XML–Anfragesprache XXL (flexible XML search language). Eine XXL–Anfrage liegt dabei initial als Textstring vor und muss zur Auswertung in eine geeignete graphbasierte Darstellung umgewandelt werden. Der Mittelteil diese Kapitels definiert diese Transformation. Im Abschluß des vierten Kapitels wird die Semantik einer graphbasierten XXL–Anfrage bzgl. eines gegebenen XML–Graphen angegeben. Zur Relevanzbewertung von Teilergebnissen und Ergebnissen werden die verfügbare Ontologie und Standardverfahren aus dem Information Retrieval verwendet.

Kapitel 5: Das Kapitel 5 beschreibt die algorithmische Verarbeitung von XXL–Anfragen auf der Basis der im Kapitel 4 definierten Syntax und Semantik.

Kapitel 6: Das sechste Kapitel befasst sich mit der Abbildung der Informationseinheiten des XML–Graphen in geeignete Indexstrukturen, die die Verarbeitung von XXL–Anfragen beschleunigen sollen. Dazu wird zunächst auf der Basis von Kapitel 5 analysiert, in welcher Art und Weise auf dem XML–Graphen navigiert werden können muss. Zur Unterstützung dieser Navigationen führen wir den Elementpfadindex zur Abbildung der Struktur und den Elementinhaltindex zur Abbildung der Inhalte von XML–Dokumenten ein. Das Kapitel schließen wir mit der Erläuterung ab, wie die Indexstrukturen in die Verarbeitung von XXL–Anfragen integriert werden.

Kapitel 7: In diesem Kapitel werden die Besonderheiten der Implementierung hervorgehoben. Dabei wird ein besonderes Augenmerk auf die Relevanz von Dokumenten sowie die Präzision und Ausbeute von Suchergebnissen gelegt.

Kapitel 8: Das achte Kapitel ist der Evaluation des entwickelten Prototyps gewidmet. Im Rahmen von zwei Experimenten wird die Effektivität und die Effizienz der XXL–Suchmaschine demonstriert.

Spezielle Aspekte dieser Arbeit wurden auf wissenschaftlichen Konferenzen präsentiert. [TW00, TW00a] geben eine Einführung in die XML–Anfragesprache XXL und deren spezielles Merkmal, den Ähnlichkeitsoperator, so dass ranglistenbasiertes Information Retrieval (Ranked Retrieval) auf XML–Daten möglich wird. In den Arbeiten [STW01, STW01a, The03, STW03] werden frühere Versionen des Konzepts der ontologiebasierten Ähnlichkeit entwickelt. Die indexbasierte Auswertung von Ähnlichkeitsanfragen auf XML–Daten wird in den Arbeiten [TW02, TW02a, STW04] diskutiert.

Kapitel 2

Grundlagen

Das World–Wide–Web–Konsortium (kurz: W3C) befasst sich mit Empfehlungen und Verabschiedungen von Standards, z.B. Spezifikationen von XML–Technologien (XML, DTD, XLink, XPointer, XPath, u.a.). Auf der Basis dieser Technologien entstehen *XML–Dokumente*, die vor allem zur Verbreitung und zum Austausch von Daten in einem einheitlichen Datenformat über das Internet bzw. Intranets verwendet werden.

Das Fundament unseres IR–Systems bildet die Definition eines abstrakten Datenmodells zur Repräsentation einer beliebigen Menge von XML–Dokumenten.

Dazu beschreiben wir in Abschnitt 2.1 den grundsätzlichen syntaktischen Aufbau von XML–Dokumenten auf der Basis der Annotationssprache XML. Im Anschluss daran gehen wir auf nützliche, häufig genutzte Erweiterungen durch die Verwendung von Dokumenttyp–Definitionen (Abschnitt 2.2) sowie durch die Verwendung von Links (Abschnitt 2.3) ein. Im letzten Abschnitt 2.4 übertragen wir Struktur– und Inhaltsdaten von einer Menge von XML–Dokumente in einen gerichteten Graphen.

Zur Illustration verwenden wir XML–Dokumente mit Inhalten aus Shakespeare's Dramen, die sich an den XML–Beispielen von J. Bosak [Bos99] orientieren.

2.1 XML–Dokumente

XML (eXtensible Markup Language) [BPS+04] ist eine Sprache, die eine Klasse von Datenobjekten – genannt XML–Dokumente – beschreibt und das Verhalten von Computerprogrammen beeinflussen kann, die diese verarbeiten. Die XML–Spezifikation [BPS+04] beschreibt die Regeln für die syntaktisch korrekte Darstellung von XML–Dokumenten in Textform. Ein XML–Dokument besteht spezifikationsgemäß immer aus den folgenden drei Abschnitten:

`Prolog XmlElement Misc`

Hierbei sind im `Prolog` die notwendigen Deklarationen, Kommentare und Verarbeitungsanweisungen verankert und `Misc` enthält eine beliebige Anzahl von Kommentaren und Verarbeitungsanweisungen. `XmlElement` umfasst die semistrukturierten Daten des XML–Dokuments in Form einer Hierarchie von XML–Elementen beginnend bei einem eindeutigen Wurzelelement (siehe Abb. 2.1).

Beispiel 2.1 In der Abb. 2.1 wird ein Ausschnitt aus dem Shakespeare–Drama "Macbeth" im XML–Format angezeigt.

http://www.arts.edu/poet/shakespeare/macbeth.xml

```
 1 <?xml version="1.0"?>
 2 <!DOCTYPE TRAGEDY SYSTEM „tragedy.dtd">
 3 <TRAGEDY>
 4     <TITLE>The Tragedy of Macbeth</TITLE>
   ...
 5     <ACT>
 6         <TITLE>ACT I</TITLE>
 7         <SCENE SUBTITLE=„SCENE I.  A desert place.“>
 8             <STAGE>Thunder and lightning. Enter three Witches</STAGE>
   ...
 9         <SCENE SUBTITLE=„SCENE III.  A heath near Forres.“>
   ...
10             <STAGE>Enter MACBETH and BANQUO
   ...
11             <ACTION>
12                 <SPEAKER>MACBETH</SPEAKER>
13                 <SPEECH>
14                     <LINE>Speak, if you can: what are you?</LINE>
                   </SPEECH>
               </ACTION>
15             <ACTION>
16                 <SPEAKER>First Witch</SPEAKER>
17                 <SPEECH>
18                     <LINE>All hail, Macbeth!</LINE>
19                     <LINE>hail to thee, thane of Glamis!</LINE>
                   </SPEECH>
               </ACTION>
   ...
20             <ACTION>
21                 <SPEAKER>Third Witch</SPEAKER>
22                 <SPEECH>
23                     <LINE>Thou shalt get kings, though thou be none:</LINE>
24                     <LINE>So all hail, Macbeth and Banquo!</LINE>
                   </SPEECH>
               </ACTION>
   ...
       </ACT>
   ...
   </TRAGEDY>
25 <!--This tragedy is written by William Shakespeare. -->
```

Abbildung 2.1: Syntaktischer Aufbau eines XML–Dokuments

In den Zeilen 1 und 2 sind die Deklarationen angegeben. In der dritten Zeile beginnt das Wurzelelement `TRAGEDY`. In den Zeilen 4 – 24 sind weitere Elemente angegeben, z.B. `TITLE`, `ACT`, `SCENE`. In der letzten Zeile steht ein Kommentar.

Ein *XML–Element* (kurz: Element) ist durch einen *Elementnamen* `EName`, seinen *Elementinhalt* `EInhalt` und einer Menge von Attributen gekennzeichnet. Dabei gehört zu einem Attribut immer ein Attributname `AName` und ein Attributwert `AWert`. In einem textbasierten XML–Dokument wird ein XML–Element dann im Allgemeinen in der Form

```
<EName AName1="AWert1" AName2="AWert2" ...>EInhalt</EName>
```

dargestellt. Der Elementinhalt kann aus beliebigen Inhaltsdaten (den *Elementwerten*) und/oder weiteren XML–Elementen bestehen (daher die Hierarchie). Die spitzen Klammern sind dabei Metazeichen der Sprache und werden im XML–Dokument zur Abgrenzung der sogenannten Elementtags vom übrigen Dokumentinhalt verwendet (siehe Abb. 2.2).

Beispiel 2.2 In der Abbildung 2.2 wird die Syntax von Elementen und Attributen sowie die Benennung der einzelnen Komponenten anhand des Beispielelementes `SCENE` illustriert.

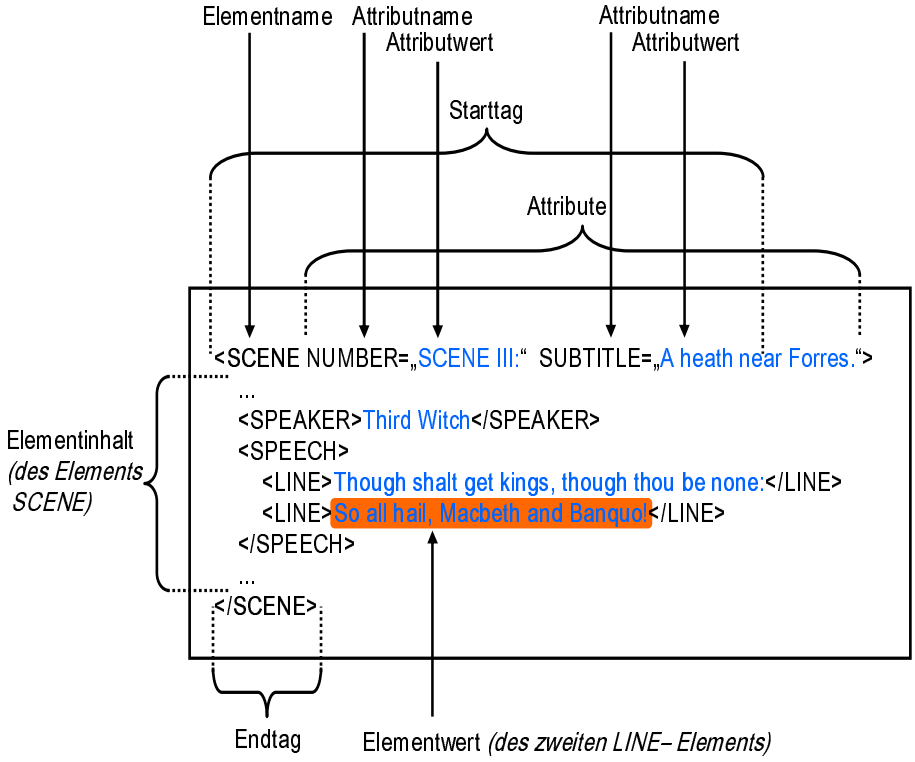


Abbildung 2.2: Syntaktischer Aufbau eines Elements

Die Namensgebung und die Inhaltsdaten unterliegen bestimmten syntaktischen Konventionen, die in der Spezifikation verankert sind. Wir geben die für diese Arbeit wichtigen Spezifikationsregeln an und leiten daraus den Aufbau von XML–Dokumenten in Textform ab.

Unicode [UC00] ist ein internationaler Standardzeichensatz, der benutzt wird, um Dokumente in einer beliebigen Sprache zu verfassen. Unicode gibt dabei jedem Zeichen seine eigene Nummer und ist somit systemunabhängig, programmunabhängig und sprachunabhängig. Sei Σ die endliche Menge der Unicode–Zeichen [UC00, BPS+04, BHL99]. Ein Wort w über dem Alphabet Σ ist eine endliche Folge von n Zeichen $w : \{1, \ldots, n\} \rightarrow \Sigma$, wobei $0 \leq n < \infty$ ist. ε bezeichnet das leere Wort und Σ^n die Menge aller Wörter der Länge n. Die Menge aller Wörter über dem Alphabet Σ wird dann mit $\Sigma^* = \bigcup_{n \leq 0} \Sigma^n$ bezeichnet.

Die Namen von Elementen und Attributen sind Wörter über dem Unicode–Alphabet Σ mit besonderen syntaktischen Eigenschaften [BPS+04, BHL99]. Namen bestehen ausschließlich aus alphanumerischen Zeichen (Buchstaben und Ziffern), Ideogrammen (Schriftzeichen) und den Interpunktionszeichen '_' (Unterstrich), ':' (Doppelpunkt), '.' (Punkt) und '–' (Bindestrich). Zudem beginnt ein Name immer mit einem Buchstaben, einem Ideogramm, einem Unterstrich oder einem Doppelpunkt. Diese Namenskonventionen sind in der folgenden rechtsregulären kontextfreien Grammatik [WM97] zusammengefasst.

Definition 2.1 (Element– und Attributnamen)
Sei Σ das endliche Unicode–Alphabet. Sei $G_N = (N_N, T_N, p_N, S_N)$ eine rechtsreguläre kontextfreie (rrkf) Grammatik mit:

- *einer endlichen Menge N_N=$\{Name\}$ von Nichtterminalen,*
- *einer endlichen Menge $T_N = \Sigma_N \subseteq \Sigma$ von Terminalen, die ausschließlich alphanumerische Zeichen, Ideogramme und die vier Interpunktionszeichen _, :, . und – umfasst,*
- *einem Startsymbol S_N=Name aus N_N und*
- *einer Funktion $p_N : N_N \rightarrow RA$, die die Menge N_N der Nichtterminale in die Menge RA der regulären Ausdrücke über $(N_N \cup T_N)$ abbildet:*

$$p_N(Name) = xy^*$$

wobei $x \in \Sigma_N \backslash \{0, \ldots, 9, ., -\}$ und $y \in \Sigma_N$.

Die von G_N definierte Sprache ist $L(G_N) = \{w \in \Sigma_N^* | Name \stackrel{*}{\Rightarrow} w\}$, die Menge der aus dem Startsymbol S_N=Name gemäß p_N ableitbaren Element– und Attributnamen. □

Der Beginn und das Ende eines Elements wird durch seinen Namen `EName` $\in L(G_N)$ gekennzeichnet. Zur Erkennung wird der Elementname mit den Metazeichen {<,>,</,/>} kombiniert. Diese Kombination wird als Elementtag, kurz *Tag*, bezeichnet. Der Beginn eines Elements wird durch ein Starttag `<EName>`, das Ende eines Elements durch das Endtag `</EName>` und ein leeres Element durch das Leertag `<EName/>` gekennzeichnet (siehe Abb. 2.2).

Die Element– und Attributwerte bestehen aus Unicode–Zeichenketten, die entweder textuelle oder multimediale Inhaltsdaten kodieren. Im Rahmen dieser Arbeit nehmen wir an, dass Element– und Attributwerte in Form von textbasierten Zeichenketten (z.B. textuelle Daten, numerische Inhaltsdaten, Datumsangaben, Währungsangaben) vorliegen, wir betrachten also keine multimedialen Inhaltsdaten (Audio, Video, Bilder, etc.).

In Element– und Attributwerten darf das Zeichen '<' (kleiner als) nicht verwendet werden, da es den Beginn oder das Ende von Elementen kennzeichnet. Im Gegensatz zu den Namenskonventionen gibt es für die Werte von Elementen und Attributen keine weiteren syntaktischen Einschränkungen. Diese Festlegungen lassen sich ebenfalls mit Hilfe einer rechtsreguläre kontextfreie Grammatik formalisieren.

Definition 2.2 (Element– und Attributwerte)
Sei Σ das endliche Unicode–Alphabet. Sei $G_W = (N_W, T_W, p_W, S_W)$ eine rechtsreguläre kontextfreie Grammatik mit:

- *einer endlichen Menge N_W=$\{Wert\}$ von Nichtterminalen,*
- *einer endlichen Menge $T_W = \Sigma_W = \Sigma \backslash \{<\}$ von Terminalen,*
- *einem Startsymbol S_W=Wert aus N_W und*
- *einer Funktion $p_W : N_W \rightarrow RA$, die die Menge N_W der Nichtterminale in die Menge RA der regulären Ausdrücke über $(N_W \cup T_W)$ abbildet:*

$$p_W(Wert) = y^*$$

wobei $y \in \Sigma_W$.

Die von G_W definierte Sprache ist $L(G_W) = \{w \in \Sigma_W^* | Wert \stackrel{*}{\Rightarrow} w\}$, die Menge der aus dem Startsymbol S_W=Wert gemäß p_W ableitbaren Element– und Attributwerte. □

Die Struktur und der Inhalt eines XML–Dokuments lassen sich durch die Definition des Wurzelelements, also seines Namens, seiner Attribute sowie seines Elementinhalts, der die Inhaltsdaten und alle weiteren Subelemente enthält, angeben. Die folgende kontextfreie Grammatik definiert den syntaktischen Aufbau eines beliebigen Elements. Hierbei sind ***Name*** und ***Wert*** Terminalsymbole, die den regulären Sprachen der Definitionen 2.1 und 2.2 entsprechen. Aus Übersichtlichkeitsgründen verzichten wir dabei auf die grammatikalische Modellierung von Leerzeichen.

Definition 2.3 (XML–Syntax von Elementen und Attributen)
*Sei Σ das endliche Unicode–Alphabet. Seien die rechtsregulären kontextfreien Grammatiken G_N und G_W gegeben. Sei **Name** ein Symbol für einen durch G_N erzeugten Ausdruck und **Wert** ein Symbol für einen durch G_W erzeugten Ausdruck.*

Sei $G_{XML} = (N_{XML}, T_{XML}, P_{XML}, S_{XML})$ eine kontextfreie Grammatik mit:

- *einer endlichen Menge N_{XML}={Element, Leertag, Starttag, Endtag, Attribute, Attribut, Elementinhalt} von Nichtterminalen,*
- *einer endlichen Menge $T_{XML} = \Sigma_{XML}$ ={<,>,</,/>,", **Name, Wert**} von Terminalen,*
- *einem Startsymbol S_{XML}=Element aus N_{XML} und*
- *einer Menge von Produktionsregeln $P_{XML} \subseteq N_{XML} \times (N_{XML} \cup T_{XML})^*$:*

$$
\begin{array}{lll}
P_{XML} = \{ & & \\
 & \textit{Element} & \rightarrow \textit{Leertag} \mid \textit{Starttag Elementinhalt Endtag} \\
 & \textit{Leertag} & \rightarrow < \textbf{\textit{Name}} \textit{ Attribute } /> \\
 & \textit{Starttag} & \rightarrow < \textbf{\textit{Name}} \textit{ Attribute } > \\
 & \textit{Endtag} & \rightarrow </ \textbf{\textit{Name}} > \\
 & \textit{Attribute} & \rightarrow \epsilon \mid \textit{Attribut Attribute} \\
 & \textit{Attribut} & \rightarrow \textbf{\textit{Name}} = \textit{"}\textbf{\textit{Wert}}\textit{"} \\
 & \textit{Elementinhalt} & \rightarrow \epsilon \mid \textbf{\textit{Wert}} \textit{ Elementinhalt} \mid \textit{Element Elementinhalt} \\
\} & &
\end{array}
$$

Die von G_{XML} definierte Sprache ist $L(G_{XML}) = \{w \in \Sigma_{XML}^* | Element \stackrel{*}{\Rightarrow} w\}$, die Menge der aus dem Startsymbol S_{XML}=Element gemäß P_{XML} ableitbaren Elemente und Attribute. □

In einer kontextfreien Grammatik ist eine Eigenschaft von XML–Elementen nicht modellierbar, die aber entsprechend der Syntax–Spezifikation [BPS+04] sehr wichtig ist. Der Name eines Elementes ist im Start– und im Endtag verankert, folglich müssen die Namen der beiden Tags eines Elements identisch sein.

Metazeichen der Sprache dürfen gemäß Definition 2.2 in Element– und Attributwerten verwendet werden. Dabei könnte dann folgendes Problem auftreten. Symbolisiert eine gefundene '>' das Ende eines Tags oder ist es Bestandteil der Daten, z.B. einer mathematischen Formel? Damit es zu keinen Missverständnissen bei der Interpretation der Zeichen kommt, werden Metazeichen, die in den Daten verwendet werden, durch sogenannte Entity–Referenzen kodiert. Eine Entity–Referenz ist eine besondere Folge von Zeichen, z.B. '&lt', die einem speziellen Symbol, z.B. '<', zugeordnet wird. Während der Dokumentverarbeitung kann diese Entity–Referenz durch das entsprechende Zeichen zum Beispiel bei der Darstellung des Dokumentes im Webbrowser ersetzt werden,

ohne das oben angedeutete Missverständnisse auftreten. Die XML–Spezifikation gibt bereits fünf solcher Entity–Referenzen fest vor, nämlich &lt für <, &amp für &, &gt für >, &quot für " und &apos für '. Zusätzlich zu diesen fünf Entity–Referenzen besteht die Möglichkeit, innerhalb einer Dokumenttyp–Definition weitere Entity–Referenzen zu deklarieren.

Ein XML–Dokument besteht aus Elementen und Attributen. Die Element– und Attributnamen werden als Strukturdaten und die Element– und Attributwerte als Inhaltsdaten von XML–Dokumenten bezeichnet. Da der Anteil von Struktur– und Inhaltsdaten von Dokument zu Dokument variieren kann, sprechen wir auch von semistrukturierten Dokumenten. Mit Hilfe der XML–Spezifikation [BPS+04] sind wir in der Lage, beliebige semistrukturierte XML–Dokumente zu verfassen. Wir können Namen und Werte von Elementen und Attributen unterscheiden und können die strenge Hierarchie der Elemente erkennen. Ein XML–Dokument wird insbesondere als *wohlgeformt* bezeichnet, wenn seine Syntax der XML–Spezifikation entspricht.

2.2 Dokumenttyp–Definitionen

Mit Hilfe einer Dokumenttyp–Definition (kurz: DTD) können Vorgaben bzgl. der Struktur und dem Inhalt von XML–Dokumenten festgelegt werden. Eine DTD wird im `Prolog` eines XML–Dokumentes angegeben, die DTD–Spezifikation ist Bestandteil der XML–Spezifikation [BPS+04] beschreibt die Syntax und Semantik von Element– und Attributdeklarationen. Sie gibt formal präzise die Namensgebung und die Elementhierarchie vor. Darüber hinaus werden den vorkommenden Attributen Attributtypen zugeordnet (siehe Abb. 2.3).

Beispiel 2.3 In der Abb. 2.3 ist links die DTD zu sehen, die den Aufbau von XML–Dokumenten wie *arts.xml* aus Abb. 2.1 festlegt. Auf der rechten Seite ist die DTD für die Shakespeare–Dramen angegeben.

http://www.arts.edu/arts.dtd

```
1 <!ELEMENT ARTS (ARTIST*,POET*,ACTOR*)>
2 <!ELEMENT ARTIST (#PCDATA)>
3 <!ATTLIST ARTIST
            NAME ID #REQUIRED>
4 <!ELEMENT POET ((TRAGEDIES|COMEDIES|HISTORIES)*,
             POEMS?)>
5 <!ATTLIST POET
            LASTNAME IDREF #REQUIRED
            FIRSTNAME CDATA #REQUIRED>
6 <!ELEMENT TRAGEDIES (title*)>
7 <!ELEMENT COMEDIES (title*)>
8 <!ELEMENT HISTORIES (title*)>
9 <!ELEMENT TITLE (#PCDATA)>
10 <!ATTLIST TITLE
            xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/
                                      xlink/namespace/"
            xlink:type CDATA #FIXED "simple"
            xlink:href CDATA #REQUIRED>
11 <!ATTLIST ACTOR
            LASTNAME IDREF #REQUIRED
            FIRSTNAME CDATA #REQUIRED>
```

http://www.arts.edu/tragedy.dtd

```
12 <!ELEMENT TRAGEDY (TITLE, CHARACTER, DESCRIPTION,
            SUBTITLE, INDUCTION?, PROLOGUE?, ACT+,
            EPILOGUE?)>
13 <!ELEMENT TITLE (#PCDATA)>
14 <!ELEMENT CHARACTER (TITLE, (PERSON | GROUP)+)>
15 <!ELEMENT GROUP (PERSON+, DESCRIPTION)>
16 <!ELEMENT PERSON  (#PCDATA)>
17 <!ELEMENT DESCRIPTION (#PCDATA)>
18 <!ELEMENT SUBTITLE (#PCDATA)>
19 <!ELEMENT INDUCTION (TITLE, SUBTITLE*,(SCENE+ |
            (ACTION | STAGE | SUBHEADING)+))>
20 <!ELEMENT PROLOGUE (TITLE, HEADLINE*, (STAGE | ACTION)+)>
21 <!ELEMENT EPILOGUE (TITLE, HEADLINE*, (STAGE | ACTION)+)>
22 <!ELEMENT ACT (TITLE, SUBTITLE*, PROLOGUE?, SCENE+,
            EPILOGUE?)>
23 <!ELEMENT SCENE (SUBTITLE*, (ACTION | STAGE)+)>
24 <!ATTLIST SCENE
            SUBTITLE  CDATA  #REQUIRED>
25 <!ELEMENT ACTION (SPEAKER, SPEECH)>
26 <!ELEMENT SPEECH (LINE | STAGE | SUBHEADING)+>
27 <!ELEMENT LINE (#PCDATA)>
28 <!ELEMENT SPEAKER (#PCDATA)>
29 <!ELEMENT STAGE (#PCDATA)>
30 <!ELEMENT SUBHEADING  (#PCDATA)>
31 <!ENTITY amp "&#38;#38;">
```

Abbildung 2.3: Syntax einer Dokumenttyp–Definition (DTD) anhand von zwei Beispielen

In den Zeilen 1 – 11 (links) und 12 – 30 (rechts) sind Elemente und deren Attribute deklariert. In der Zeile 31 steht eine Entity–Deklaration.

Eine DTD besteht also im wesentlichen aus Element– und Attributdeklarationen. In diesem Zusammenhang hält die XML– und DTD–Spezifikation zahlreiche reservierte Wörter und Metazeichen mit entsprechenden besonderen Bedeutungen bereit. In den folgenden beiden Abschnitten 2.2.1 und 2.2.2 erläutern wir die Syntax und Semantik von Element– und Attributdeklarationen und fassen am Ende die daraus resultierenden Möglichkeiten bei der Erstellung von semistrukturierten XML–Dokumenten zusammen.

2.2.1 Elementdeklaration

Die Deklaration eines Elements dient zur Festlegung des Elementnamens sowie zur Definition des Elementinhalts bestehend aus beliebigen Elementwerten und/oder weiteren Elementen:

```
<!ELEMENT EName EInhalt>
```

Gemäß der Namenskonventionen aus Definition 2.1 gilt `EName` $\in L(G_N)$.

Der Elementinhalt kann entsprechend der Definition 2.3 eine beliebige Zahl von Objekten enthalten, die dem gegebenen Element untergeordnet sind. Diese Objekte können weitere Subelemente und/oder beliebige Elementwerte sein. Die mögliche Strukturierung des Elementinhalts durch Daten und Subelemente wird durch die Deklaration von `EInhalt` festgelegt.

Ein Elementwert wird mit dem reservierten Wort #PCDATA (parsed character data) in der Elementdeklaration angezeigt.

Die Reihenfolge und die Anzahl der Subelemente wird mit Hilfe reservierter Metazeichen ausgedrückt. Die Reihenfolge wird durch ',' (Komma) festgelegt. '?' (Fragezeichen) steht für ein optionales Vorkommen, '+' (Plus) für ein ein– oder mehrfaches Vorkommen, '*' (Stern) für ein beliebig häufiges Vorkommen des Subelementes bzw. des Klammerausdrucks. Das Zeichen '|' (senkrechter Strich) erlaubt die wahlweise Verwendung des angegebenen Subelements oder Klammerausdrucks.

Beispiel 2.4 Die Elementdeklarationen in einer DTD der Form

```
<!ELEMENT GROUP  (PERSON+, DESCRIPTION)>
<!ELEMENT PERSON (#PCDATA)>
```

(siehe Zeilen 15 und 16 in der Abb. 2.3) besagen, dass das Element `GROUP` mindestens ein weiteres Element `PERSON` und genau ein Element `DESCRIPTION` enthält. Das Element `PERSON` enthält keine weiteren Subelemente.

Vom *gemischten Elementinhalt* (engl.: mixed content) spricht man bei einem Element, wenn sowohl Subelemente als auch Elementwerte im Elementinhalt auftreten. In dem folgenden Beispiel gegeben wir dazu eine Elementdeklaration und die zugehörige Anwendung in einem XML–Dokument an.

Beispiel 2.5 Die Elementdeklaration für ein Element mit gemischtem Inhalt kann folgendes Aussehen haben:

```
<!ELEMENT ABSTRACT (#PCDATA | CONCEPT)*>
```

In einem XML–Dokument können dann folgende Zeilen erscheinen:

```
<ABSTRACT>
   The intelligent search on <CONECPT>XML data</CONCEPT> combines
   query evaluation on document structure given by the <CONCEPT>
   element names<CONCEPT> and their hierarchy and query evaluation
   on content given by element values and attribute values.
</ABSTRACT>
```

Die Reihenfolge von Elementen und deren Werte kann in Abhängigkeit vom Inhalt und der Bedeutung des XML–Dokuments wichtig sein.

Beispiel 2.6 Die folgende Elementdeklaration:

```
<!ELEMENT PROLOGUE (TITLE, HEADLINE*, (STAGE | ACTION)+)>
```

erlaubt XML–Dokumente unterschiedlichen Aufbaus. Diese folgenden beiden Fragmente von XML–Dokumenten genügen der gegebenen Elementdeklaration:

```
<PROLOGUE>                    <PROLOGUE>
  <TITLE> ... </TITLE>          <TITLE> ... </TITLE>
  <STAGE> ... </STAGE>          <HEADLINE> ... </HEADLINE>
  <ACTION> ... </ACTION>        <ACTION> ... </ACTION>
  <STAGE> ... </STAGE>          <ACTION> ... </ACTION>
</PROLOGUE>                   </PROLOGUE>
```

Zuerst muss in einem zugehörigen XML–Dokument das Element `TITLE` erscheinen, dann können mehrere Elemente `HEADLINE` auftreten und danach in beliebiger Reihenfolge die Elemente `STAGE` und `ACTION`.

Durch Klammerung können die angegebenen Subelemente und Elementwerte beliebig kombiniert und mit entsprechenden Optionen versehen werden.

2.2.2 Attributdeklaration

Die Deklaration von Attributen eröffnet die Möglichkeit, nach vorgegebenen Regeln Elemente in XML–Dokumenten durch Attribute und deren Werte ergänzend zu beschreiben. Die Deklaration von Attributen der Form:

```
<!ATTLIST EName
          AName Attributtyp Bedingung>
```

legt fest, zu welchem Element das Attribut gehört, welchen Namen, welchen einfachen Typ und welche Bedingung es an sein Vorkommen hat. Die Element– und Attributnamen müssen dabei den Namenskonventionen entsprechen (Definition 2.1).

Die `Bedingung` bietet die Möglichkeit, Vorgaben über das Vorkommen eines Attributs bzw. seines Wertes im XML–Dokument zu machen. Im Rahmen einer DTD werden die Bedingungen `#IMPLIED` für das optionale Vorkommen des Attributs, `#REQUIRED` für das geforderte Vorkommen des Attributs, `#FIXED` für einen konstanten, unveränderbaren Attributwert und Literal für einen konkreten, in Anführungszeichen gesetzten Attributwert aus $L(G_W)$ angegeben.

Die XML– und DTD–Spezifikationen stellen verschiedene einfache Attributtypen zur Verfügung, nämlich `CDATA, NMTOKEN, NMTOKENS, Enumeration, ENTITY, ENTITIES, ID, IDREF, IDREFS, NOTATION`. Im Rahmen dieser Arbeit sind bestimmte Attributtypen von besonderem Interesse. Für diese Attributtypen werden wir die Auswirkung auf XML–Dokumente, in denen Attribute dieser Typen vorkommen, näher erläutern.

`CDATA` besagt, dass der Attributwert beliebiger Text ist, also ein Wort der Sprache $L(G_W)$ ist.

Ein Attribut vom Typ `ID` dient zur eindeutigen Identifikation eines Elementes durch den zugehörigen Attributwert. Dies bedeutet, dass kein anderes ID–Attribut in demselben XML–Dokument den gleichen Attributwert hat wie das betrachtete ID–Attribut.

Ein Attribut vom Typ `IDREF` verlangt die Existenz eines ID–Attributs im selben XML–Dokument mit gleichem Attributwert.

Ein Attribut vom Typ `IDREFS` umfasst eine Aufzählung von Attributwerten. Zu jedem dieser Werte muss es im selben Dokument jeweils ein ID–Attribut mit genau diesem Wert geben.

`ENTITY`– und `ENTITIES`–Attribute dienen zur Deklaration von Platzhaltern, die innerhalb eines XML–Dokuments während der Verarbeitung entsprechend ersetzt bzw. verarbeitet werden.

Beispiel 2.7 Betrachten wir die DTD aus Abb. 2.3. Die Attributdeklaration zu dem Element `SCENE` der Form:

```
<!ATTLIST SCENE
          SUBTITLE CDATA #REQUIRED>
```

besagt, dass das Element `SCENE` in einem XML–Dokument das Zeichenketten–Attribut `SUBTITLE` haben muss.

Mit Hilfe einer DTD gemäß der XML–Spezifikation [BPS+04] sind wir in der Lage, für XML–Dokumente Vorgaben an die Strukturierung und die Daten zu formulieren. Ein XML–Dokument, das einer vorgegebenen DTD genügt, wird nicht nur als *wohlgeformt* sondern insbesondere auch als *gültig* bezeichnet. Von einem gültigen XML–Dokument können wir neben den Namen der vorkommenden Elemente und Attribute, deren Werte, und der erlaubten Elementhierarchie auch die Attributtypen der vorkommenden Attribute erkennen.

Allerdings können mit Hilfe von DTDs keine Namensräume für die vorkommenden Element– und Attributnamen und keine Datentypen (z.B. Integer, Währung, Datum) für die Element– und Attributwerte definiert werden. Dazu wird anstelle einer DTD ein sogenanntes XML–Schema verwendet [Fall01, TBMM01, BM01]. Im Rahmen dieser Arbeit genügen die Möglichkeiten, die sich durch die Verwendung von DTDs ergeben.

2.3 XML–Verknüpfungen

XML bietet umfangreiche Möglichkeiten der Verknüpfung (*Links*) von Elementen, die zu demselben oder zu verschiedenen XML–Dokumenten gehören. Hierbei unterscheiden wir:

1. Querverweise zwischen Elementen innerhalb eines Dokumentes mittels *ID–Referenzen*,
2. Querverweise von Elementen eines XML–Dokuments auf das Wurzelelement desselben oder anderer XML–Dokumente mittels *XLink* und
3. Querverweise von Elementen eines XML–Dokuments auf Elemente desselben oder anderer XML–Dokumente mittels *XPointer*.

In den nächsten drei Abschnitten werden wir kurz die Syntax und Semantik dieser XML–Verknüpfungen erläutern.

2.3.1 ID–Referenz

Eine ID–Referenz ist eine dokumentinterne Verknüpfung von zwei Elementen ein und desselben XML–Dokuments auf der Basis ihrer Attributwerte und Attributtypen. Die Voraussetzungen für ID–Referenzen werden durch die Deklaration von ID– und IDREF– bzw. IDREFS–Attributen in DTDs geschaffen. Eine ID–Referenz ist der Verweis eines Elements mit einem IDREF– oder einem IDREFS–Attribut auf ein Element mit einem ID–Attribut innerhalb eines XML–Dokuments. Dabei müssen entweder die Attributwerte des IDREF– und ID–Attributs übereinstimmen, oder das ID–Attribut kommt in der Liste der Attributwerte des IDREFS–Attributs vor.

Beispiel 2.8 Zunächst werden in der DTD zu dem Element `PUBLICATION` das ID–Attribut `ISBN`, zu dem Element `PAPERBACK` das IDREF–Attribut `CODE` und zu dem Element `COLLECTION` das IDREFS–Attribut `SET` deklariert.

```
<!ATTLIST PUBLICATION
          ISBN ID #REQUIRED>
...
<!ATTLIST PAPERBACK
          CODE IDREF #REQUIRED>
...
<!ATTLIST COLLECTION
          SET IDREFS #REQUIRED>
```

Nun wird der Ausschnitt eines XML–Dokuments gezeigt, in dem ein `PAPERBACK`–Element und ein `COLLECTION`–Element auf ein `PUBLICATION`–Element verweisen.

```
<PUBLICATION ISBN="1234">
   <PUBLISHER>Addison Wesley</PUBLISHER>
   ...
</PUBLICATION>
...
<PAPERBACK CODE="1234"> ... </PAPERBACK>
...
<COLLECTION SET="1222 1230 1234 1245"> ... </COLLECTION
```

2.3.2 XLink

XLinks ermöglichen die Verknüpfung von einem oder mehreren Elementen eines XML–Dokuments mit einem oder mehreren Elementen eines anderen oder des gleichen XML–Dokuments. Die Definition von XLinks [DMO01] basiert auf der Verwendung eines spezifischen Namensraums (engl. namespace). Einem Attribut (typischerweise xlink genannt) wird der Namensraum für XLinks zugewiesen. Auf diese Weise können alle XLink–Attribute des Namensraums mit dem Attributnamen als Präfix zur Definition des XLinks verwendet werden.

Beispiel 2.9 Betrachten wir die XML–Dokumente aus der Abbildung 1.2. Im XML–Dokument *arts.xml* kommen die folgenden Zeilen vor:

```
<TITLE xmlns:xlink="http://www.w3.org/1999/xlink/namespace/"
       xlink:type="simple"
       xlink:href="http://www.literature.edu/shakespeare/macbeth.xml">
...
</TITLE>
```

Dem Attribut `xlink` wird der Namensraum für XLinks zugewiesen. Die Art des XLinks und das Zieldokument werden in diesem Beispiel mit Hilfe der XLink–Attribute `type` und `href` festgelegt, wobei diese immer in Verbindung mit dem zugehörigen Attribut aus dem XML–Dokument auftreten. Der einfache XLink verbindet das Element `TITLE` des XML–Dokuments *arts.xml* mit dem Wurzelelement `TRAGEDY` des XML–Dokuments *macbeth.xml*.

Die XLink–Attribute, z.B. `type, href, role, arcrole, title, show, actuate, label, from, to`, definieren die Art und das Verhalten des XLinks während der Dokumentverarbeitung. Im Rahmen dieser Arbeit betrachten wir ausschließlich einfache XLinks, bei denen ein Element mit den Attributen `xlink:type="simple"` und `xlink:href="..."` auf das Wurzelelement des XML–Dokuments verweist, dessen URL im href–Attribut angegeben ist.

Die Spezifikation von XLinks ist nicht Bestandteil der XML–Spezifikation, bildet aber die Grundlage für die dokumentübergreifende Verknüpfung von Elementen. Für die Erkennung von einfachen XLinks in XML–Dokumenten bestehend aus einem `type`– und einem `href`–Attribut ist eine DTD nicht zwingend erforderlich.

2.3.3 XPointer

XPointer ist eine Erweiterung des XLink–Konzepts, bei dem Elemente dokumentübergreifend verbunden werden können, und ermöglicht den Verweis von einem Element eines XML–Dokuments auf ein beliebiges anderes Element desselben XML–Dokuments oder eines anderen XML–Dokuments. Bei einem XPointer [GMMW03] wird das XLink–Attribut `href` um die genaue Angabe des Zielelements in der Form `xlink:href="http://...#xpointer(expression)"` erweitert. Hierbei steht `expression` für einen XPath–Ausdruck [BBC+03], der den Pfad vom Wurzelelement des Zieldokuments zum eigentlichen Zielelement festlegt.

Beispiel 2.10 Betrachten wir wiederum die XML–Dokumente aus der Abbildung 1.2. Seien dabei die folgende Zeilen aus dem XML–Dokument *references.xml*:

```
<CITATION
  xmlns:xlink="http://www.w3.org/TR/xlink/"
  xlink:type="simple"
  xlink:href=".../macbeth.xml#xpointer(id('SCENE III. A heath near Forrest.'))">
...
</CITATION>
```

Zunächst verweist der einfache XLink auf das Macbeth–Dokument. Der XPointer gibt dann an, dass das Zielelement das ID–Attribut mit dem oben genannten Wert besitzen muss.

Im Rahmen dieser Arbeit beschränken wir uns auf die Betrachtung einfacher XPointer, bei denen ein Element auf ein eindeutiges Zielelement durch Angabe des ID–Attributwertes verweist. In diesem Fall muss das Zieldokument über eine DTD verfügen, so dass die ID–Attribute des Zieldokuments ermittelt werden können. Je nach dem, wie der XPath–Ausdruck aussieht, ist die Existenz einer DTD für das Zieldokument nicht erforderlich. Analog zu XLinks muss das XML–Dokument, in dem der XPointer angegeben ist, nicht zwingend über eine DTD verfügen.

Mit Hilfe der XML–Verknüpfungen können Elemente dokumentintern oder dokumentübergreifend verbunden werden. Auf diese Weise entsteht durch ID–Referenzen innerhalb eines XML–Dokuments aus der dokumentbasierten Elementhierarchie ein dokumentbasiertes Netz von Elementen. Durch die Hinzunahme von Links zwischen Elementen verschiedener XML–Dokumente bildet sich ein dokumentübergreifendes Netz von Elementen.

2.4 XML–Graph

Die Annotationssprache XML sowie weitere XML–Technologien (ID–Referenzen, XLink, XPointer) ermöglichen uns, semistrukturierte XML–Dokumente mit jeweils einem dokument–eindeutigen Wurzelelement zu erstellen. Darüber hinaus können für XML–Dokumente dokumentinterne und dokumentübergreifende Links zwischen Elementen angelegt werden.

Semistrukturierte XML–Dokumente mit jeweils einer dokumentbasierten Elementhierarchie und einer Menge von Links zwischen Elementen desselben XML–Dokuments oder verschiedener XML–Dokumente legt die Aufbereitung dieser Dokumente in einer graphbasierten Datenstruktur mit dem Ziel, von der XML–Syntax zu abstrahieren unter Erhaltung der Struktur– und Inhaltsinformationen der XML–Dokumente, nahe.

In diesem Abschnitt werden wir eine Menge von textuell gegebenen XML–Dokumenten in eine graphbasierte Darstellung in zwei Schritten übertragen. Im ersten Schritt führen wir eine Tupelschreibweise für XML–Dokumente ein, um Namen und Werte von Elementen und Attributen von den Metazeichen der XML–Syntax zu abstrahieren. Außerdem fassen wir die dokumentinternen und dokumentübergreifenden Links in einer Menge zusammen. Im zweiten Schritt werden wir auf der Basis der Tupelschreibweise und der Linkmenge einen gerichteten Graphen definieren. Dabei sollen folgende Informationen unverändert erhalten bleiben:

- die Element– und Attributnamen
- die Element– und Attributwerte,
- die Zuordnung der Attribute zu den entsprechenden Elementen,
- die Hierarchie der Elemente innerhalb eines XML–Dokuments,
- die Reihenfolge der Subelemente und Elementwerte eines Elements sowie
- die dokumentinternen und dokumentübergreifenden Links zwischen Elementen.

Aus der Definition 2.3 lässt sich direkt eine Tupelschreibweise ableiten, die die XML–spezifischen Syntaxmerkmale vernachlässigt, die Struktur und die Daten der XML–Dokumente aber unberührt lässt.

Definition 2.4 (Element und XML–Dokument)
Sei Σ das endliche Unicode–Alphabet. Seien die rrkf Grammatiken:

– G_N=($\{Name\}$, Σ_N, p_N, *Name) und*
– G_W=($\{Wert\}$, $\Sigma_W = \Sigma \backslash \{<\}$, p_W, *Wert) gegeben.*

Ein Element ist ein 3-Tupel der Form (label, attributes, content) mit einem Elementnamen label $\in L(G_N)$*, einer u.U. leeren Liste attributes von Attributen und einer u.U. leeren Liste content, die den Elementinhalt angibt. Der Elementinhalt kann aus weiteren Elementen bzw. Elementwerten aus* $L(G_W)$ *bestehen. Ein Attribut besteht aus einem Attributname aus* $L(G_N)$*, der innerhalb des Elements eindeutig ist, und einem Attributwert aus* $L(G_W)$*.*

Ein XML-Dokument ist ein (Wurzel–)Element, das in keinem anderen Element enthalten ist. □

Für ein gegebenes XML-Element x bezeichnen *x.label, x.attributes* und *x.content* die entsprechenden Komponenten. *x.attributes[j]* bezeichnet das j-te Attribut der Attributliste des Elements x, wobei $1 \leq j \leq |x.attributes|$. Mit *x.attributes[j].name* und *x.attributes[j].value* werden der Attributname bzw. der Attributwert referenziert. *x.content[i]* bezeichnet das i-te Objekt des Elementinhalts eines Elements x mit $1 \leq i \leq |x.content|$. Dabei kann ein Objekt *x.content[i]* entweder ein Element y oder ein Elementwert sein.

Beispiel 2.11 Das XML–Dokument *macbeth.xml* aus der Abb. 2.1 lässt sich also durch ein 3–Tupel beschreiben:

```
(tragedy, {}, {
     (title, {}, {"The Tragedy of Macbeth"}),
     ...
     (act, {}, {
         (title, {}, {"Act I"}),
         (scene, {number="Scene I." subtitle="A desert place."}, {
               (stage,  {}, {"Thunder and lightning. Enter three Witches."}),
               (speech, {}, {
                      (speaker, {}, {"First Witch"}),
                      "When shall we meet again in thunder, lightning, or in rain?"
               })
         ...}),
     ...})
... })
```

Dabei ist `tragedy` der Name des Wurzelelements, das ein Attribut und zahlreiche Subelemente besitzt. Jedes dieser Subelemente ist wiederum durch ein entsprechendes 3–Tupel bestehend aus dem Elementnamen, der Attributliste und dem Elementinhalt beschrieben.

Die Tupelschreibweise repräsentiert die Elementhierarchie eines XML–Dokuments. Die bestehenden Links werden dabei nicht berücksichtigt. Aus diesem Grund führen wir die Menge $\mathcal{L}$ ein, die die dokumentinternen und dokumentübergreifenden Links beinhaltet. Dazu nehmen wir an, dass jedes Element eines XML–Dokuments separat in einer Menge $\mathcal{X}$ erfasst ist.

Definition 2.5 (Linkmenge)
Sei $\mathcal{X}$ die Menge der Elemente einer beliebigen Menge von XML–Dokumenten. Ein Link zwischen zwei Elementen $x, y \in \mathcal{X}$ wird durch ein Paar (x,y) angegeben, wobei das Startelement x auf das Zielelement y mittels einer ID–Referenz, eines XLinks oder eines XPointers verweist. $\mathcal{L}$ bezeichnet dann die Menge der Links zwischen Elementen aus der Menge $\mathcal{X}$, die in den zugehörigen XML–Dokumenten verankert sind. □

Aus der Tupelschreibweise und der Linkmenge, die eine beliebige Menge von XML–Dokumenten bzgl. Struktur– und Inhaltsdaten vollständig repräsentieren, konstruieren wir nun einen gerichteten XML–Graphen.

Definition 2.6 (XML–Graph)
Sei $\mathcal{X}$ die Menge der Elemente einer beliebigen Menge von XML–Dokumenten und sei $\mathcal{L}$ die zugehörige Menge der Links. Sei Σ das endliche Unicode–Alphabet.

Der XML–Graph zu einer Menge $\mathcal{X}$ von Elementen ist der gerichtete, markierte, geordnete Graph $X = (V, E, \Sigma^*)$ mit einer endlichen Menge V von markierten Knoten, einer endlichen Menge $E \subseteq V \times V$ von Kanten und einer Menge Σ^* von Knotenmarkierungen über dem Alphabet Σ.

Die Markierungsfunktion $f_V : V \rightarrow \Sigma^*$ weist jedem Knoten eine Markierung zu.

Sei x=(label, attributes, content) $\in \mathcal{X}$ ein Element. Dann werden die Knoten als Repräsentanten von x wie folgt definiert.

- *Ein n–Knoten (Namensknoten) $y \in V$, der das Element x oder das j-te Attribut ($1 \leq j \leq |\ x.attributes\ |$) des Elementes x repräsentiert, wird mit dem Element– bzw. Attributnamen aus $L(G_N) \subseteq \Sigma^*$ markiert:*

$$f_V(y) = \begin{cases} x.label & \text{falls } y \text{ das Element repräsentiert} \\ x.attributes[j].name & \text{falls } y \text{ das } j\text{–te Attribut repräsentiert} \end{cases}$$

- *Ein i–Knoten (Inhaltsknoten) $y \in V$, der einen Elementwert $x.content[i]$ aus dem Elementinhalt ($1 \leq i \leq |\ x.content\ |$) oder den Attributwert des j-ten Attributes ($1 \leq j \leq |\ x.attributes\ |$) eines Elementes x repräsentiert, wird mit dem Element– bzw. Attributwert aus $L(G_W) \subseteq \Sigma^*$ markiert:*

$$f_V(y) = \begin{cases} x.content[i] & \text{falls } y \text{ einen Elementwert repräsentiert} \\ x.attributes[j].value & \text{falls } y \text{ den Attributwert repräsentiert} \end{cases}$$

Der Graph besteht aus Element– und Attributkanten $E = E_e \cup E_a$.

- *Es gibt eine gerichtete Elementkante $(a, b) \in E_e$, falls einer der folgenden Fälle zutrifft.*
  1. *Der Knoten a repräsentiert das Element x und der Knoten b repräsentiert y, ein Subelement oder einen Elementwert von x, also $y \in x.content$.*
  2. *Die Knoten a, b repräsentieren die Elemente x, y und $(x, y) \in \mathcal{L}$, d.h. es gibt einen Link von x nach y,*
  3. *Die Knoten a, b repräsentieren die Elemente x, y und $(y, x) \in \mathcal{L}$, d.h. es gibt einen Link von y nach x.*
- *Es gibt eine gerichtete Attributkante $(a, b) \in E_a$, falls einer der beiden Fälle zutrifft.*

1. *Der Knoten a repräsentiert ein Element x und der Knoten b eines der Attribute y von x, also $y \in x.attributes$.*
2. *Es gibt einen Knoten c, der ein Element x repräsentiert, so dass der Knoten a ein Attribut y des Knotens x repräsentiert, also $y \in x.attributes$. Zudem repräsentiert der Knoten b den Attributwert $y.value$ des Attributs y.*

Es gibt eine lokale Ordnung auf den Kindern eines Knotens, die über Elementkanten erreicht werden. Sei c der Knoten, der ein Element x repräsentiert. Seien a, b Kinder von c, die die Subelemente/Elementwerte y=$x.content[i]$ und z=$x.content[j]$ von x repräsentieren, also $(c, a), (c, b) \in E_e$. Dann sind die Knoten gemäß ihrer Position im Elementinhalt von x geordnet, also $a < b \iff i < j$ bzw. $a > b \iff i > j$. □

Die Ordnung der Objekte eines Elementinhalts eines Elements ist zur Reproduktion von Elementwerten in der richtigen Reihenfolge erforderlich (siehe Abb. 2.4 *reference.xml*).

Beispiel 2.12 Erinnern wir uns an die Shakespeare–Dokumente im XML–Format aus Abb. 2.1. In der Abbildung 2.4 ist ein gerichteter Graph zur Repräsentation dreier dieser XML–Dokumente dargestellt. Die Knoten enthalten die Element– und Attributnamen und die gestrichelten Knoten die Element– und Attributwerte.

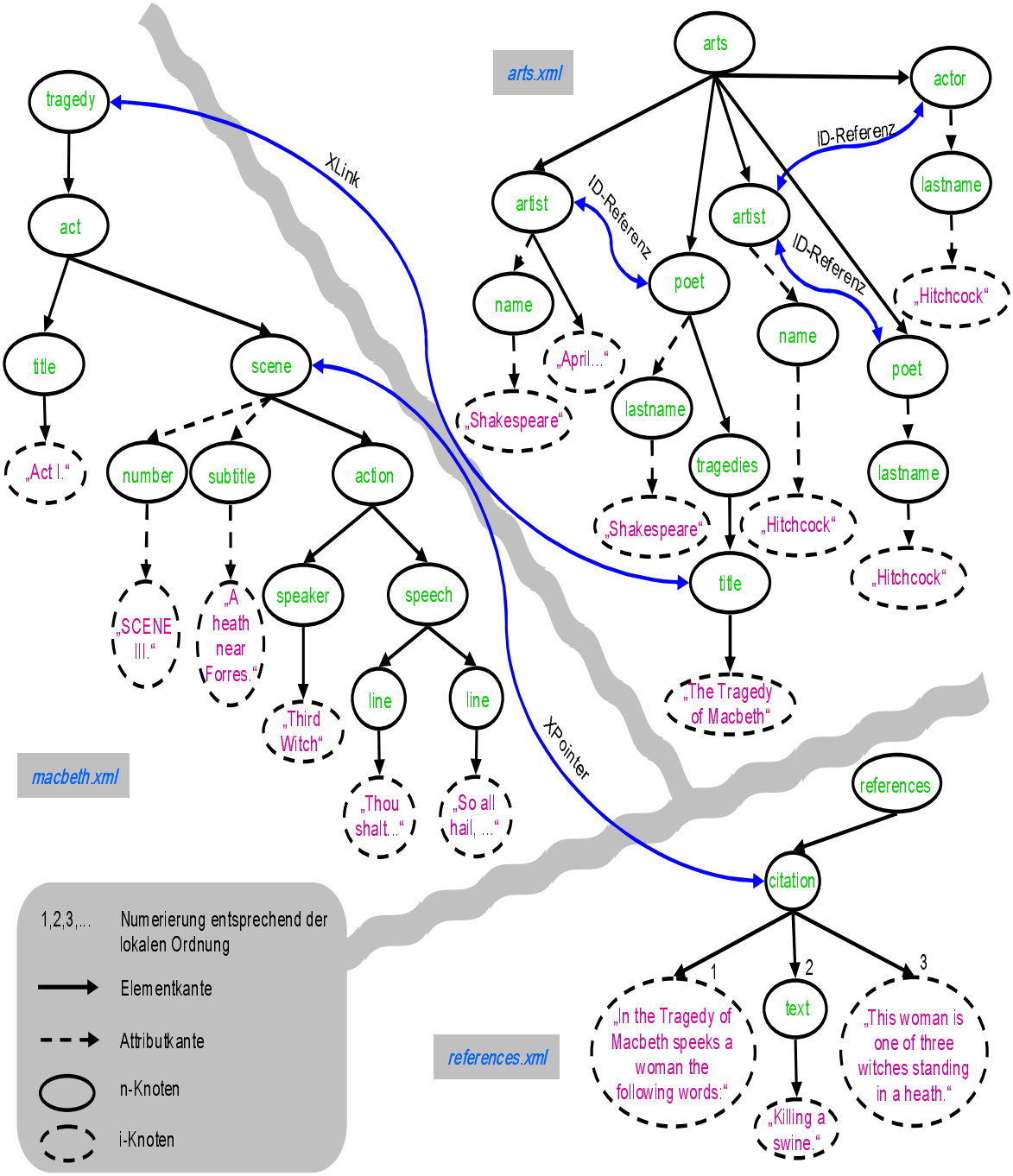


Abbildung 2.4: XML–Graph für die Elemente von drei XML–Dokumenten

Ein (gerichteter) *Pfad p* im Graphen X ist eine zusammenhängende Folge von Knoten, die über gleich gerichtete Kanten verbunden sind. Insbesondere bezeichnen wir einen Pfad, der nur aus n–Knoten besteht, als *n–Pfad*.

Ein Pfad lässt sich eindeutig durch die Konkatenation der beteiligten Knoten und mehrdeutig durch die Konkatenation der Namen und Inhalte der beteiligten Knoten bezeichnen. Da '.' (Punkt) Bestandteil eines Namens sein kann, verwenden wir '/' (Slash) als Konkatenationstrennzeichen. Zusätzlich wird jeder Inhalt in doppelte Anführungszeichen gesetzt.

Beispiel 2.13 Die zwei Knotenfolgen

```
1. play/act/scene
2. play/act/scene/speech/speaker/"First Witch"
```

sind Pfade des XML–Graphen aus Abb. 2.4, deren Bezeichnung nicht eindeutig ist. Dabei ist der erste ein n–Pfad.

An dieser Stelle möchten wir die wichtigsten Grundlagen diese Kapitels noch einmal zusammenfassen.

Jedes dieser XML–Dokumente besteht aus hierarchisch geordneten Elementen beginnend bei einem eindeutigen Wurzelelement. Auf diese Weise wird der Dokumentinhalt in Informationseinheiten partitioniert, der Dokumentinhalt strukturiert und durch die sorgfältige Wahl der Elementnamen annotiert. Zusätzlich gibt es die Möglichkeit, Elemente von XML–Dokumente dokumentintern und dokumentübergreifend über entsprechende *Links* miteinander zu verknüpfen. XML–Dokumente bestehen also aus Strukturdaten und Inhaltsdaten. Der Anteil von Struktur– und Inhaltsdaten kann von XML–Dokument zu XML–Dokument variieren; deshalb spricht man auch von *semistrukturierten* Dokumenten. Die Strukturdaten der XML–Dokumente (Element– und Attributnamen, dokumentbasierte Elementhierarchie, Verknüpfungen von Elementen) haben bzgl. der Dokumentinhalte beschreibenden Charakter. Die partitionierten Bestandteile der Dokumentinhalte werden als Inhaltsdaten (Element– und Attributwerte) des XML–Dokuments bezeichnet.

Eine Menge von XML–Dokumenten lässt sich somit durch einen gerichteten, lokal geordneten XML–Graphen darstellen. Dieser weist folgende Besonderheiten auf.

- Die Abbildung von XML–Dokumenten aus dem Web auf einen XML–Graphen führt aufgrund der webtypischen Linkstruktur dazu, dass der XML–Graph nicht zwingend zusammenhängend sein muss.
- Jeder Link zwischen zwei Elementen x, y wird im XML–Graphen durch zwei gerichtete Kanten (x, y) und (y, x) dargestellt.
- Die lokale Ordnung von Knoten bezieht sich jeweils auf die Objekte (Elementwerte und Subelemente) des Elementinhalts $x.content$ eines gegebenen Elements $x \in \mathcal{X}$ und ist nicht total, aber transitiv.
- Die Reihenfolge der Attribute eines Elements wird vernachlässigt.
- Jeder i–Knoten ist eine Senke, d.h. er hat keine ausgehenden Kanten.
- Jeder i–Knoten gehört zu genau einem n–Knoten.
- Zu einem n–Knoten gehört genau ein i–Knoten, falls der n–Knoten einen Attributnamen repräsentiert.
- Zu einem n–Knoten können mehr als ein i–Knoten gehören, falls dieser n–Knoten einen Elementnamen repräsentiert.
- Jeder n–Pfad besteht aus einer Folge von n–Knoten.

Unter *semistrukturierten XML–Daten* verstehen wir die Informationen eines XML–Dokumentes bzw. einer Menge von XML–Dokumenten unabhängig von seiner/ihrer Repräsentation (textuell, graphbasiert, etc.).

Kapitel 3

Ontologie

Ein zentraler Begriff im Information Retrieval ist die Relevanzbewertung von Informationen. Eine besondere Herausforderung ist dabei die Relevanzbewertung, die auf der semantischen Ähnlichkeit einer Information zu einer gegebenen Bedingung einer Anfrage beruht.

Um semantische Ähnlichkeit zwischen Wörtern auf der Basis ihrer Bedeutung verstehen zu können, müssen wir die Zusammenhänge von Wort/Bedeutung/Begriff und Daten/Wissen/Information begreifen.

Bei der Informationssuche ist es wünschenswert, zum einen Mehrdeutigkeiten von Wörtern zu erkennen und in der Sortierung des Ergebnisses deutlich zu machen und zum anderen semantisch ähnliche Wörtern in die Suche einfließen zu lassen, ohne sie explizit aufzählen zu müssen.

Dazu fassen wir ein Wort und eine Bedeutung zu einem Begriff zusammen, so dass wir ein Ordnungssystem für Begriffe angeben können, mit dessen Hilfe Mehrdeutigkeiten und semantische Ähnlichkeiten von Wörtern in die Anfrageauswertung einfließen können. Als semantisches Ordnungssystem auf der Basis der Bedeutungen von Wörtern betrachten wir in dieser Arbeit Ontologien, auch Begriffshierarchien genannt. Dazu schaffen wir zunächst die Grundlagen für das Verständnis von Wörtern, Begriffen und Ontologien.

Definition 3.1 (Wort)
Sei Σ das endliche Unicode–Alphabet. Ein Wort w über dem Alphabet Σ wird als endliche Folge von n Zeichen ($n \in \mathbb{N}_0$) definiert:

$$w : \{1, \ldots, n\} \to \Sigma$$

Dann ist $\varepsilon : \emptyset \to \Sigma$ das leere Wort und Σ^n ($n \in \mathbb{N}_0$) die Menge aller Wörter der Länge n. Die Menge aller Wörter über dem Alphabet Σ ist $\Sigma^* = \bigcup_{n \geq 0} \Sigma^n$. □

Die *Semantik* (Bedeutungslehre) befasst sich mit dem Inhalt von Wörtern. Die *Bedeutung* umfasst die Aspekte des Inhalts, die sich aus den Relationen zwischen den Wörtern und Objekten der realen Welt ergeben. Der Zusammenhang zwischen einem Objekt der realen Welt, einem Wort und seiner Bedeutung wird durch die Abb. 3.1 illustriert.

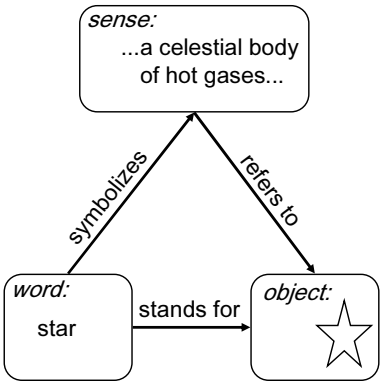


Abbildung 3.1: Zusammenhang von Objekt–Wort–Bedeutung

Der Mensch ordnet in seinem Sprachgebrauch jedem Objekt ein Wort oder mehrere Wörter und jedem Wort eine oder mehrere verschiedene Bedeutungen zu. Begriffe bestehend aus jeweils einem Wort und einer zugehörigen Bedeutung werden im Universum $\mathcal{U}$ zusammengefasst. Dabei betrachten wir $\mathcal{B}$ als die Menge der Bedeutungen, wobei die Art der Repräsentation einer Bedeutung b zu einem Wort w an dieser Stelle noch ohne Belang ist.

Definition 3.2 (Begriff, Universum)
Sei $w \in \Sigma^*$ ein Wort über dem Alphabet Σ. Sei $b \in \mathcal{B}$ eine Bedeutung. Ein Begriff $u=(w,b)$ ist ein Paar, für das gilt, dass w die Bedeutung b hat. Das Universum $\mathcal{U}$ ist dann die Menge aller Begriffe:

$$\mathcal{U} = \{(w, b) \mid w \in \Sigma^*,\ b \in \mathcal{B} :\ \textit{Wort } w \textit{ hat Bedeutung } b\}$$

□

Die Semantische Beziehung zwischen zwei Begriffen wird aus ihren Bedeutungen abgeleitet.

Beispiel 3.1 Die Tabelle 3.1 verdeutlicht die Mehrdeutigkeit des englischen Wortes „star“ sowie die semantische Ähnlichkeit der Begriffe (star, ... a celestial body of hot gases ...) und (sun, ... any star around which a planetary sytem evolves ...). Die Bedeutungen stammen aus dem Thesaurus WordNet [WordNet].

Wort w	Bedeutung b
star	... a plane figure with 5 or more points...
star	... a celestial body of hot gases...
sun	... any star around which a planetary system evolves...

Tabelle 3.1: Mehrdeutigkeit von Wörtern und semantische Ähnlichkeit von Begriffen

---

Definition 3.3 (Ontologie)
Eine Ontologie (ein Thesaurus) ist eine Spezifikation von einem repräsentativen Vokabular aus Begriffen einschließlich der hierarchischen Beziehungen und der assoziativen Beziehungen zwischen diesen Begriffen auf der Basis ihrer Bedeutungen. Sie wird zum Indexieren, zur Suche und zur Unterstützung von gemeinsamer Wissensnutzung (knowledge sharing) und Wissenswiederverwendung (knowledge reuse) verwendet [Gru93, Gua98]. □

Die folgenden Abschnitte befassen sich schwerpunktmäßig mit der Entwicklung eines Datenmodells, der Untersuchung des Ontologieinhalts sowie der semantischen Ähnlichkeit von Ontologiebegriffen. Das weitere Kapitel ist dazu in die folgenden acht Abschnitte 3.1 bis 3.8 gegliedert.

- Im ersten Abschnitt entwickeln wir ein Datenmodell für eine themenunabhängige Ontologie.
- Im zweiten Abschnitt untersuchen wir die Ontologiedaten, die teilweise aus einer umfangreichen, bereits existierenden ontologischen Wissensbasis stammen.
- In den zwei darauf folgenden Abschnitten definieren wir die Quantifizierung der semantischen Ähnlichkeit von benachbarten Begriffen auf der Basis der statistischen Korrelation dieser Begriffe.
- Im fünften Abschnitt definieren wir die semantische Ähnlichkeit zwischen zwei beliebigen Begriffen der Ontologie.
- Darauf aufbauend geben wir einen Algorithmus an, mit dem zu einem gegebenen Begriff mit Hilfe der Ontologie semantisch ähnliche Begriffe ermittelt werden können.
- Im vorletzten Abschnitt geben wir eine Methode an, wie ein neuer Begriff in die Ontologie eingefügt wird.
- Abschließend stellen wir eine Variante der Präsentation der Ontologie für den Benutzer vor.

Im Rahmen dieser Arbeit beschränken wir uns auf die Betrachtung von Substantiven, ihren Bedeutungen und den semantischen Beziehungen zwischen ihnen. Dazu werden wir in den folgenden Beispielen englische Substantive aus dem Thesaurus WordNet [WordNet, Fell98] verwenden.

3.1 Ontologiegraph

Im Rahmen dieser Arbeit sind wir an einem Datenmodell für eine themenunabhängige Ontologie interessiert, mit dem wir

1. die Mehrdeutigkeit von Begriffen erkennen können,
2. Begriffe mit gleicher Bedeutung erkennen können,
3. semantische Beziehungen zwischen Begriffen ausdrücken können und
4. das den Grad der semantischen Beziehung zwischen zwei Begriffen quantitativ wiedergeben kann.

Dazu müssen wir zunächst die semantischen Beziehungen zwischen Begriffen definieren und die aus der Linguistik gebräuchliche Terminologie eingeführen. Die semantische Beziehung zwischen Begriffen ergibt sich aus ihren Bedeutungen.

Definition 3.4 (Semantische Begriffsbeziehung)
Seien $u = (w, b), u' = (w', b') \in \mathcal{U}$ zwei Begriffe aus dem Universum $\mathcal{U}$.

- *Der Begriff $u' = (w', b')$ wird als Synonym für u bezeichnet, falls die Bedeutungen b und b' gleich sind.*

- *Der Begriff $u' = (w', b')$ wird als Hypernym (Oberbegriff) zu u bezeichnet, falls die Bedeutung b' allgemeiner als die Bedeutung b ist.*
- *Der Begriff $u' = (w', b')$ wird als Hyponym (Unterbegriff) zu u bezeichnet, falls die Bedeutung b' spezieller als die Bedeutung b ist.*
- *Der Begriff $u' = (w', b')$ wird als Holonym (Ganzheitsbegriff) zu u bezeichnet, falls die Bedeutung b' etwas beschreibt, von dem b einen Teil beschreibt.*
- *Der Begriff $u' = (w', b')$ wird als Meronym (Teilheitsbegriff) zu u bezeichnet, falls die Bedeutung b' einen Teil dessen beschreibt, das b beschreibt.*

□

Beispiel 3.2 In den folgenden Tabellen 3.2 und 3.3 geben wir die Bedeutung von Wörtern und die semantische Beziehung zwischen jeweils zwei Begriffen an.

Begriffsnr.	Wort w	Bedeutung b
(1)	star	. . . a celestial body of hot gases. . .
(2)	star	. . . a performer who receives prominent billing. . .
(3)	headliner	. . . a performer who receives prominent billing. . .
(4)	heavenly body	. . . natural objects visible in the sky. . .
(5)	celestial body	. . . natural objects visible in the sky. . .
(6)	galaxy	. . . a collection of star systems . . .

Tabelle 3.2: Wörter und ihre Bedeutung

Begriffsnr.	Wort w	semantische Beziehung zu einem anderen Begriff
(1)	star	. . . ist ein Hyponym zu (4) . . .
		. . . ist ein Meronym zu (6) . . .
(2)	star	. . . ist ein Synonym für (3) . . .
(3)	headliner	. . . ist ein Synonym für (2) . . .
(4)	heavenly body	. . . ist ein Hypernym zu (1) . . .
(5)	celestial body	. . . ist ein Synonym zu (4) . . .
(6)	galaxy	. . . ist ein Holonym zu (1) . . .

Tabelle 3.3: Semantische Beziehung zwischen Begriffen

Es gibt allerdings weitere Begriffsbeziehungen, z.B. 'ein Begriff u ist eine Instanz eines Begriffs v', 'ein Begriff u gehört zur Klasse des Begriffs v','ein Begriff u ist ein Korrelat zu einem Begriff v'. Im Rahmen dieser Arbeit beschränken wir uns auf die hier definierten Begriffsbeziehungen (Definition 3.4).

Der folgende Satz gibt einen besonderen Zusammenhang zwischen je zwei semantischen Begriffsbeziehungen an.

Satz 3.1
Seien $u = (w, b), u' = (w', b') \in \mathcal{U}$. Dann gilt:

(a) u ist ein Hypernym zu $u' \iff u'$ ist ein Hyponym zu u
(b) u ist ein Holonym zu $u' \iff u'$ ist ein Meronym zu u

□

Beweis:
Die Gültigkeit des Satzes folgt direkt aus der Definition 3.4. □

Die Wörter mit der gleichen Bedeutung lassen sich in Synonymmengen, genannt Synsets, zusammenfassen.

Definition 3.5 (Synset)
Sei $b \in \mathcal{B}$ eine Bedeutung. Eine Synset s wird definiert als die Menge der Wörter $w \in \Sigma^*$ mit der Bedeutung b:

$$s = synset(b) = \{w \mid (w, b) \in \mathcal{U}\}$$

□

An dieser Stelle möchten wir darauf aufmerksam machen, dass aus der Gleichheit zweier Synsets nicht die Gleichheit der zugehörigen Bedeutungen folgt. Aber zu jeder Bedeutung gibt es genau ein Synset. Aus diesem Grund führen wir den Konzeptbegriff ein.

Definition 3.6 (Konzept)
Sei $b \in \mathcal{B}$ eine Bedeutung und sei s ein Synset. Ein Konzept $c = (s, b)$ ist ein Paar, für das gilt s=synset(b). Dann ist $\mathcal{C}$ die Menge aller Konzepte:

$$\mathcal{C} = \{(s, b) | b \in \mathcal{B} \text{ und } s=synset(b)\}$$

□

Ein Konzept $c = (s, b) \in \mathcal{C}$ repräsentiert eine Menge von Begriffen gleicher Bedeutung, nämlich die Begriffe $u = (w, b) \in \mathcal{U}$, für die gilt: $w \in s$, also $w \in \text{synset}(b)$. Im folgenden erlauben wir die Schreibweise $c(w)$ für ein Wort w, das zu dem Synset des Konzeptes c gehört. Damit ist für alle Wörter w, w' eines Synsets mit $c(w)$ und $c(w')$ das selbe Konzept gemeint.

Beispiel 3.3 Die Begriffe (1), (4) und (5) aus der Tabelle 3.2 werden durch folgende Konzepte repräsentiert:

Konzeptnr.	Synset s	Bedeutung b
(1)	star	...a celestial body of hot gases...
(2)	heavenly body, celestial body	...natural objects visible in the sky...

Tabelle 3.4: Konzepte

Für das Design eines Ontologiegraphen gibt es verschiedene Ansätze, z.B.

1. Die Knoten des Graphen enthalten die Begriffe des Universums $\mathcal{U}$ und die semantischen Begriffsbeziehungen werden durch gerichtete, beschriftete Kanten zwischen den Knoten ausgedrückt.

2. Jeder Begriff $u \in \mathcal{U}$ wird einem Konzept $c \in \mathcal{C}$ zugeordnet. Die Knoten des Graphen enthalten die Konzepte der Menge $\mathcal{C}$ und die semantischen Beziehungen zwischen den Konzepten resultieren aus den semantischen Beziehungen der durch sie repräsentierten Begriffe. Diese werden durch gerichtete, beschriftete Kanten zwischen den Knoten ausgedrückt.

Im Rahmen dieser Arbeit verfolgen wir den zweiten Ansatz. Dazu übertragen wir zunächst die semantischen Begriffsbeziehungen auf Konzepte.

Definition 3.7 (Semantische Konzeptbeziehung)
Seien $c = (s, b), c' = (s', b') \in \mathcal{C}$ zwei Konzepte. Seien $\mathcal{U}_s, \mathcal{U}_{s'} \subseteq \mathcal{U}$ die Menge der Begriffe, die durch die Konzepte c, c repräsentiert werden, also $\mathcal{U}_s = \{(w, b) \in \mathcal{U} | w \in s = \ synset(b)\}$ bzw. $\mathcal{U}_{s'} = \{(w', b') \in \mathcal{U} | w \in s' = synset(b')\}$.

- *Das Konzept c wird als Hypernym zu c' bezeichnet, falls für alle Begriffe $u \in \mathcal{U}_s$ und für alle Begriffe $u' \in \mathcal{U}_{s'}$ gilt: u ist ein Hypernym zu u'.*
- *Das Konzept c wird als Hyponym zu c' bezeichnet, falls für alle Begriffe $u \in \mathcal{U}_s$ und für alle Begriffe $u' \in \mathcal{U}_{s'}$ gilt: u ist ein Hyponym zu u'.*
- *Das Konzept c wird als Holonym zu c' bezeichnet, falls für alle Begriffe $u \in \mathcal{U}_s$ und für alle Begriffe $u' \in \mathcal{U}_{s'}$ gilt: u ist ein Holonym zu u'.*
- *Das Konzept c wird als Meronym zu c' bezeichnet, falls für alle Begriffe $u \in \mathcal{U}_s$ und für alle Begriffe $u' \in \mathcal{U}_{s'}$ gilt: u ist ein Meronym zu u'.*

□

Basierend auf den semantischen Konzeptbeziehungen gemäß der Definition 3.7 lassen sich Relationen über dem kartesischen Produkt $\mathcal{C} \times \mathcal{C}$ formulieren.

Definition 3.8 (Konzeptrelationen)
Seien $c, c' \in \mathcal{C}$ zwei Konzepte. Dann spiegeln die nachstehenden Relationen semantische Beziehungen zwischen den Konzepten, also zwischen den Begriffen, die sie repräsentieren, wider.

- *$Hyper \subseteq \mathcal{C} \times \mathcal{C} : (c, c') \in Hyper$, falls c ein Hypernym zu c' ist.*
- *$Hypo \subseteq \mathcal{C} \times \mathcal{C} : (c, c') \in Hypo$, falls c ein Hyponym zu c' ist.*
- *$Holo \subseteq \mathcal{C} \times \mathcal{C} : (c, c') \in Holo$, falls c ein Holonym zu c' ist.*
- *$Mero \subseteq \mathcal{C} \times \mathcal{C} : (c, c') \in Mero$, falls c ein Meronym zu c' ist.* □

Diese Relationen sind partielle Ordnungen, sie sind also jeweils asymmetrisch, irreflexiv und transitiv.

Beispiel 3.4 In Anlehnung an Tabelle 3.4 gehören die Konzepte
– (1) $c = (\{\text{star}\}, \ldots \text{a celestial body of hot gases} \ldots)$
– (2) $c' = (\{\text{heavenly body, celestial body}\}, \ldots \text{natural objects visible in the sky} \ldots)$
zu folgenden Relationen:
– (a) $(c, c') \in Hypo$
– (b) $(c', c) \in Hyper$

Aus dem Satz 3.1 und den Definitionen 3.4 und 3.7 ergibt sich für die oben definierten Konzeptrelationen folgender Zusammenhang.

Satz 3.2 *Seien $c, c' \in \mathcal{C}$ zwei beliebige Konzepte. Dann gilt:*

(a) $(c, c') \in Hyper \iff (c', c) \in Hypo$
(b) $(c, c') \in Holo \iff (c', c) \in Mero$

□

Beweis:
Dieser Zusammenhang ergibt sich direkt aus dem Satz 3.1 und den Definitionen 3.4 und 3.7. □

Nun lässt sich für die Menge der Konzepte $\mathcal{C}$ eine graphbasierte Ontologie definieren, die quantifizierte Hypernym/Hyponym– und Holonym/Meronym–Beziehungen zwischen Konzepten und damit zwischen den entsprechenden Begriffen repräsentiert.

Definition 3.9 (Ontologiegraph)
Sei $\mathcal{C}$ die Menge der Konzepte und sei Σ das endliche Unicode–Alphabet. Der Ontologiegraph (kurz: Ontologie) zu der Menge $\mathcal{C}$ ist ein gerichteter, markierter, gewichteter Graph $O = (V, E, \mathcal{Z}_B)$, wobei $V \subseteq \mathcal{C}$ die endliche Menge der Knoten, $E \subseteq V \times V$ die endliche Menge der gerichteten, markierten Kanten und $\mathcal{Z}_B = \{$is_hypernym_of, is_hyponym_of, is_holonym_of, is_meronym_of $\} \subseteq \Sigma^*$ die Menge der Kantenmarkierungen bezeichnen.

Die Markierungsfunktion $f_E : E \to \mathcal{Z}_B$ weist jeder Kante eine Markierung zu, die die semantische Beziehung der beteiligten Knoten ausdrückt.

Für zwei Knoten $x, y \in V$ gibt es folgende gerichtete, markierte Kanten $e = (x, y) \in E$ von x nach y mit einer Kantenmarkierung:

$$f_E(e) = \begin{cases} is\_synonym\_of & falls\ (x,y) \in Syn \\ is\_hypernym\_of & falls\ (x,y) \in Hyper \\ is\_hyponym\_of & falls\ (x,y) \in Hypo \\ is\_holonym\_of & falls\ (x,y) \in Holo \\ is\_meronym\_of & falls\ (x,y) \in Mero \end{cases}$$

Durch die Kantengewichtung $weight : E \to [0, 1]$ wird jeder Kante ein Kantengewicht zugeordnet, das die semantische Ähnlichkeit der benachbarten Knoten ausdrückt. □

Im Rahmen dieser Arbeit modellieren wir in der Ontologie nur direkte semantische Beziehungen explizit. Für alle oben definierten Kanten $(x, y) \in E, f_E(x, y) = t$ gilt also jeweils zusätzlich: $\neg\exists z \in V : (x, z), (z, y) \in E$ und $f_E(x, z) = f_E(z, y) = t$.

Beispiel 3.5 In der Abbildung 3.2 ist eine Ontologie dargestellt. Dabei wird für jedes Konzept das Synset fett und die Bedeutung kursiv gedruckt. Der Übersichtlichkeit halber haben wir die Kantenbeschriftungen abgekürzt, d.h. "hyper" steht für "is_hypernym_of", etc.

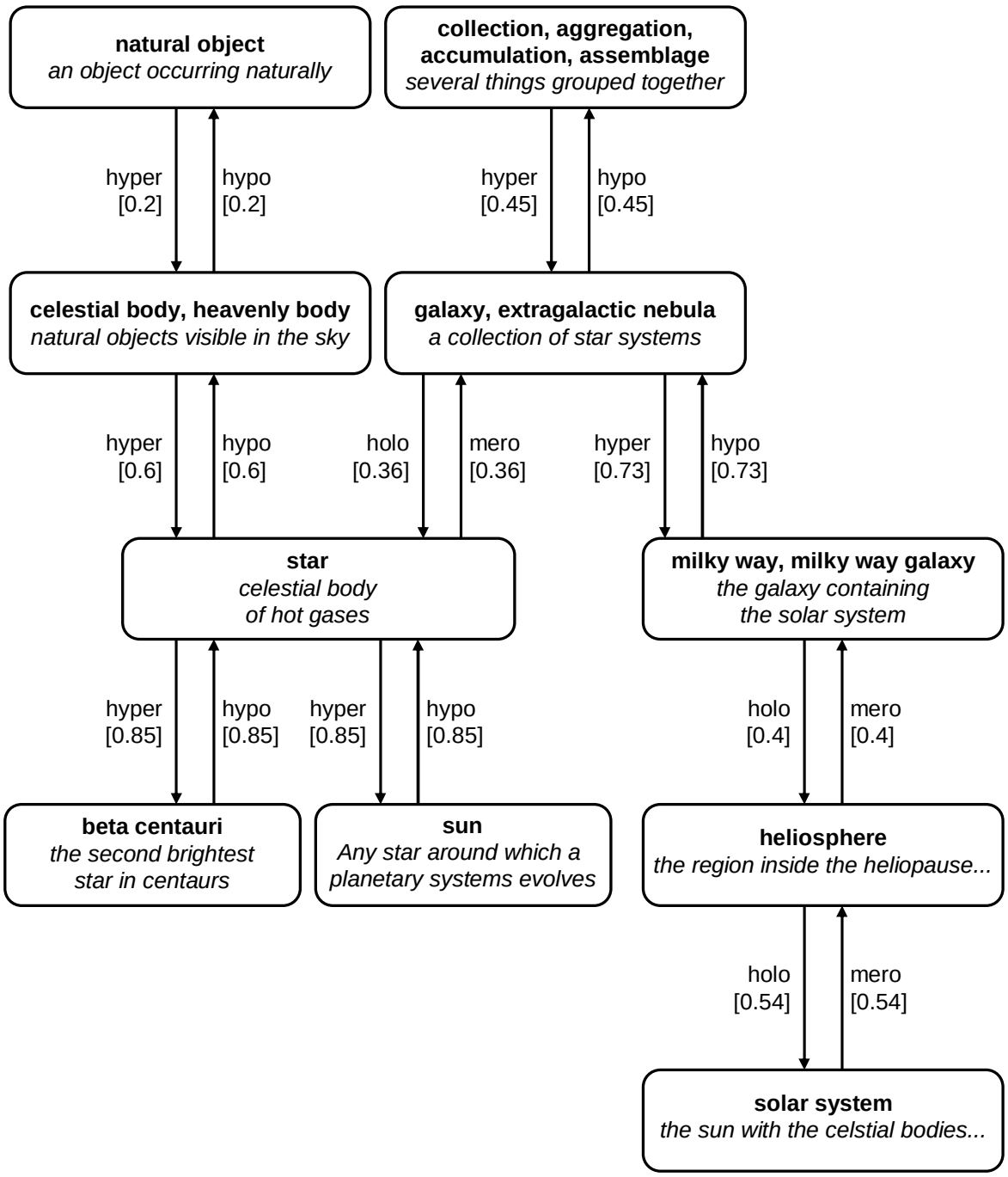


Abbildung 3.2: Ontologiegraph $O = (V, E, \mathcal{Z}_B)$

Semantische Beziehungen, die sich aus der Transitivität der Konzeptrelationen ergeben, werden gemäß der Definition 3.9 nicht explizit durch eigene gerichtete Kanten modelliert. Die Kantengewichte in den eckigen Klammern sind zu diesem Zeitpunkt willkürlich gewählt.

Ein (gerichteter) *Pfad* $p = \langle u_0.u_1. \ \ldots \ .u_k \rangle$ im Graphen O ist eine Folge von Knoten, die über gleich gerichtete Kanten miteinander verbunden sind. Die *Länge* eines Pfades ergibt sich aus der Anzahl der Kanten, in diesem Fall ist $\mid p \mid = k$. Jeder Pfad p kann als Konkatenation von Teilpfaden $p_1.p_2. \ \ldots \ .p_m$ $(1 \leq m)$ aufgefasst werden. Im Gegensatz zum XML–Graphen (siehe Kapitel 2) verwenden wir hier den Punkt zur Darstellung der Konkatenation von Knoten in Pfaden.

3.2 WordNet

Die Zuordnung von Bedeutungen zu gegebenen Wörtern sowie die Benennung der semantischen Beziehung zwischen zwei Begriffen ist eine enorme linguistische und kognitionswissenschaftliche Herausforderung, die wir im Rahmen dieser Arbeit weder automatisch noch manuell lösen möchten. Zur Vereinfachung füllen wir die im vorherigen Abschnitt definierte Ontologie mit Begriffen und den zugehörigen semantischen Beziehungen aus einer umfangreichen, bereits existierenden ontologischen Wissensbasis.

Im Rahmen dieser Arbeit verwenden wir WordNet 1.7.1 [WordNet, Fell98] als themenübergreifende ontologische Quelle für die englische Sprache. WordNet ist eine elektronische Datenbank, die Substantive, Verben, Adjektive und Adverben der englischen Sprache bereitstellt. Diese sind in Konzepten organisiert. Diese Konzepte sind durch verschiedene semantische Beziehungen miteinander verknüpft.

Zu einem gegebenen Wort (noun, d.h. englisches Substantiv) $w \in \Sigma^*$ liefert WordNet Konzepte $c = (s, b)$, wobei die Bedeutung b in Form einer Phrase angegeben ist. Zu jedem Konzept können Konzepte abgerufen werden, die in einer semantischen Beziehung zueinander stehen. In WordNet sind die semantischen Beziehungen zwischen den Konzepten nicht gewichtet (siehe Abb. 3.3).

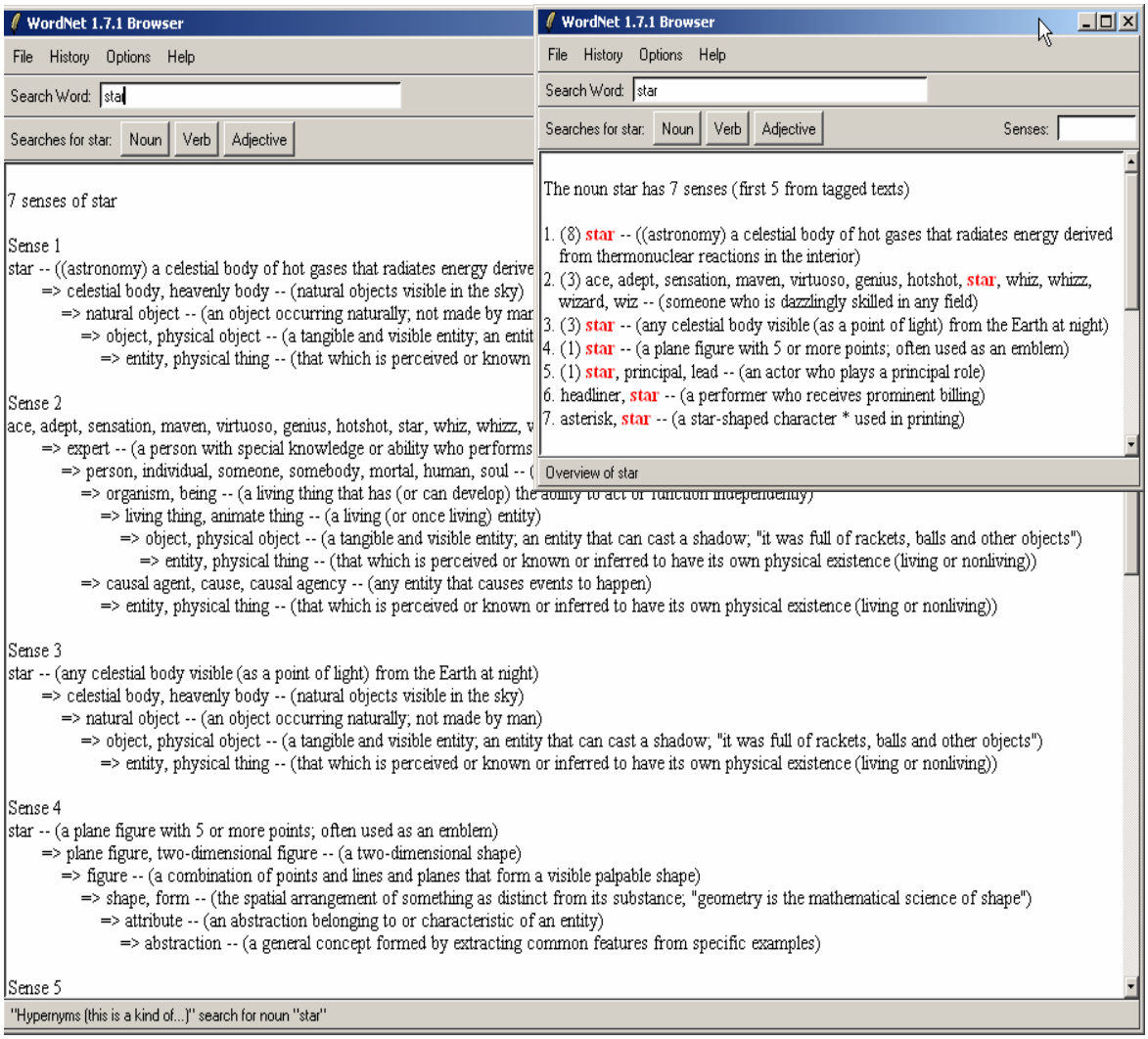


Abbildung 3.3: WordNet–Browser in der Version 1.7.1

Die Konzepte für die Substantive der englischen Sprache und die semantischen Bezie-

hungen zwischen ihnen können aus der WordNet–Datenbank in einen gerichteten, markierten Ontologiegraphen ohne Kantengewichtung, wie er in Definition 3.9 angegeben ist, übertragen werden. Diesen Ontologiegraphen werden wir in Zukunft als WordNet–Ontologie $O_W = (V_W, E_W, \mathcal{Z}_B)$ bezeichnen.

3.3 Kontext eines Konzepts

Im Rahmen der Beschäftigung mit Ontologien eröffnen sich verschiedene Aufgabenfelder, in denen die Bedeutungen von Begriffen eine zentrale Rolle spielen. Wir skizzieren im Folgenden zwei dieser Aufgabenfelder und nehmen dazu an, dass $O = (V, E, \mathcal{Z}_B)$ eine Ontologie sei und $O_W = (V_W, E_W, \mathcal{Z}_B)$ die WordNet–Ontologie auf der Basis der WordNet–Datenbank sei.

1. Eine Ontologie wird unter anderem für die Zuordnung eines Wortes aus einer Anfrage oder einem Dokument zu einem Konzept benötigt. Typischer Weise qualifizieren sich für ein gegebenes Wort w mehrere Konzepte $c = (s, b)$, für die gilt $w \in s$. Wird eine eindeutige Zuordnung gewünscht, müssen weitere Informationen aus dem Ursprungsdokument des Wortes w sowie weitere Informationen zu den betrachteten Konzepten zur Auflösung der Mehrdeutigkeit herangezogen werden.

2. Die Konstruktion einer Ontologie O auf der Basis der Daten einer umfangreichen, statischen, ontologischen Wissensbasis (z.B. WordNet) umfasst zwei Schritte.
   (a) Die Zuordnung einer semantischen Beziehung zwischen zwei gegebenen Konzepten $c, c' \in V$ führt zum Eintrag einer entsprechenden gerichteten, markierten Kante in O. Diese Information ist direkt aus der WordNet–Ontologie in die eigene Ontologie übertragbar.
   (b) Die Bewertung der semantischen Beziehung zwischen den beiden Konzepten erfolgt durch die Zuordnung eines entsprechenden Kantengewichts $weight(c, c') \in [0, 1]$. Dieses Kantengewicht drückt den 'Grad' der semantischen Beziehung aus und beruht auf der Korrelation der Konzepte bzgl. eines repäsentativen Dokumentkorpuses.

Im Rahmen dieser Arbeit genügt uns die Zuordnung eines gegebenen Wortes zu den Konzepten der Ontologie, die dieses Wort in ihrem Synset enthalten. Die Auflösung von Mehrdeutigkeiten ist ein weites Feld und soll an dieser Stelle nicht weiter untersucht werden.

Die Gewichtung semantischen Beziehung zwischen zwei Konzepten der Ontologie gemäß des oben genannten zweiten Punktes soll im Folgenden genauer untersucht.

In dem vorliegenden Szenario stehen die Bedeutungen von Konzepten in Form von Phrasen zur Verfügung. Die Bedeutung eines Konzepts besteht also aus einer Folge von Wörtern zur Erklärung seiner Bedeutung. Diese Folge von Wörtern enthält wichtige Wörter (sinntragende Wörter) und weniger wichtige Wörter (Stopp– und Füllwörter). Somit sind die Bedeutungen in dieser Form zur Bewertung der semantischen Ähnlichkeit zweier Konzepte wenig geeignet. Wir benötigen eine Menge von ausnahmslos wichtigen, sinntragenden Wörtern.

Als Alternative schlagen wir die Verwendung von Kontexten zur Bewältigung der oben gestellten Aufgaben vor. Dabei verstehen wir unter dem Kontext eines Konzepts eine Menge von Wörtern, die dieses Konzept kompakt charakterisieren. Hierzu gibt es verschiedene Möglichkeit.

Ein Ansatz beruht darauf, die wichtigen, sinntragenden Wörter aus der gegebenen Phrase eines Konzepts zu extrahieren, und diese als Kontext des Konzepts zu betrachten.

Ein anderer Ansatz beruht auf der Idee, die Wörter eng verwandter Konzepte zur Kontextbildung heranzuziehen. Dazu würde sich die Konzeptumgebung, also benachbarte Knoten, innerhalb einer Ontologie anbieten. Um ein Konzept durch eine Menge von Wörtern anderer Konzepte zu charakterisieren, eignen sich z.B. die Synsets übergeordneter Konzepte (Väter oder Vorfahren) sowie die Synsets eng verwandter Konzepte (Geschwister).

Im Rahmen dieser Arbeit werden wir den Kontext eines Begriffs wahlweise auf der Basis der Ontologie O oder auf der Basis der WordNet–Ontologie O_W aus den Synsets der Hypernyme konstruieren. Diese Konstruktion erfolgt unabhängig von den gegebenen Phrasen.

Definition 3.10 (Kontext)
Sei $O = (V, E, \mathcal{Z}_B)$ *eine Ontologie und* $c = (s, b) \in V$ *ein Konzept. Der Kontext* $con : \mathcal{C} \rightarrow \{w | w \in \Sigma^*\}$ *des Konzepts c wird aus dem eigenen Synset und den Synsets der Hypernyme zusammengesetzt.*

$$con(c) = s \cup \{s' \mid c' = (s', b') \in V : (c', c) \in E, f_E(c', c) = is\_hypernym\_of\}$$

□

Beispiel 3.6 In der Abbildung 3.4 werden alle Knoten der Ontologie schattiert dargestellt, die zur Kontextbildung des Konzeptes *c=(star, ... a celestial body of hot gases ...)* beitragen.

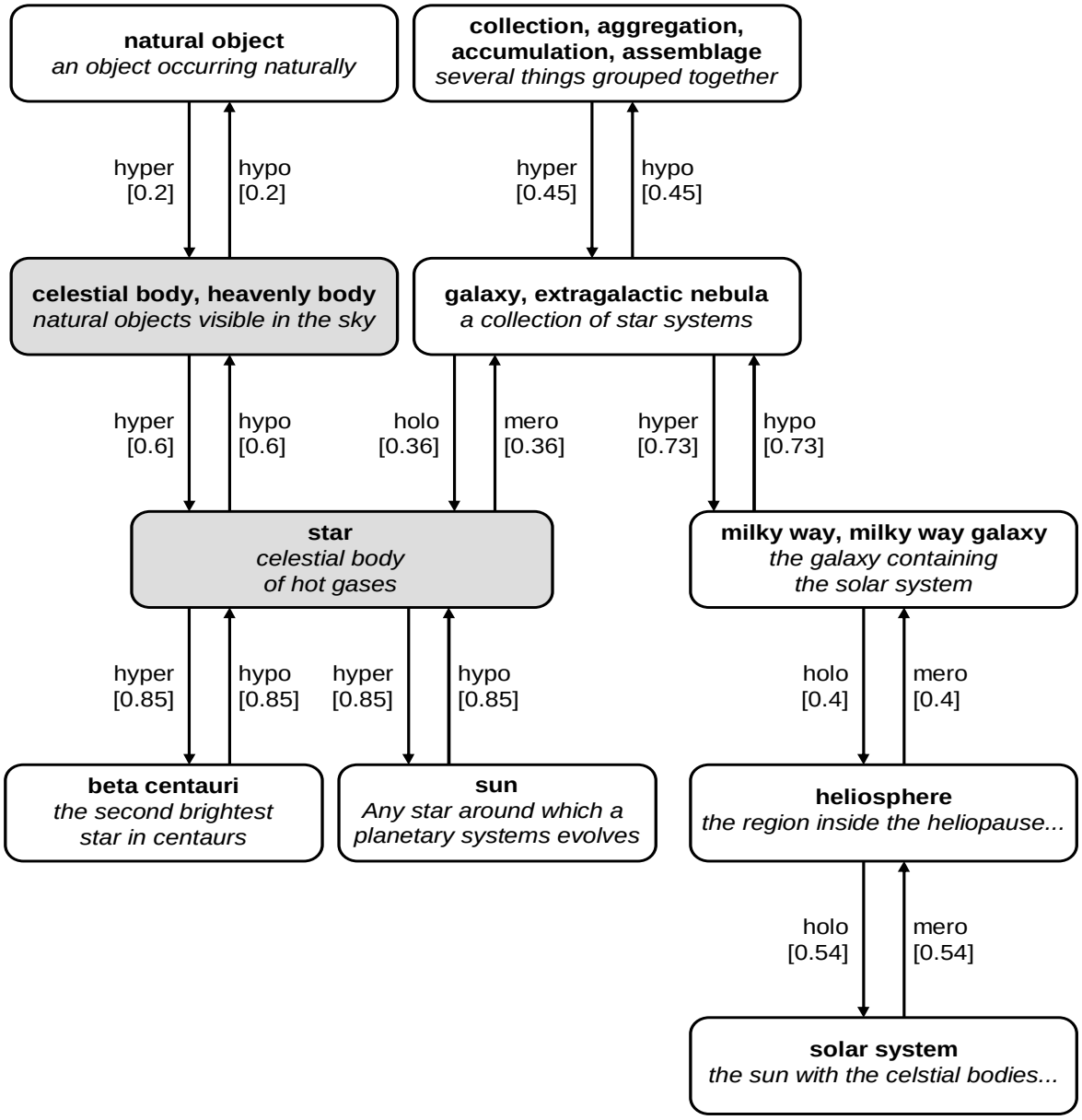


Abbildung 3.4: Kontext eines Begriffs

Der Kontext entspricht dann der hier angegebenen Wortmenge *con(u)=(star, celestial body, heavenly body)*.

Diese Art der Kontextkonstruktion weist eine Besonderheit auf. Es kann Fälle geben, in denen die Kontexte von Polysemen gleich sind. Betrachten wir zum Beispiel die Begriffe c= *(star, ... celestial body of hot gases ...)* und c' =*(star, ... any celestial body visible from the earth at night ...)*. Für beide Begriffe ergibt sich auf der Basis der WordNet–Daten folgender Kontext $con(c) = con(c')$ =*(star, celestial body, heavenly body)*. Im Bereich der ersten Aufgabenstellung hat dieser Umstand keine Auswirkungen, da wir keine Kantengewichtungen für Polysemkanten berechnen müssen. Für das zweite Aufgabenfeld bedeutet es, dass wir in diesem Zusammenhang nur bis zu einem gewissen Grad die Auflösung von Mehrdeutigkeiten erreichen. Darauf werden wir gesondert eingehen.

Zum Abschluß dieses Abschnitts möchten wir darauf hinweisen, dass der Kontext eines Begriffs häufig Wörter enthält, die in der durch eine Phrase gegebenen Bedeutung

auftreten. Der Kontext wird aber unabhängig von dieser Phrase bestimmt und darf in den folgenden Ausführungen nicht mit dieser verwechselt werden.

3.4 Kantengewichtung

Sei die Ontologie $O = (V, E, \mathcal{Z}_B)$ gegeben. Die gerichteten, markierten Kanten drücken in O die semantischen Beziehungen zwischen benachbarten Knoten aus. Die zusätzliche Gewichtung der Kanten mit einem Wert zwischen 0 und 1 drückt den Grad der semantischen Ähnlichkeit der gegebenen Beziehung aus. Zur Quantifikation der semantischen Beziehungen gibt es verschiedene Möglichkeiten.

Im Rahmen dieser Arbeit verfolgen wir folgenden Ansatz. Der Kontext eines Konzepts besteht aus einer Menge von Wörtern, die dieses Konzept charakterisieren aber auch von anderen Konzepten differenzieren. Für eine gerichtete, markierte Kante zwischen den Konzepten $c, c' \in V$ wird das Kantengewicht auf der Basis der Korrelation (Wechselbeziehungen) der zugehörigen Kontexte $con(c)$ und $con(c')$ ermittelt. Die Korrelation zwischen den Kontexten stützt sich auf die Häufigkeiten der Wörter der Kontexte in Dokumenten eines großen Korpus. Betrachten wir das Web als Dokumentkorpus, dann können die Häufigkeiten mit Hilfe einer Suchmaschine wie Google ermittelt werden. Sie gehen dann in eine geeignete Formel zur Berechnung der statistischen Korrelation ein.

Definition 3.11 (Häufigkeitsfunktion)
Sei z eine logische Suchbedingung bestehend aus der logischen Verknüpfung (mittels $\wedge$ und $\vee$) von Wörtern. Dann liefert die Häufigkeitsfunktion $freq(z)$ die Anzahl der Dokumente, die die Bedingung z erfüllen. □

Um eine Suchanfrage an das Web zu stellen, müssen die Wörter der betrachteten Kontexte in geeigneter Art und Weise logisch verknüpft werden. Betrachten wir dazu das Konzept $c = (s_0, b_0) \in \mathcal{C}$ und seinen Kontext $con(c) = s_0 \cup \ldots \cup s_k$ $(0 \leq k \leq \infty)$ bestehend aus den Wörtern des eigenen Synsets s_0 und den Wörtern von k Synsets von übergeordneten Konzepten. Seien $x_1, \ldots, x_{s_i}$ die Wörter eines Synsets s_i $(0 \leq i \leq k)$.

Der Kontext eines Konzepts ist dann wirksam, wenn alle beteiligten Synsets in den betrachteten Dokumenten eine Rolle spielen. Hier bleibt zu entscheiden, ob die Wörter gleicher Bedeutung eines Synsets konjunktiv oder disjunktiv in der Suchanfrage verknüpft werden. Im ersten Fall lautet die Suchanfrage dann:

$$Q = \bigwedge_{i=0}^{k} \left(\bigvee_{j=1}^{s_i} x_j \right)$$

Im zweiten Fall sieht die Suchanfrage wie folgt aus:

$$Q = \bigwedge_{i=0}^{k} \left(\bigwedge_{j=1}^{s_i} x_j \right)$$

Für zwei gegebene Konzepte c und c' und die zugehörigen Kontexte $con(c)$ und $con(c')$ generieren wir die Suchanfragen Q und Q'. Mit Hilfe der Häufigkeitsfunktion $freq()$ und der Suchmaschine Google ermitteln wir

- die Anzahl der Dokumente $freq(Q \wedge Q')$, in denen beide Konzepte c und c' eine Rolle spielen,

- die Anzahl der Dokumente *freq*(Q), in denen das Konzept c vorkommt sowie
- die Anzahl der Dokumente *freq*(Q'), in denen das Konzept c' vorkommt.

Aus der Statistik [MS01] kennen wir Maße, die die Korrelation zweier diskreter Zufallsvariablen ausdrücken, z.B. die symmetrischen Maße Varianz/Kovarianz, das Chi–Quadrat–Maß, der Dice–Koeffizient, der Jaccard–Koeffizient, der Overlap–Koeffizient. Diese Maße können wir verwenden, um mit Hilfe der oben genannten Häufigkeiten statistisch die semantische Korrelation der Konzepte c und c' zu quantifizieren (siehe Tabelle 3.5).

Ähnlichkeitsmaß	Formel
Dice–Koeffizient	$2\frac{freq(Q \wedge Q')}{freq(Q)+freq(Q')}$
Jaccard–Koeffizient	$\frac{freq(Q \wedge Q')}{freq(Q)+freq(Q')-freq(Q \wedge Q')}$
Overlap–Koeffizient	$\frac{freq(Q \wedge Q')}{min\{freq(Q),freq(Q')\}}$

Tabelle 3.5: Korrelationsmaße

Im Rahmen dieser Arbeit betrachten wir die drei Koeffizienten, um die Korrelation zwischen den Konzepten c und c' auf der Basis der ermittelten Häufigkeiten auszudrücken (siehe Tabelle 3.5).

Beispiel 3.7 Für die Ontologie gemäß der Abb. 3.2 ergeben sich unter Verwendung der Suchmaschine Google und der ontologischen Wissensbasis WordNet folgende Häufigkeiten auf der Basis des oben genannten ersten Ansatzes.

In der Tabelle 3.6 sind die konzeptbasierten Häufigkeiten angegeben, wobei die Wörter konjunktiv verknüpft wurden. Aus technischen Gründen erlaubt die Suchmaschine Google höchstens 10 Wörter in einer Anfrage. Die kursiv gedruckten Wörter wurden aus den Anfragen herausgelassen.

$c = (s, b)$	s	Wörter von hypernymen Konzepten	$freq(Q)$
c_1	natural object	object, physical object	2.060
c_2	celestial body, *heavenly body*	natural object	1.060
c_3	star	celestial body, heavenly body	461
c_4	Beta Centauri	star	782
c_5	sun	star	4.430.000
c_6	collection, aggregation, accumulation, *assemblage*	group, grouping	1.580
c_7	galaxy, extragalactic nebula	collection, aggregation *accumulation, assemblage*	97
c_8	milky way, *milky way galaxy milky way system*	galaxy, extragalactic nebula	41
c_9	heliosphere	region, part	3.340
c_{10}	solar system	heliosphere	1.920.000

Tabelle 3.6: Konzeptbasierte Häufigkeiten

Mit Hilfe der Daten aus obiger Tabelle lassen sich dann die Ähnlichkeitsmaße auf der Basis der oben angegebenen Koeffizienten berechnen. Die Ergebnisse sind in der Tabelle 3.7 dargestellt, wobei hier Q und Q' für die Suchanfragen basierend auf den Kontexten $con(c)$ und $con(c')$ stehen.

Kante $e = (c, c')$	$freq(Q \wedge Q')$	Dice	Jaccard	Overlap
(c_1, c_2)	965	0,6186	0,4478	0,9104
(c_2, c_3)	16	0,0210	0,0106	0,0347
(c_3, c_4)	6	0,0097	0,0049	0,0130
(c_3, c_5)	397	0,0002	0,0001	0,8611
(c_3, c_7)	10	0,0358	0,0182	0,1031
(c_6, c_7)	8	0,0095	0,0048	0,0825
(c_7, c_8)	6	0,0869	0,0455	0,1463
(c_8, c_9)	1	0,0006	0,0003	0,0244
(c_9, c_{10})	1.790	0,0019	0,0009	0,5359

Tabelle 3.7: Häufigkeiten und Kantengewichte

Im Rahmen dieser Arbeit verwenden wir zur Berechnung der Kantengewichte den Dice–Koeffizienten. Der Grund dafür beruht auf den Ergebnissen umfangreicher Tests, bei denen die Ergebnisse des Dice–Koeffizienten am praxistauglichsten erschienen.

Definition 3.12 (Kantengewichtung)
Seien $c = (s, b), c' = (s', b') \in \mathcal{C}$ zwei Konzepte, sei Q die Suchanfrage auf der Basis des Kontextes $con(c)$ und sei Q' die Suchanfrage auf der Basis des Kontextes $con(c')$. Die Kantengewichtung $weight : E \to [0, 1]$ ordnet den Kanten $(c, c'), (c', c) \in E$ jeweils das gleiche Kantengewicht zwischen 0 und 1 wie folgt zu:

$$weight(c, c') = weight(c', c) = 2\frac{freq(Q \wedge Q')}{freq(Q) + freq(Q')}$$

□

Hier haben wir auf der Basis der uns zur Verfügung stehenden Mittel (Webdokumente, die einer gewissen Dynamik unterliegen; Suchmaschine Google mit ihren Festlegungen bzgl. der Auswertung von Suchanfragen) einen Ansatz beschrieben, der nicht unbedingt perfekt ist, sich aber vollautomatisch realisieren lässt. In der Praxis werden Kanten mit einem Gewicht *weight=0* üblicherweise nicht erfasst.

3.5 Semantische Ähnlichkeit

Zur Berechnung der semantischen Ähnlichkeit zwischen zwei Knoten der Ontologie werden die Pfade zwischen diesen Knoten und die Gewichte an den Kanten betrachtet. Dabei muss berücksichtigt werden, dass es mehr als einen Pfad von einem Knoten c zu einem Knoten c' geben kann.

Definition 3.13 (Semantische Pfadähnlichkeit)
Sei $O = (V, E, \mathcal{Z}_B)$ eine Ontologie und seien $c, c' \in V$ zwei Konzepte dieser Ontologie. Sei $p = \langle v_0 \dots v_k \rangle$ ein gerichteter Pfad der Länge k ($0 < k < \infty$) in O von c nach c', d.h. $c = v_0$ und $c' = v_k$.

Die semantische Pfadähnlichkeit $sim_p : V \times V \to [0, 1]$ für den Pfad p vom Knoten c zum Knoten c' wird wie folgt definiert:

$$sim_p(c, c') = \prod_{i=1}^{k} weight(v_{i-1}, v_i)$$

□

Zur Erinnerung möchten wir darauf hinweisen, dass es im Ontologiegraphen zu einer Kante $(x, y) \in E$ immer auch die Kante $(y, x) \in E$ gibt. Folglich gibt es fast immer einen Pfad zwischen zwei Begriffen, dessen zunehmende Länge sich aber in einer geringeren semantischen Ähnlichkeit niederschlägt.

Die semantische Ähnlichkeit zwischen zwei Konzepten ergibt sich aus der maximalen Pfadähnlichkeit aller Pfade zwischen diesen beiden Konzepten.

Definition 3.14 (Semantische Konzeptähnlichkeit)
Sei $O = (V, E, \mathcal{Z}_B)$ eine Ontologie und seien $c, c' \in V$ zwei Konzepte dieser Ontologie. Die semantische Ähnlichkeit $sim : V \times V \to [0, 1]$ von c und c' wird wie folgt definiert:

$$sim(c, c') = \begin{cases} max\{sim_p(c, c') | p = \langle c \dots c' \rangle \text{ Pfad in } O\} & \text{– falls es mindestens einen solchen Pfad gibt} \\ 1.0 & \text{– falls c=c'} \\ 0 & \text{– sonst} \end{cases}$$

□

Aus den Definitionen 3.12, 3.13 und 3.14 wird deutlich, dass die Multiplikation von Kantengewichten zwischen 0 und 1 zur Bestimmung der semantischen Ähnlichkeit spezielle Eigenschaften der Pfadähnlichkeit verursacht.

Lemma 3.1 (Eigenschaften für Pfadähnlichkeiten)
Sei $O = (V, E, \mathcal{Z}_B)$ eine Ontologie. Die Berechnung der Pfadähnlichkeit eines Pfades p in O genügt folgenden Eigenschaften.

1. *Alle Kantentypen vergrößern die Pfadähnlichkeit nicht.*
2. *Insbesondere können Zyklen die Pfadähnlichkeit nie vergrößern.*
3. *Die Pfadähnlichkeit für den Pfad von einem Konzept c zu einem Konzept c' ist gleich der Pfadähnlichkeit desselben Pfades in umgekehrter Richtung.*
4. *Die Kantengewichte werden unabhängig voneinander berechnet. Folglich können zwei unterschiedliche Pfade mit gleicher Folge von Kantentypen unterschiedliche Pfadähnlichkeitswerte haben. Insbesondere muss ein kurzer Pfad nicht zwingend die maximale Pfadähnlichkeit besitzen.*

□

Beweis:
Die Gültigkeit der obigen Eigenschaften ergibt sich direkt aus den Definitionen 3.13 und 3.14. □

Daraus ergeben sich für den Fortgang der Arbeit folgende Festlegungen. Zur Bestimmung der semantischen Ähnlichkeit zwischen zwei Konzepten betrachten wir ausschließlich zyklenfreie Pfade. Wegen der Kommutativität der Multiplikation genügt es, nur eine Richtung, also von c nach c' oder umgekehrt, zu betrachten. Wir können keine zusätzlichen Annahmen über eine Relation zwischen der Pfadlänge und der maximalen Pfadähnlichkeit treffen.

Beispiel 3.8 Betrachten wir die Pfade zwischen den Konzepten c_2 und c_5 bzw. c_2 und c_9 in der Abbildung 3.5.

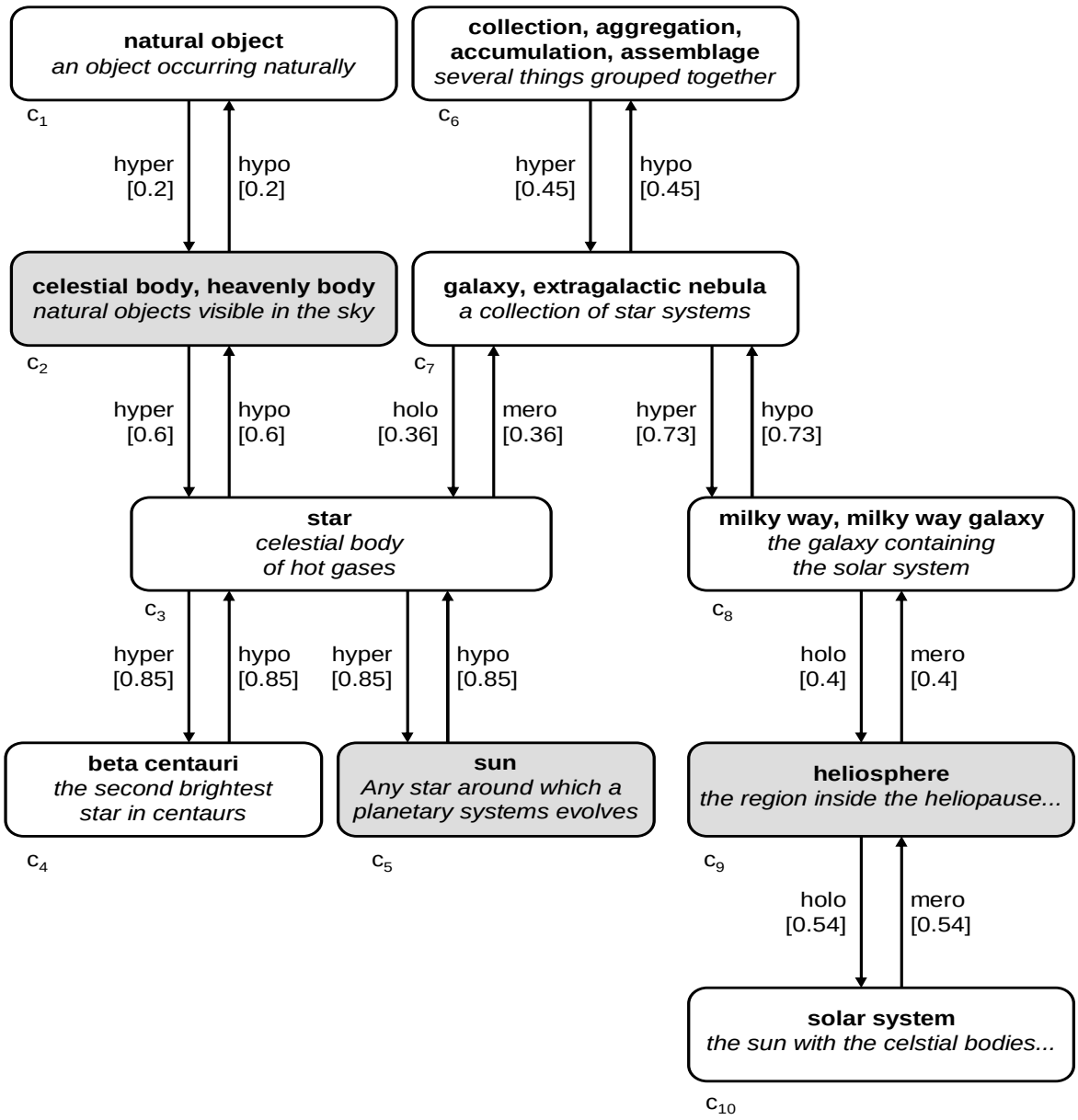


Abbildung 3.5: Pfade zwischen zwei Knoten in der Ontologie

Gemäß obiger Festlegungen gibt es in diesem Beispiel nur je einen zyklenfreien Pfad $p_1 = \langle c_2.c_3.c_5 \rangle$ bzw. $p_2 = \langle c_2.c_3.c_7.c_8.c_9 \rangle$, deren Pfadähnlichkeiten berechnet werden müssen. Damit drücken die Pfadähnlichkeiten automatisch die Ähnlichkeit der Konzepte aus, so dass $sim(c_2, c_5) = 0,51$ bzw. $sim(c_2, c_9) = 0,05$ ist.

3.6 Berechnung semantisch ähnlicher Konzepte

Sei $O = (V, E, \mathcal{Z}_B)$ eine Ontologie bestehend aus den Konzepten aus $\mathcal{C}$. Zu einem gegebenen Konzept $s \in V$ ist ein Konzept $v \in V$ semantisch ähnlich, falls es einen Pfad von s nach v in O gibt und die semantische Ähnlichkeit $sim(s, v) > 0$ ist. Die Berechnung aller semantisch ähnlichen Konzepte zu einem gegebenen Konzept bezeichnen wir als *"Single–Source Best Path Problem"* (kurz: SSBP–Problem).

Das SSBP–Problem beschreibt die Suche nach Pfaden mit maximaler Pfadähnlichkeit von einem Ausgangsknoten s zu jedem anderen erreichbaren Knoten v im Graphen O.

Die Pfadähnlichkeit eines Pfades ergibt sich aus der Multiplikation der Kantengewichte (Definition 3.13); ein Pfad mit maximaler Pfadähnlichkeit (Definition 3.14) wird *bester Pfad* genannt.

Zur Lösung dieses Problems verwenden wir die bekannte Lösung eines sehr ähnlichen Problems als Vorlage. Das *„Single–Source Shortest Path Problem“* (kurz: SSSP–Problem) beschreibt die Suche nach Pfaden mit minimalen Pfadgewichten von einem Ausgangsknoten s zu jedem erreichbaren Knoten v im Graphen O. Das Gewicht eines Pfades ergibt sich dabei aus der Summe der Kantengewichte; ein Pfad mit minimalem Gewicht wird *kürzester Pfad* genannt. Das SSSP–Problem lässt sich für einen gerichteten Graphen mit positiven Kantengewichten mit Hilfe des Dijkstra–Algorithmus [Dij59, CLR90] lösen.

Im Rahmen dieser Arbeit verändern wir den Dijkstra–Algorithmus zur Lösung des SSSP–Problems so, dass man mit dem modifizierten Dijkstra–Algorithmus das SSBP–Problem lösen kann. Dazu erläutern wir zunächst den Ablauf des modifizierten Dijkstra–Algorithmus. Danach geben wir die Beweise für zunächst postulierte Annahme und für die Korrektheit des Algorithmus an. Abschließend werden wir die Komplexität des modifizierten Dijkstra–Algorithmus untersuchen.

3.6.1 Modifizierter Dijkstra–Algorithmus

Der Dijkstra–Algorithmus berechnet für einen gegebenen gerichteten, gewichteten Graphen mit nicht negativen Kantengewichten die kürzesten Pfade von einem gegebenen Ausgangsknoten zu allen erreichbaren Knoten auf der Basis der summierten Kantengewichte als Maß für die Pfadlänge [CLR90].

Der modifizierte Dijkastra–Algorithmus (kurz: MDA) soll für einen gegebenen gerichteten, gewichteten Graphen mit Kantengewichten zwischen 0 und 1 die besten Pfade von einem gegebenen Ausgangsknoten zu allen erreichbaren Knoten auf der Basis der multiplizierten Kantengewichte als Maß für die Pfadähnlichkeit bestimmen.

Sei die Ontologie $O = (V, E)$ ein gerichteter, markierter Graph mit einer Kantengewichtung $weight : E \rightarrow [0, 1]$ und sei $s \in V$ der Ausgangsknoten. Sei weiterhin die Pfadähnlichkeit durch die Multiplikation der Kantengewichte (Definitionen 3.13 und 3.14) gegeben. Für jeden Knoten $v \in V$ definieren wir den BP–Schätzwert $est[v] \in [0, 1]$ als Schätzwert für die Pfadähnlichkeit eines besten Pfades von s nach v (best–path estimate).

Die *Initialisierung* der BP–Schätzwerte des Graphen O findet zu Beginn des MDA statt und weist jedem Knoten des Graphen entsprechend der angegebenen Prozedur Init() einen anfänglichen BP–Schätzwert zu.

INIT(O,s)
(1) **foreach** $(v \in V)$ **do** (2) $est[v] = 0$ (3) **od** (4) $est[s] = 1$

Nach der Initialisierung hat der Ausgangsknoten s einen BP–Schätzwert von 1, das ist der maximal mögliche Wert, und alle anderen Knoten haben den BP–Schätzwert 0, das ist der minimal mögliche Wert.

Die *Relaxation einer Kante* $(u, v) \in E$ prüft, ob der bisher gefundene beste Pfad $p = \langle s \ldots v \rangle$ mit der Pfadähnlichkeit $sim_p(s, v) = est[v]$ durch den Pfad $p' = \langle s \ldots uv \rangle$, der als letztes über den Knoten u läuft, bevor er v erreicht, mit der Pfadähnlichkeit $sim_{p'}(s, v)$ verbessert werden kann (siehe Abb. 3.6).

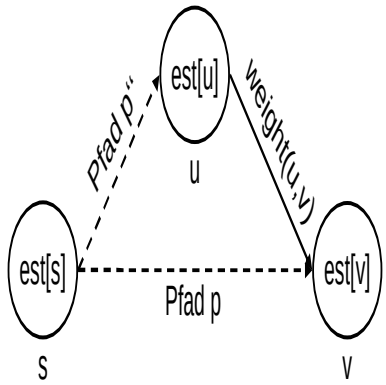


Abbildung 3.6: Relaxationsschritt

Falls also $sim_{p'}(s, v) > sim_p(s, v)$ ist, wird der BP–Schätzwert von v mittels $est[v] = sim_{p'}(s, v)$ aktualisiert. Die folgende Prozedur fasst einen Relaxationsschritt zusammen.

RELAX(u,v,weight)
(1) **if** $(est[v] < est[u] \cdot weight(u, v))$ **then** (2) $est[v] = est[u] \cdot weight(u, v)$ (3) **fi**

Die *Traversierung* des Graphen und die *Terminierung* des MDA stützt sich auf die noch zu beweisende Annahme, dass ein Teilpfad eines besten Pfades selbst ein bester Pfad ist. Die Traversierung wird durch eine Prioritätenwarteschlange (priority queue) Q realisiert. In ihr werden die Knoten absteigend nach ihrem BP–Schätzwert sortiert bereit gehalten. Der nächste Relaxationsschritt wird bei dem Knoten u mit dem maximalen BP–Schätzwert durchgeführt, wobei dieser aus der Warteschlange Q mit Hilfe einer Prozedur EXTRACT_MAX() entfernt wird. Die Relaxation wird auf alle vom aktuellen Knoten u ausgehenden Kanten angewendet. Ist die Warteschlange leer, terminiert der

MDA, und es stehen die maximalen Pfadähnlichkeiten als BP–Schätzwerte der Knoten v zur Verfügung.

Im folgenden geben wir den vollständigen MDA an, wobei $adj[u] \subseteq V$ die Menge der Knoten $v \in V$ ist, die durch eine Kante $(u, v) \in E$ mit dem Knoten u verbunden sind.

```
        ModifiedDijkstra(O,s)

(1)  Init(O,s)
(2)  S = ∅                                  // set of visited nodes
(3)  Q = V                                  // priority queue sorted by est[v]
(4)  while (Q ≠ ∅) do
(5)    u = Extract_Max(Q)                   // node u with maximum est[u]
(6)    S = S ∪ {u}
(7)    foreach (v ∈ adj[u]) do              // for each neighbor node v ...
(8)      Relax(u,v,weight)
(9)    od
(10) od
```

Zum besseren Verständnis demonstrieren wir das Verhalten des oben beschriebenen Algorithmus anhand eines überschaubaren gerichteten und gewichteten Graphen.

Beispiel 3.9 Die Abbildung 3.7 illustriert die Funktionsweise des MDA zur Lösung des SSBP–Problems anhand eines Beispielgraphen. Dabei wird in jedem Schritt (ii)–(vii) der Knoten u dunkel schattiert, der aus der Prioritätenwarteschlange entnommen wurde. Die schattierten Knoten sind nicht mehr in der Prioritätenwarteschlange enthalten. Die fettgedruckten gerichteten Kanten symbolisieren die aktuell besten Pfade.

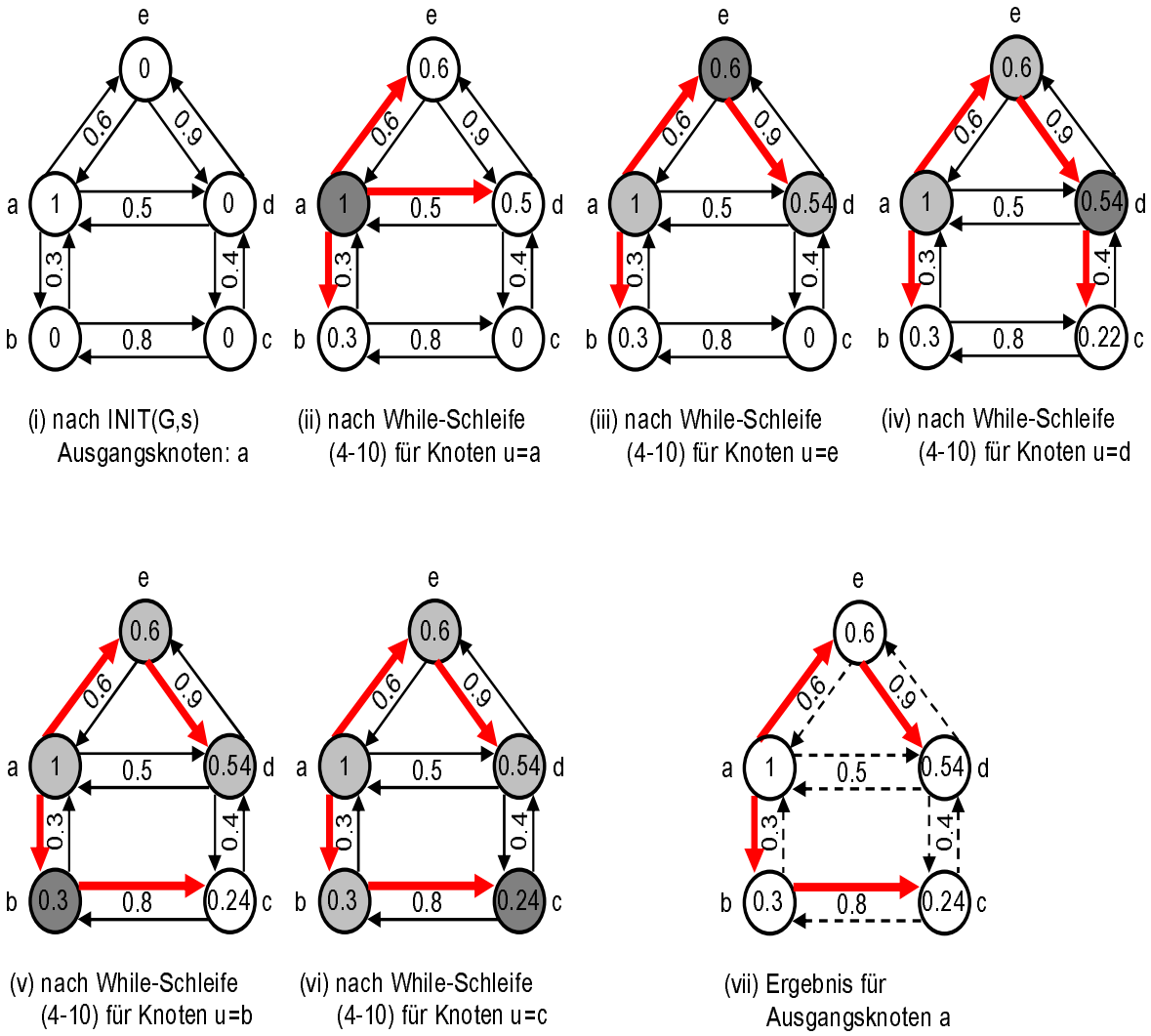


Abbildung 3.7: modifizierter Dijkstra–Algorithmus (MDA) zur Lösung des SSBP–Problems

3.6.2 Korrektheit des modifizierten Dijkstra–Algorithmus

Wir fassen zunächst die wesentlichen Unterschiede zwischen dem SSBP– und dem SSSP–Problem in einer Tabelle zusammen, da die Korrektheit des Dijkstra–Algorithmus zur Lösung des SSSP–Problems bekannt ist [CLR90] und die Modifikationen für den Korrektheitsbeweis für den modifizierten Dijkstra–Algorithmus (MDA) entscheidend sind.

MDA	DA
Pfadähnlichkeit vs. Pfadgewicht	
$sim_p(u,v) = \prod_{i=1}^{k} weight(v_{i-1}, v_i)$	$pathweight_p = \sum_{i=1}^{k} weight(v_{i-1}, v_i)$
bester Pfad vs. kürzester Pfad	
$sim(u,v) = \begin{cases} max\{sim_p(u,v)\} & \text{falls } p \text{ existiert} \\ 1.0 & \text{falls } u = v \\ 0 & \text{sonst} \end{cases}$	$pathweight(u,v) = \begin{cases} \min\{pathweight_p(u,v)\} & \text{falls } p \text{ existiert} \\ \infty & \text{sonst} \end{cases}$
Initialisierung	
$\forall v \in V : est[v] = 0$ $est[s] = 1$	$\forall v \in V : est[v] = \infty$ $est[s] = 0$
Relaxationsschritt	
falls $est[v] < est[u] \cdot weight(u,v)$ dann $est[v] = est[u] \cdot weight(u,v)$	falls $est[v] > est[u] + weight(u,v)$ dann $est[v] = est[u] + weight(u,v)$
Dijkstra–While–Schleife	
u = EXTRACT_MAX(Q)	u = EXTRACT_MIN(Q)

Tabelle 3.8: Unterschiede zwischen modifiziertem Dijkstra–Algorithmus (MDA) und dem ursprünglichen Dijkstra–Algorithmus

Um die Korrektheit des MDA zeigen zu können, müssen wir die oben gemachte Aussage über beste Teilpfade sowie das korrekte Verhalten der Prozeduren INIT() und RELAX() beweisen.

3.6.2.1 Optimale Teilpfade eines besten Pfades

Der modifizierte Dijkstra–Algorithmus stützt sich auf die Aussage, dass ein Teilpfad eines besten Pfades ebenfalls ein bester Pfad ist. Diese Aussage gilt es nun zu beweisen.

Lemma 3.2
Sei $p = \langle v_0 \dots v_k \rangle$ ein bester Pfad mit einer Pfadähnlichkeit $sim(v_0, v_k)$. Sei $p_{ij} = \langle v_i v_{i+1} \dots v_j \rangle$ ein Teilpfad von p für beliebige i und j ($0 \leq i \leq j \leq k$). Dann ist p_{ij} ein bester Pfad mit Pfadähnlichkeit $sim(v_i, v_j)$.

Beweis:
Der Pfad p lässt sich in die Teilpfade $p_{0i} = \langle v_0 \dots v_i \rangle$, $p_{ij} = \langle v_i \dots v_j \rangle$, $p_{jk} = \langle v_j \dots v_k \rangle$ zerlegen, so dass $p = p_{0i} p_{ij} p_{jk}$. Aus den Definitionen 3.13 und 3.14 ergibt sich die Pfadähnlichkeit von p aus $sim_p(v_0, v_k) = sim_{p_{0i}}(v_0, v_i) \cdot sim_{p_{ij}}(v_i, v_j) \cdot sim_{p_{jk}}(v_j, v_k)$.

Wir nehmen nun an, dass es einen Pfad $p'_{ij} = \langle v_i \dots v_j \rangle$ gibt, für den $sim_{p'_{ij}}(v_i, v_j) > sim_{p_{ij}}(v_i, v_j)$ gilt, der also besser ist als p_{ij}.

Dann ergibt sich die Pfadähnlichkeit für den zerlegten Pfad $p = p_{0i} p'_{ij} p_{jk}$ aus den Pfadähnlichkeiten der beteiligten Teilpfade $sim_{p_{0i}}(v_0, v_i) \cdot sim_{p'_{ij}}(v_i, v_j) \cdot sim_{p_{jk}}(v_j, v_k) > sim_p(v_0, v_k)$.

Das ist ein Widerspruch zu der Voraussetzung, dass der Pfad p ein bester Pfad ist und somit seine Pfadähnlichkeit maximal ist. □

Das folgende Korollar präzisiert diese Eigenschaft für Pfade und Teilpfade mit gleichem Startknoten s, dem gegebenen Ausgangsknoten.

Korollar 3.1
Sei $p = \langle s \dots v \rangle$ ein bester Pfad mit der Pfadähnlichkeit sim(s,v). Sei p zerlegbar in den Pfad $p' = \langle s \dots u \rangle$ und die Kante (u,v). Dann gilt für die Pfadähnlichkeit von p $sim(s, v) = sim(s, u) \cdot weight(u, v)$. □

Beweis:
Aufgrund des Lemmas 3.2 ist klar, dass der Teilpfad p' ein bester Pfad mit der Pfadähnlichkeit $sim(s, u)$ ist. Folglich ist $sim(s, v) = sim(s, u) \cdot weight(u, v)$. □

Das nun folgende Lemma gibt eine einfache, aber nützliche Eigenschaft der Ähnlichkeit bester Pfade an.

Lemma 3.3
Sei $s \in V$ ein Ausgangsknoten. Dann gilt für alle Kanten $(u, v) \in E$: $sim(s, v) \geq sim(s, u) \cdot weight(u, v)$. □

Beweis:
Ein bester Pfad p von einem Ausgangsknoten s zu einem Knoten v hat keine geringere Pfadähnlichkeit als irgendein anderer Pfad von s nach v.

Insbesondere gilt, dass der Pfad p keine geringere Pfadähnlichkeit als der Pfad hat, der aus dem besten Pfad von s nach u besteht und dann die Kante (u, v) hinzunimmt. □

3.6.2.2 Initialisierung und Relaxation von Kanten

Für jeden Knoten $v \in V$ definieren wir einen Schätzwert $est[v]$, der eine untere Schranke für die semantische Ähnlichkeit, also für die Pfadähnlichkeit eines besten Pfades von einem Ausgangsknoten s zu einem Knoten v darstellt. In einem Relaxationsschritt wird der aktuelle Pfad um die Kante (u, v) verlängert und dabei überprüft, ob der bisherige Schätzwert $est[v]$ durch die aktuelle Pfadähnlichkeit $sim_p(s, v)$ verbessert wird.

Die Korrektheit des modifizierten Dijkstra–Algorithmus hängt von den bedeutenden Eigenschaften der Relaxation von Kanten ab. Die folgenden Lemmata fassen diese Eigenschaften zusammen.

Lemma 3.4
Sei $(u, v) \in E$. Dann gilt direkt nach der Relaxation der Kante (u,v) durch die Prozedur RELAX*(u,v,weight) für den BP–Schätzwert est[v] für die beste Pfadähnlichkeit:*

$$est[v] \geq est[u] \cdot weight(u, v).$$

□

Beweis:
Falls wir vor der Relaxation der Kante (u, v) den Zustand $est[v] < est[u] \cdot weight(u, v)$ haben, dann gilt nach der Relaxation $est[v] = est[u] \cdot weight(u, v)$.

Falls wir jedoch vor der Relaxation der Kante (u, v) den Zustand $est[v] \geq est[u] \cdot weight(u, v)$ haben, dann bleibt $est[v]$ unverändert. □

Lemma 3.5
Sei $s \in V$ der Ausgangsknoten und seien die BP–Schätzwerte des Graphen initialisiert durch INIT*(O,s). Dann gilt $\forall v \in V$: $est[v] \leq sim(s, v)$. Diese Invariante bleibt über jede Folge von Relaxationsschritten erhalten. Außerdem wird der Wert est[v] nicht mehr verändert, sobald er einmal die obere Schranke sim(s,v) erreicht hat.* □

Beweis:
Die Invariante $est[v] \leq sim(s,v)$ ist offensichtlich wahr nach der Initialisierung, denn $est[s] = 1 = sim(s,s)$ und $est[v] = 0$ impliziert, dass $est[v] \leq sim(s,v) \in [0,1]$ für alle $v \in V \backslash \{s\}$ ist.

Sei v der erste Knoten, für den die Relaxation einer Kante (u,v) zu $est[v] > sim(s,v)$ führt. Dann erhalten wir nach der Relaxation der Kante (u,v):

$$\begin{aligned} est[u] \cdot weight(u,v) &= est[v] \\ &> sim(s,v) \\ &\geq sim(s,u) \cdot weight(u,v) \text{ wegen Lemma 3.3} \end{aligned}$$

Dieser Zusammenhang impliziert, dass $est[u] > sim(s,u)$ ist. Da in der Relaxation der Kante (u,v) der BP–Schätzwert $est[u]$ nicht verändert wird, muß diese Ungleichung bereits vor der Relaxation gelten, also $est[u] > sim(s,u)$. Das ist aber ein Widerspruch zu der Annahme, dass v der erste Knoten ist, für den die Verletzung der Invariante gilt. Folglich bleibt die Invariante $est[v] \leq sim(s,v)$ für alle Knoten $v \in V$ während der Relaxation erhalten.

Es ist leicht einzusehen, dass der BP–Schätzwert $est[v]$ nicht mehr verändert wird, sobald er einmal die obere Schranke $est[v] = sim(s,v)$ erreicht hat. Der BP–Schätzwert $est[v]$ kann durch keinen Relaxationsschritt weiter erhöht werden, da wir gezeigt haben, dass $est[v] \leq sim(s,v)$. □

Korollar 3.2
Sei $v \in V$ ein Knoten, für den es keinen Pfad vom Ausgangsknoten s nach v in O gibt. Dann gilt nach der Initialisierung durch die Prozedur INIT*(O,s), dass $est[v] = sim(s,v) = 0$. Diese Gleichung bleibt invariant über jede Folge von Relaxationen von Kanten in O.* □

Beweis:
Wegen Lemma 3.5 gilt nach der Initialisierung $0 = sim(s,v) \geq est[v]$. Da jede Relexation $est[v]$ nicht verändert, bleibt $est[v] = 0 = sim(s,v)$, unabhängig von irgendwelchen Relaxationen. □

Das folgende Lemma ist entscheidend für den Korrektheitsbeweis. Es liefert die Gründe dafür, dass sich durch die Relaxation von Kanten der BP–Schätzwert eines Knotens v dem Wert der maximalen Pfadähnlichkeit stetig annähert und zum Ende des Algorithmus den Wert der maximalen Pfadähnlichkeit angenommen hat.

Lemma 3.6
Sei $s \in V$ ein Ausgangsknoten und sei $p = \langle s \ldots uv \rangle$ ein bester Pfad in O für zwei Knoten $u, v \in V$. Wir nehmen an, dass die BP–Schätzwerte des Graphen O durch die Prozedur INIT*(O,s) initialisiert wurde und dann eine Folge von Relaxationen von Kanten in O einschließlich der Relaxation der Kante (u,v) durch* RELAX*(u,v,weight) durchgeführt wurde. Falls $est[u] = sim(s,u)$ irgendwann vor der Relaxation der Kante (u,v) erreicht wurde, dann gilt $est[v] = sim(s,v)$ zu jeder Zeit nach der Relaxation der Kante (u,v).* □

Beweis:
Wegen Lemma 3.5 gilt, falls $est[u] = sim(s,u)$ vor der Relaxation der Kante (u,v) erreicht wird, dann gilt diese Gleichung auch nach der Relaxation der Kante (u,v). Insbesondere gilt nach der Relaxation der Kante (u,v):

$$\begin{aligned} est[v] &\geq est[u] \cdot weight(u,v) && \text{wegen Lemma 3.4} \\ &= sim(s,u) \cdot weight(u,v) \\ &= sim(s,v) && \text{wegen Korollar 3.1} \end{aligned}$$

Durch das Lemma 3.5 ist bekannt, dass sich $est[v]$ von unten an $sim(s,v)$ annähert. Daraus können wir schließen, dass $est[v] = sim(s,v)$ ist und die Gleichung nach der Relaxation erhalten bleibt. □

3.6.2.3 Korrektheit des modifizierten Dijkstra–Algorithmus

In dem folgenden Satz wird die Korrektheit des modifizierten Dijkstra–Algorithmus formuliert und anschließend bewiesen.

Satz 3.3
Wird der modifizierte Dijkstra–Algorithmus auf einer Ontologie $O = (V, E)$ und einem Ausgangsknoten $s \in V$ ausgeführt, dann gilt nach Beendigung des Algorithmus für alle Knoten $u \in V : est[u] = sim(s,u)$. □

Beweis:
Sei $u \in V$ der erste Knoten, für den der BP–Schätzwert zum Zeitpunkt der Übernahme des Knotens u aus der Prioritätenwarteschlange Q in die Menge S einen Wert $est[u] \neq sim(s,u)$ hat.

Die Knoten s und u sind verschieden, da s der erste Knoten ist, der in die Menge S aufgenommen wird und für den durch die Initialisierung zu diesem Zeitpunkt $est[s] = sim(s,s) = 1$ gilt. Folglich gilt für die Menge S: $S \neq \emptyset$.

Gibt es keinen Pfad von s nach u, dann ist nach Korollar 3.2 $est[u] = sim(s,u) = 0$, was einen Widerspruch zur Auswahl von u darstellt.

Gibt es hingegen einen Pfad von s nach u, dann gibt es insbesondere einen besten Pfad p von s nach u, wobei $s \in S$ und $u \in V \backslash S$. Sei y der erste Knoten auf dem Pfad p, so dass $y \in V \backslash S$ und sei x der Vorgänger von y auf p. Folglich kann der Pfad p zerlegt werden in $p_1 = \langle s \ldots x \rangle$, $p_2 = \langle xy \rangle = (x,y)$ und $p_3 = \langle y \ldots u \rangle$, so dass $p = p_1 p_2 p_3$. Wegen obiger Annahme muss für den Knoten x gelten, dass $est[x] = sim(s,x)$ zum Zeit der Übernahme von x aus Q in die Menge S. Es folgt die Relaxation der Kante (x,y) und damit gilt die Gleichung $est[u] = sim(s,u)$ wegen des Lemmas 3.6. Da y vor u auf dem Pfad nach u liegt und alle Kantengewichte zwischen 0 und 1 liegen, gilt die Ungleichung $sim(s,y) \geq sim(s,u)$. Wegen $est[y] = sim(s,y)$ und wegen des Lemmas 3.5 gilt die Ungleichung $est[y] \geq est[u]$. Da $u, y \in V \backslash S$ zum Zeitpunkt der Wahl von u als nächsten Knoten, der aus Q in die Menge S übernommen werden soll, muss $est[u] \geq est[y]$ zu diesem Zeitpunkt gelten. Dann ergibt sich aber für obige Ungleichung $est[y] = sim(s,y) = sim(s,u) = est[u]$. Folglich ist $est[u] = sim(s,u)$, was einen Widerspruch zur obigen Annahme darstellt. □

3.6.3 Komplexität des modifizierten Dijkstra–Algorithmus

Die Modifikationen des Dijkstra–Algorithmus beschränken sich auf Veränderungen der Berechnungs– und Vergleichsfestlegungen. Wir haben keine Veränderungen im algorithmischen Ablauf vorgenommen und können uns daher an den Komplexitätsbetrachtungen in [CLR90] orientieren.

Sei für den Graphen $O = (V, E, \mathcal{Z}_B)$ $n =\mid V \mid$ die Anzahl der Knoten und $m =\mid E \mid$ die Anzahl der Kanten.

Der Gesamtalgorithmus hat eine Laufzeit von $O(n^2+m) = O(n^2)$, falls die Prioritätenwarteschlange als lineares Array realisiert wird.

Die Laufzeit des Gesamtalgorithmus verbessert sich auf $O((n + m)log\ n)$, falls die Prioritätenwarteschlange als binärer Heap realisiert wird.

3.7 Aktualisierung der Ontologie

Die Konstruktion und die Erweiterung einer quantifizierten Ontologie gemäß der Definition 3.9 ergibt sich aus dem Einfügen neuer Konzepte. Dazu werden XML–Dokumente oder XML–Anfragen als Stichwortlieferanten verwendet.

Im Rahmen dieser Arbeit verfolgen wir dabei folgenden Ansatz. Das Einfügen eines neuen Konzepts umfasst eine Vorbereitungsphase und eine Einfügephase. In der Vorbereitungsphase wird ein gegebenes Wort, das aus einem XML–Dokument oder einer XML–Anfrage stammt, einem oder mehreren Konzepten mit Hilfe einer umfangreichen statischen, ontologischen Wissensbasis (z.B. WordNet) zugeordnet. Mit Hilfe eben dieser Wissensbasis können verwandte Begriffe in der Ontologie lokalisiert und deren semantische Beziehung zum Ausgangsbegriff ermittelt werden. Auf der Basis des Webs kann mit Hilfe einer Suchmaschine (z.B. Google) die Korrelation zwischen den neuen und einem existierenden verwandten Begriff berechnet und in die Ontologie eingetragen werden.

3.7.1 Vorbereitungsphase

In der Vorbereitungsphase erfolgt die Zuordnung von Bedeutungen zu einem gegebenen Wort $w \in \Sigma^*$. Dabei muss festgelegt werden, welche Konzepte an die Einfügephase zum Einfügen in die Ontologie übergeben werden. Dabei stehen zwei Fragen im Vordergrund.

1. Welche Bedeutungen gibt es für das Wort w?
2. Welche Bedeutung hat das Wort w unter Berücksichtigung seiner Herkunft?

Die erste Frage lässt sich mit Hilfe von WordNet beantworten. Hierbei werden die Bedeutungen für w im Polyset $polyset(w)$ zusammengefasst.

Die zweite Frage dient zur Entscheidungsfindung, für welche Bedeutungen von w neue Konzepte in die Ontologie eingefügt werden sollen. Dazu gibt es verschiedene Möglichkeiten.

Eine Möglichkeit, die Bedeutung eines Wortes w zu ermitteln, basiert auf dem Vergleich der Quelle des Wortes mit dem Kontext $con(c)$ eines potentiell geeigneten Konzepts $c = (s, b)$. Diese Vorgehensweise wird auch *Desambiguierung* genannt. Mit Hilfe von Methoden aus der Linguistik und dem Information Retrieval werden weitere Wörter aus der Quelle extrahiert und als Kontext des Wortes w bzgl. der Quelle interpretiert. Diese Wörter können mittels Verfahren aus der Statistik mit den Kontexten potentiell geeigneter Konzepte verglichen werden, so dass letztendlich eine Zuordnung zu wenigen Begriffen, idealerweise zu einem Begriff möglich wird. Hier muß u.U. die Definition des

Kontextes eines Konzepts (Abschnitt 3.3) neu überdacht werden, da die Gleichheit von Kontexten von Polysemen die Desambiguierung erschwert.

Im Rahmen dieser Arbeit verwenden wir eine andere Möglichkeit der Desambiguierung. Wir lösen die Mehrdeutigkeit eines gegebenen Wortes w nicht explizit auf, sondern fügen alle potentiell relevanten Konzepte $c = (s, b)$ sowie deren direkt verwandte Konzepte in die Ontologie ein.

3.7.2 Einfügephase

Wir nehmen an, dass $O_W = (V_W, E_W, \mathcal{Z}_B)$ die WordNet–Ontologie ist und die betrachteten Konzepte in der WordNet–Ontologie vorkommen. Direkt verwandt zu einem gegebenen Konzept $c \in V_W$ sind alle Konzepte $c' \in \mathcal{C}$, für die es in WordNet die Kanten von $(c, c') \in E_W, f_{E_W}(c, c') = t$ und $(c', c) \in E_W, f_{E_W}(c', c) = t'$ in der WordNet–Ontologier gibt.

Das Konzept c und seine verwandten Konzepte c' werden einzeln und unabhängig voneinander in die Ontologie eingefügt. Dabei geht die Einfügeroutine in drei Schritten vor. Als Erstes wird für das einzufügende Konzept ein neuer Knoten erzeugt, falls er noch nicht in der Ontologie vorkommt. Im zweiten Schritt werden alle Kanten zu direkt verwandten Konzepten, die bereits in der Ontologie vorkommen, eingetragen. Das Wissen über die direkte Verwandtschaft wird der WordNet–Ontologie entnommen. Abschließend werden die Kantengewichte für die neu erzeugten Kanten berechnet.

Sei $c = (s, b) \in \mathcal{C}$ ein neues Konzept, das in die bestehende Ontologie $O = (V, E, \mathcal{Z}_B)$ eingefügt werden soll. Im Rahmen dieser Arbeit wird jedes Konzept einzeln gemäß des folgenden Algorithmus eingefügt, wobei die Prozedur COMPUTE_WEIGHT() das Kantengewicht für eine gegebene Kante gemäß der Definition 3.12 berechnet.

INSERT(O, O_W, u, weight)

```
(1)  if (u ∉ V) then
(2)     V = V ∪ {u}
(3)     forall ((u, u') ∈ E_W) do
(4)        e = (u, u'); f_{E_W}(e) = t
(5)        e' = (u', u, ); f_{E_W}(e') = t'
(6)        if (u' ∈ V) then
(7)           E = E ∪ {e, e'}
(8)           f_E(e) = f_{E_W}(e)
(9)           f_E(e') = f_{E_W}(e')
(10)          weight(u,u')=COMPUTE_WEIGHT(e)
(11)          weight(u',u)=weight(e)
(12)       fi
(13)    od
(14) fi
```

Beispiel 3.10 Anhand eines Beispiels soll das Einfügen eines Konzepts demonstriert werden. Seien die Ontologie aus Abb. 3.2 und das neue Konzept *c=({ universe, cosmos}, . . . a whole collection of existing things . . .)* gegeben. Da der (schattierte) Knoten noch nicht vorkam, wurde er der Knotenmenge V hinzugefügt (siehe Abb. 3.8).

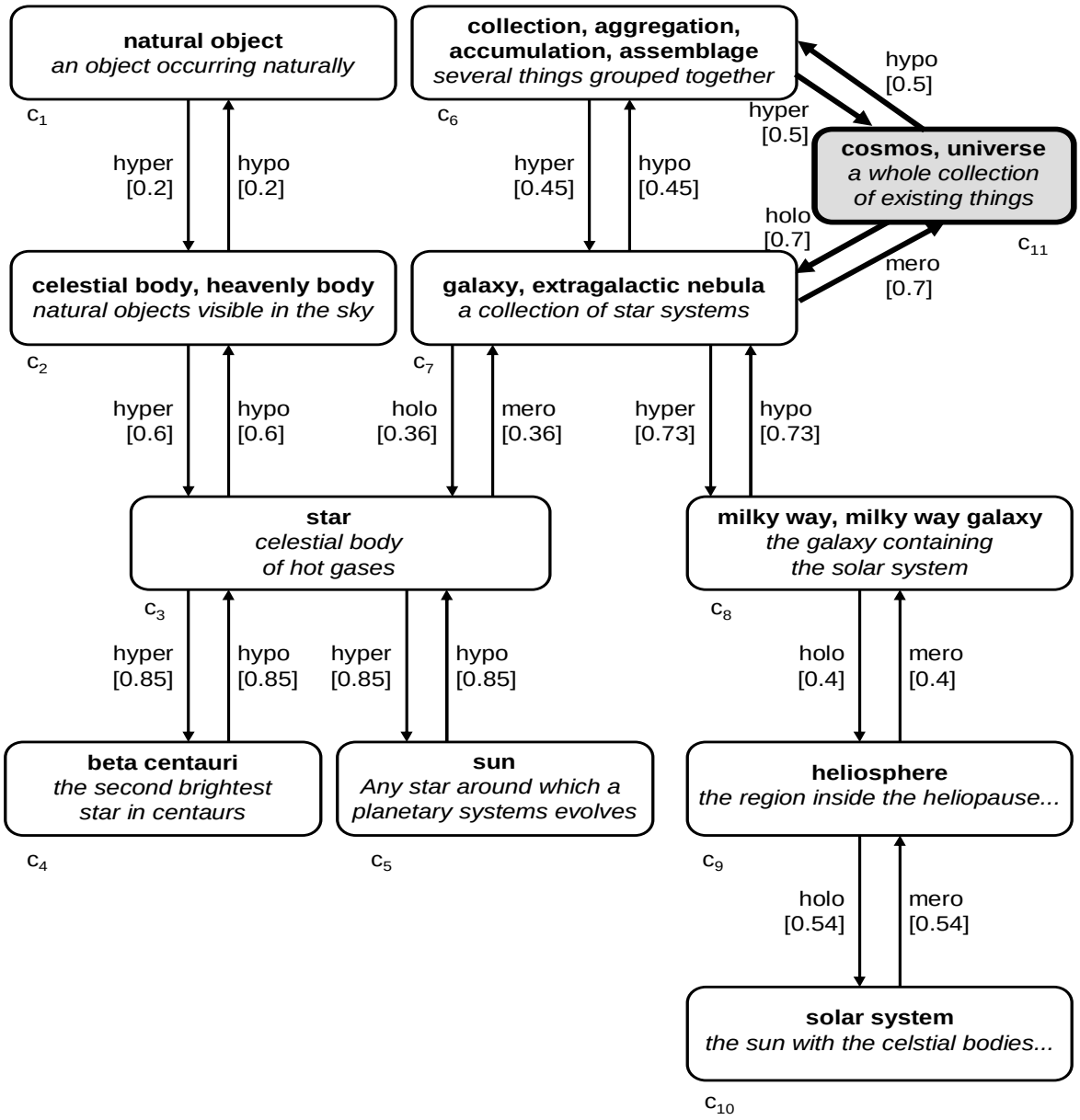


Abbildung 3.8: Ontologie aus Abb. 3.2 nach dem Einfügen

Zu diesem Knoten werden alle fett gedruckten Kanten zu Knoten in O gesetzt, die aus O_W bekannt sind. Danach werden in gleicher Art und Weise die zu c verwandten Konzepte c' in die Ontologie O aufgenommen. In diesem Beispiel gibt es keine weiteren verwandten Konzepte, die einzufügen wären.

3.8 Präsentation der Ontologie

Wenn ausreichend Platz auf dem Bildschirm oder dem Papierausdruck zur Verfügung steht, dann ist die Präsentation einer Ontologie zur Ansicht für den Nutzer kein Problem. Da dies im Allgemeinen nicht gegeben ist, verfolgen wir neben der ausschnittsweisen Anzeige der Ontologie das Ziel, die Bedeutung eines Wortes so kompakt wie möglich zu repräsentieren und die Anzahl der sichtbaren Kanten zu reduzieren.

Im Rahmen dieser Arbeit gehen wir dabei wie folgt vor.

1. Zur Präsentation einer Ontologie genügt es, jeweils eine Richtung der Kanten explizit zu modellieren. Die Gegenrichtung ist durch den Satz 3.2 implizit gegeben.

2. Zur Repräsentation der Bedeutung eines Konzepts wählen wir ein geeignetes Wort des Kontextes als Erläuterung der Bedeutung aus.

Das Konzept $c = (s, b) \in V$ der Ontologie $O = (V, E, \mathcal{Z}_B)$ soll durch ein geeignetes Paar (s, w) präsentiert werden, wobei $w \in con(c)$. Da das Wort w die Bedeutung b repräsentieren soll, eignen sich übergeordnete Konzepte. Dabei muss darauf geachtet werden, dass die Paare von Polysemen $u_1 = (w, b_1), u_2 = (w, b_2), ...$ paarweise verschieden sind.

Für die Wahl der Wortpaare zu den darzustellenden Begriffen gibt es verschiedene Möglichkeiten. Verschiedene Methoden aus der Linguistik und dem Information Retrieval befassen sich mit dem Problem der Featureselektion zur Klassifikation von Dokumenten. In unserem Fall wären die Dokumente die Begriffskontexte.

Im Rahmen dieser Arbeit vertiefen wir dieses Problem nicht, sondern wählen der Einfachheit halber einen pragmatischen Ansatz. Wir wählen das lexikalisch kleinste, möglichst nicht zusammengesetzte Wort aus den Wörtern der Hypernyme als Repräsentanten für die Bedeutung eines Begriffs. Das kann bei Gleichheit der Kontexte (Abschnitt 3.3) in der Darstellung zu Mehrdeutigkeiten führen, die wir an dieser Stelle hinnehmen; wir gehen davon aus, dass benachbarte Ontologiebegriffe ausreichen, um die gleich dargestellten Begriffe zu unterscheiden.

Beispiel 3.11 Die erweiterte Ontologie aus Abb. 3.8 wird dann in folgender Weise dem Nutzer präsentiert. Hierbei wird ein Wortpaar (w, w') eines Begriffs $u = (w, b)$ durch die Formatierung $''\mathbf{w}[w']''$ dargestellt.

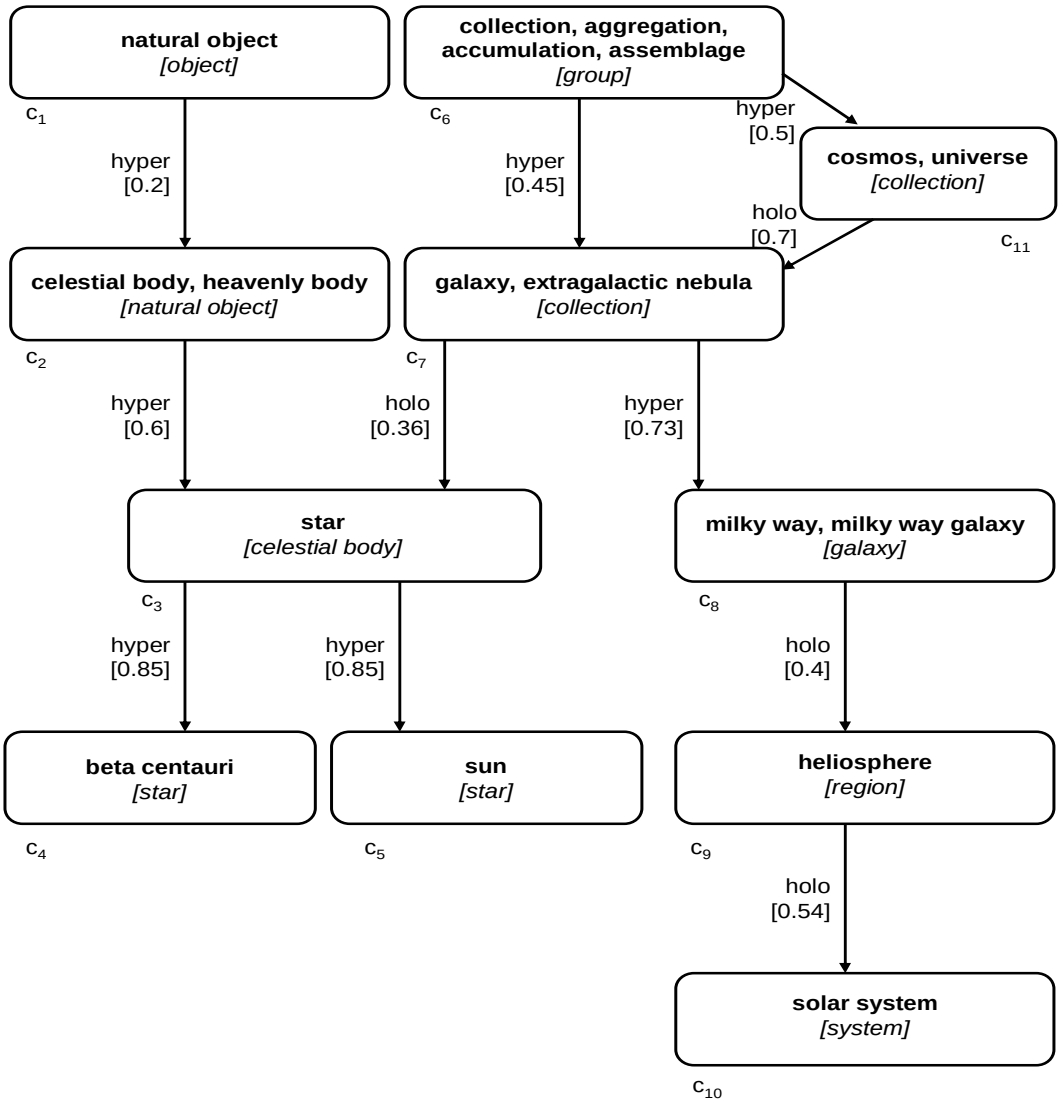


Abbildung 3.9: Präsentation der Ontologie O aus Abb. 3.8 für einen Nutzer

---

An dieser Stelle möchten wir die wichtigsten Aspekte diese Kapitels zusammenfassen.

Im Rahmen dieser Arbeit entwickeln wir ein IR–Modell, das die Suche von Benutzeranfragen nach Informationen in XML–Dokumenten unterstützt, wobei semantische Ähnlichkeit von gefundenen zu vorgegebenen Begriffen berücksichtigt werden soll. Dazu muss einerseits der Zusammenhang von Wort – Bedeutung – Begriff – Konzept und andererseits die Ordnung von Konzepten definiert werden.

Wörter gleicher Bedeutung werden in Synsets zusammenfasst. Ein Konzept ist ein Paar aus einem Synset und einer Bedeutung für die Wörter dieses Synsets. Semantische Beziehungen zwischen Konzepten basieren auf den semantischen Beziehungen zwischen den Begriffen, die sie repräsentieren. Diese lassen sich in durch einen Ontologiegraphen repräsentieren, wobei die Knoten die Konzepte darstellen und und die Kanten gerichtet und mit den entsprechenden semantischen Beziehungen beschriftet und semantischen Ähnlichkeiten gewichtet sind. Die Kantengewichte ergeben sich aus der Korrelation der beteiligten Konzepte auf der Basis der Korrelation zwischen ihren Kontexten unter Verwendung des Dice–Koeffizienten und einer Websuchmaschine.

Die semantische Ähnlichkeit zwischen zwei beliebigen Konzepten wird mit Hilfe des modifiziertem Dijkstra–Algorithmus berechnet und basiert auf der Multiplikation aller am Pfad beteiligten Kantengewichte, falls es diesen Pfad gibt.

Kapitel 4

XML–Anfragesprache XXL

XML–Dokumente zeichnen sich durch die flexible Kombination von Struktur– und Inhaltsdaten aus (siehe Kapitel 2). Um gezielte Informationssuche in XML–Dokumenten bzw. im XML–Graphen, der diese repräsentiert, zu ermöglichen, wird eine geeignete XML–Anfragesprache benötigt. XML–Anfragesprachen sind insbesondere in der Lage, Bedingungen an die Strukturdaten und Bedingungen an die Inhaltsdaten zu kombinieren und geeignet auszuwerten.

In diesem Kapitel führen wir die XML–Anfragesprache XXL ein; XXL steht für *Flexible XML Search Language*. Für den Entwurf von XXL spielten folgende Aspekte eine wesentliche Rolle:

- Orientierung an Veröffentlichungen des World–Wide–Web–Konsortiums (kurz: W3C), z.B. XML–QL, XQuery, XPath [DFF+98, BCF+03, BBC+03], wobei wir uns auf häufig genutzte Kerneigenschaften beschränken,
- einfache Lesbarkeit der Anfragen durch Orientierung an der weit verbreiteten Datenbankanfragesprache SQL [ANSI92] und Vermeidung unnötiger syntaktischer Notationen,
- Einführung eines neuen Operators '$\sim$' zum Ausdruck semantischer, ontologiebasierter Ähnlichkeitssuchbedingungen auf Namen und Werten von Elementen und Attributen sowie
- Erweiterbarkeit des Modells, so dass z.B. die Formulierung typspezifischer Ähnlichkeitssuchbedingungen (z.B. Orts–, Zeit–, Währungsangaben) definiert werden kann.

Diese Anfragesprache unterstützt die Suche nach Informationen im XML–Graphen. Mit Hilfe von XXL–Anfragen ist der Informationssuchende in der Lage, gezielt nach Informationen in den Struktur– und/oder in den Inhaltsdaten von XML–Dokumenten zu suchen. Darüber hinaus wird durch den Einsatz von Ontologien die semantische Ähnlichkeit von Begriffen in den Daten zu gegebenen Suchbegriffen bei der Informationssuche berücksichtigt.

Zur Suche in den Strukturdaten benötigen wir zum einen elementare Bedingungen an die Element– und Attributnamen, die in den n–Knoten des XML–Graphen stehen, und zum anderen strukturbasierte Bedingungen, also Bedingungen an die Anordnung dieser Elemente und Attribute. Zur Suche in den Inhaltsdaten benötigen wir elementare Bedingungen an die Element– und Attributwerte, wobei wir berücksichtigen müssen, dass Attributwerte in einem i–Knoten zu finden sind und Elementwerte typischer Weise über mehrere i–Knoten verteilt sind.

Das Ergebnis einer XXL–Anfrage besteht aus einer Rangliste von Teilgraphen des XML–Graphen, die absteigend nach ihrer Relevanz sortiert sind. Die Relevanzbewertung basiert auf einem probabilistischen Ansatz und stützt sich dabei zum einen auf den Ähnlichkeitsvergleich von vorgegebenen Suchbedingungen mit Konzepten der Ontologie und zum anderen auf die Häufigkeit von Suchbedingungen in den vorgegebenen Daten.

4.1 Überblick

Eine XXL–Anfrage im SQL–Stil besteht in Anlehnung an die bekannte Datenbankanfragesprache SQL [ANSI92] aus einer SELECT–, einer FROM– und einer WHERE–Klausel.

Die *SELECT–Klausel* definiert die Ausgabe. Mit Hilfe verschiedener Parameter wird die Ausgabe spezifiziert. Hierbei zeigt des Symbol $*$ die ungekürzte Ausgabe der Ergebnisse an. Das reservierte Wort `URLS` steht für die Ausgabe der URLs der relevanten Dokumente. Als dritte Variante besteht die Möglichkeit, Variablennamen aufzulisten, deren Belegung als Ergebnis in der Rangliste der Treffer ausgegeben werden soll.

Die *FROM–Klausel* spezifiziert den Suchraum. Aktuell wird ausschließlich in den Daten der gegebenen Indexstrukturen gesucht, was durch das reservierte Wort `INDEX` angegeben wird.

Die *WHERE–Klausel* beinhaltet die Anfragebedingungen (siehe Abb. 4.1). Sie besteht aus k konjunktiv verknüpften *XXL–Suchbedingungen* $W_1, \ldots, W_k$ $(0 < k < \infty)$. Dabei werden elementare Bedingungen an die Namen und Werte von Elementen und Attributen mit Hilfe von *Knoten–* bzw. *Inhaltsbedingungen* formuliert. Komplexe Bedingungen an die Struktur werden durch entsprechende *Pfadbedingungen* angegeben. Dabei kann der letzte Knoten eines zutreffendes Pfades mit Hilfe einer *Variablenbindung* an einen Variablennamen geknüpft werden. Variablenbindungen ermöglichen die Belegung von Variablen mit Knoten des XML–Graphen. *Variablenbelegungen* müssen von allen XXL–Suchbedingungen erfüllt werden, um zu einem gemeinsamen Ergebnis zu gelangen. Die konjunktive Abhängigkeit der XXL–Suchbedingungen definiert sich also über die gemeinsam verwendeten Variablen.

Beispiel 4.1 Zur Illustration geben wir im Folgenden die Suche nach Szenen im Drama "Macbeth", in denen eine Frau (engl.: woman | lady) über Macbeth als eine Führungsperson (engl.: Macbeth & leader) spricht, als XXL–Anfrage an.

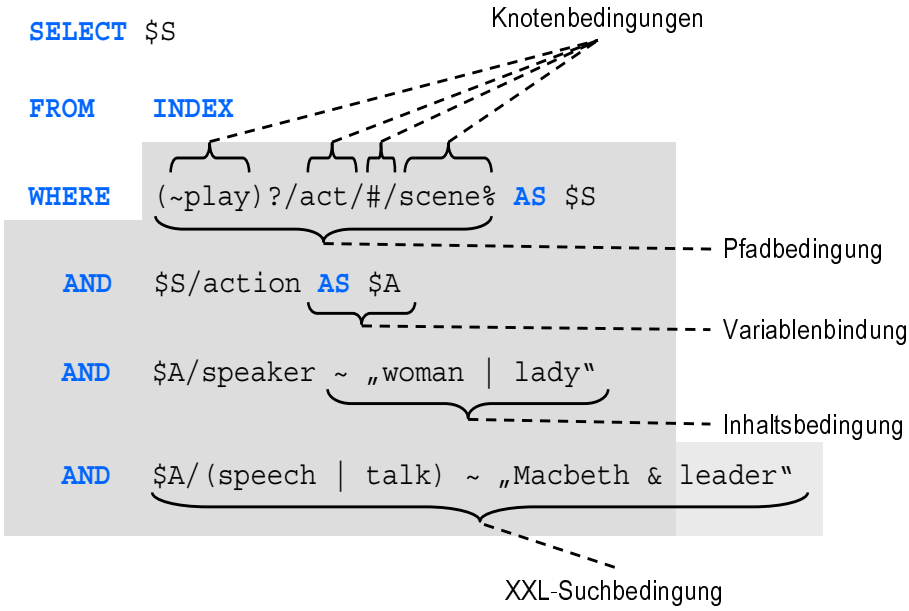


Abbildung 4.1: Beispiel für eine XXL-Anfrage

Groß geschriebene Wörter, z.B. `SELECT`, sind reservierte Wörter der Sprache XXL. Die Konkatenation von Knoten wird durch das `/`–Zeichen symbolisiert. Dabei steht `#` als Platzhalter für einen beliebigen Pfad. Das `?`–Zeichen gibt das optionale Vorkommen eines Pfadausdrucks im Ergebnis an, z.B. `~play`. Eine Variable wird durch das vorangestellte `$`–Zeichen gekennzeichnet und dient als Platzhalter für einen Knoten im XML–Graphen. Das reservierte Wort `AS` bindet die entsprechende Variable an den letzten Knoten des sich qualifizierenden Pfades im XML–Graphen. Der Ähnlichkeitsoperator ∼ kann sowohl unär auf Element– und Attributnamen, z.B. `~play`, als auch binär auf Element– und Attributwerte, z.B. `$B` ∼ "`Macbeth & leader`", angewendet werden.

Als Treffer werden die Teilgraphen im XML–Graphen gewertet, die mit den vorgegebenen Knotenbedingungen gemäß der formulierten Bedingungen exakt übereinstimmen oder semantisch ähnlich sind und dabei den Inhaltsbedingungen und Variablenbindungen genügen. Exakte Übereinstimmung liegt z.B. bei einem Treffer vor, in dem eine Frau (woman) über Macbeth als eine Führungsperson (leader) spricht. Ein semantisch ähnlicher Treffer wäre zum Beispiel ein Treffer, bei dem eine Hexe (witch) von Macbeth als Baron (thane) spricht.

Im Rahmen dieser Arbeit basiert die Syntax und Semantik einer XXL–Anfrage auf induktiven Definitionen der XXL–Suchbedingungen ihrer Where–Klausel. Dazu illustrieren wir zunächst die Definitionskette anhand der ersten und der dritten XXL–Suchbedingung aus der Beispielanfrage aus Abb. 4.1. Dieser Überblick soll den Zusammenhang der einzelnen Schritt verdeutlichen. Im Anschluss daran werden wir die einzelnen Definitionen angeben und detailliert erläutern.

1. Die XXL–Syntax (Abschnitt 4.2) gibt die korrekte Art und Weise an, elementaren Bedingungen an die Namen und Werte von Elementen und Attributen des XML–Graphen zu konjunktiv verknüpften XXL–Suchbedingungen einer Where–Klausel einer XXL–Anfrage in Textform zusammenzusetzen (siehe Abb. 4.1).

- Elementare Bedingungen an die Namen von Elementen und Attributen, die in den

n–Knoten des XML–Graphen stehen, werden in Form von Knotenbedingungen (Labelausdrücke inkl. Wildcards oder Variablen) angegeben.

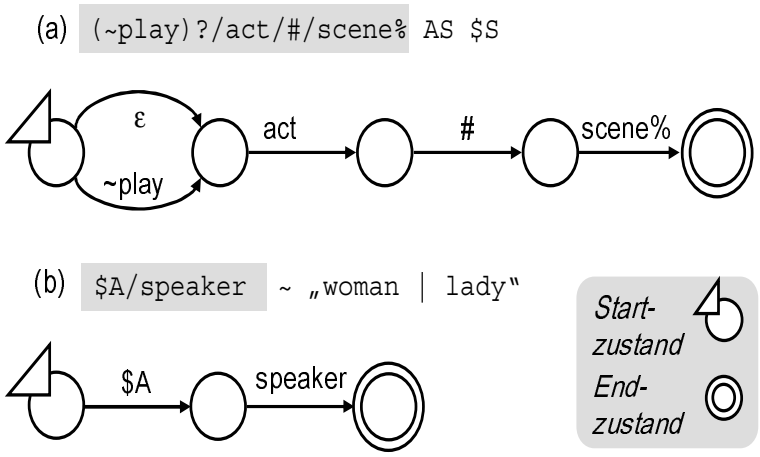
- Bedingungen an die Struktur, also die Anordnung von n–Knoten, werden in Form von Pfadbedingungen angegeben. Pfadbedingungen beschreiben gesuchte n–Pfade des XML–Graphen und werden als reguläre Ausdrücke über Knotenbedingungen formuliert.
- Eine Pfadbedingung kann an eine Inhaltsbedingung geknüpft sein. Eine Inhaltsbedingung besteht aus einem binären Operationszeichen und einer Variablen bzw. einem Stringausdruck. Sie bezieht sich auf den letzten durch eine Pfadbedingung ermittelten n–Knoten des XML–Graphen und vergleicht dessen Element–/Attributwert in dem/den zugehörigen i–Knoten gemäß des gegebenen binären Operationszeichens mit der Variablen oder dem Stringausdruck. Ein Stringausdruck ist dabei eine aussagenlogische Formel bestehend aus Stringkonstanten, die sich durch einen Operatorbaum darstellen lässt.
- Eine Pfadbedingung kann an eine Variable gebunden werden. Eine Variablenbindung knüpft einen Variablennamen mit Hilfe des reservierten Wortes *'AS'* an den/die letzten Knoten einer Pfadbedingung.
- Eine XXL–Suchbedingung besteht aus einer Pfadbedingung, einer Pfadbedingung und einer Inhaltsbedingung oder einer Pfadbedingung und einer Variablenbindung.
- Die Where–Klausel einer XXL–Anfrage besteht aus konjunktiv verknüpften XXL–Suchbedingungen.

Die Besonderheit dieser XML–Anfragesprache liegt in der Bereitstellung eines Operators, mit dessen Hilfe sich semantische, ontologiebasierte Ähnlichkeitsbedingungen unär an Element– und Attributnamen einerseits und binär an Element– und Attributwerte andererseits formulieren lassen.

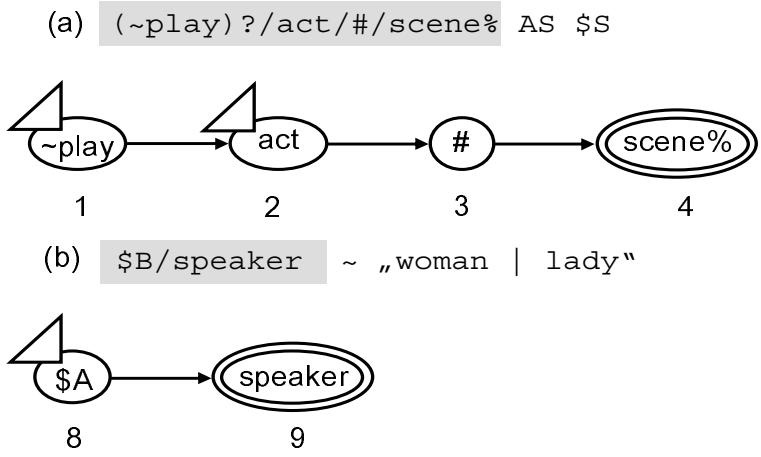
2. Die XXL–Semantik legt fest, welche Knoten, Pfade bzw. Teilgraphen im XML–Graphen den gegebenen Pfadbedingungen, Variablenbindungen und Inhaltsbedingungen genügen und somit als Treffer gewertet werden.

2.1 Bedingungen an die Strukturdaten im XML–Graphen werden mit Hilfe von Pfadbedingungen formuliert. Da jede Pfadbedingung ein regulärer Ausdruck über Knotenbedingungen ist, lässt sie sich durch einen nichtdeterministischen endlichen Automaten (kurz: NEA) darstellen (Abschnitt 4.3).

- Eine Pfadbedingung lässt sich durch ein Übergangsdiagramm (graphbasierte Darstellung des NEA), einen gerichteten, kantenmarkierten Graphen repräsentieren, der die durch die Pfadbedingung beschriebene reguläre Sprache akzeptiert (siehe folgende Abbildung).



- Ein Übergangsdiagramm kann in einen gerichteten, knotenmarkierten Pfadbedingungsgraphen transformiert werden, der dieselbe reguläre Sprache akzeptiert wie das Übergangsdiagramm (siehe folgende Abbildung).



2.2 Die Semantik einer XXL–Anfrage wird durch die Art der Auswertung von Pfadbedingungsgraphen, Inhaltsbedingungen und Variablenbindungen und durch die daraus resultierende Relevanzbewertung von n–Knoten, n–Pfaden und i–Knoten im XML–Graphen definiert (Abschnitt 4.4). Dabei spielen folgende Aspekte eine wesentliche Rolle.

- Die Auswertung von atomaren Pfadbedingungen basiert auf dem Vergleich elementarer Bedingungen mit den Namen von Elementen und Attributen.
- Die Auswertung von zusammengesetzten Pfadbedingungen beruht auf dem Finden von n–Pfaden im XML–Graphen (sogenannten *relevanten Pfade*), die vom entsprechenden Pfadbedingungsgraphen akzeptiert werden.
- Gebundene Variablen werden mit dem letzten n–Knoten eines vom zugehörigen Pfadbedingungsgraphen akzeptierten n–Pfades belegt. Die Belegung einer Variablen muss für alle XXL–Suchbedingungen gelten, in denen sie definiert bzw. angewandt wird.
- Die Auswertung einer zu einer Pfadbedingung gehörenden Inhaltsbedingung basiert auf dem Vergleich elementarer Bedingungen mit den Werten von Elementen und Attributen. Dazu werden die zum letzten n–Knoten eines akzeptierten n–Pfades gehörenden i–Knoten mit der gegebenen Bedingung verglichen.
- Ein Ergebnis für eine XXL–Anfrage ist ein *Resultatgraph*, der sich aus je einem relevanten Pfad pro XXL–Suchbedingung zusammensetzt, wobei die Belegung der vorkommenden Variablen für alle beteiligten relevanten Pfade gilt.

Beispiel 4.2 Die folgende Abbildung fasst die wesentlichen Aspekte der Syntax– und Semantikdefinitionen für XXL anhand eines Beispiels zusammen. Dazu haben wir relevante Knoten des XML–Graphen grau schattiert und mit den knotenbasierten, lokalen Relevanzwerten bzw. belegten Variablen annotiert. Die hier schattierten Knoten ergeben einen Resultatgraphen zur gegebenen Anfrage, dessen Relevanzwert sich aus der Multiplikation der lokalen Relevanzwerte aller beteiligten Knoten ergibt. Die gepunkteten Knoten und Kanten sind für dieses eine Ergebnis nicht relevant.

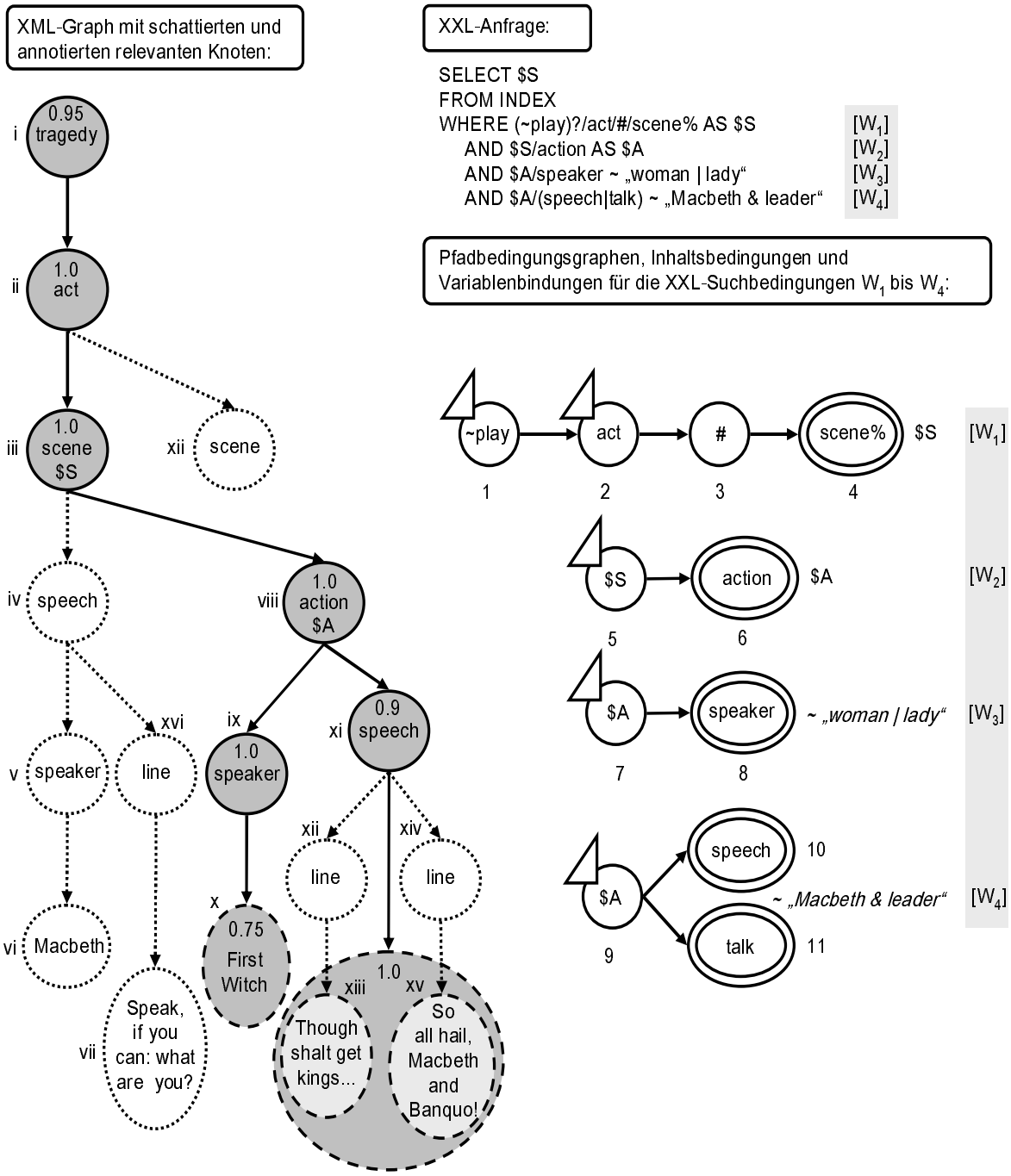


Abbildung 4.2: Syntax und Semantik von XXL–Anfragen

Aus den XXL–Suchbedingungen in Textform (rechts oben) werden die zugehörigen Pfadbedingungsgraphen, Variablenbindungen und Inhaltsbedingungen (rechts unten) erzeugt. Für die Variablenbelegung $\$S \rightarrow iii$ und $\$A \rightarrow viii$ gibt es zu jeder XXL–Suchbedingung einen relevanten Pfad im XML–Graphen (links), der dieser Variablen-

belegung genügt. Für die erste XXL–Suchbedingung genügt der n–Pfad *i/ii/iii* dem Pfad 1/2/3/4 im Pfadbedingungsgraphen mit den eingetragenen lokalen Relevanzwerten. Für die zweite XXL–Suchbedingung genügt der n–Pfad *iii/viii* dem Pfad 5/6 im Pfadbedingungsgraphen mit den eingetragenen lokalen Relevanzwerten. Für die dritte XXL–Suchbedingung genügt der n–Pfad *viii/ix* dem Pfad 8/9 mit den eingetragenen lokalen Relevanzwerten, wobei der zum n–Knoten *ix* gehörende Elementwert die gegebene Inhaltsbedingung mit dem angegebenen Relevanzwert erfüllt. Schließlich ergibt sich für die vierte XXL–Suchbedingung, dass der n–Pfad *viii/xi* dem Pfad 9/11 des Pfadbedingungsgraphen genügt, wobei die gegebenen Inhaltsbedingung von den zugehörigen i–Knoten *xiii* und *xv* erfüllt wird.

4.2 Syntax

Die Syntax gibt die korrekte Art und Weise an, wie elementare Bedingungen an die Namen und Werte von Elementen und Attributen des XML–Graphen zu komplexen konjunktiv verknüpften XXL–Suchbedingungen einer Where–Klausel einer XXL–Anfrage zusammengesetzt werden. In diesem Abschnitt wird die Syntax der textuell gegebenen XXL–Suchbedingungen einer gegebenen XXL–Anfrage induktiv definiert. Die Syntax einer XXL–Anfrage, insbesondere der in ihr enthaltenen XXL–Suchbedingungen basiert auf dem Unicode–Zeichensatz [UC00] und enthält dabei diverse Zeichen mit besonderer Bedeutung. Sei Σ das endliche Unicode–Alphabet [UC00]. Wir unterscheiden folgende Alphabete, die Sonderzeichen der Sprache XXL enthalten und Teilmengen von Σ sind:

- Σ_{META}={(,), /, $, ?, +, *, |, #, "} die Menge der Metazeichen,
- Σ_{WC} ={?, % } die Menge der Wildcardzeichen,
- Σ_{BOOL} = {&, |, !} die Menge der Booleschen Operationszeichen und
- Σ_{OP} = {$\sim$, *LIKE*, =, $\neq$} die Menge der Operationszeichen.

Symbole werden im Folgenden immer als Zeichen und nicht als Zeichenketten interpretiert. Die Darstellung eines Symbols durch eine Zeichenkette, z.B. 'LIKE' statt 'L', dient der Verbesserung der Lesbarkeit formaler Zusammenhänge. Jedes Symbol kann natürlich leicht durch ein einzelnes Zeichen dargestellt werden. Damit können wir einheitlich von Alphabeten sprechen.

4.2.1 Labelausdrücke, Variablen und Knotenbedingungen

Knotenbedingungen sind elementare Bedingungen an Element– und Attributnamen, die in den n–Knoten des XML–Graphen stehen. Knotenbedingungen werden aus Labelausdrücken und Variablennamen gebildet.

Ein Labelausdruck ist eine nicht leere Zeichenkette, die einem Element– oder Attributnamen entsprechen soll. In diesem Labelausdruck können Wildcardzeichen als Platzhalter für einzelne Zeichen (?) oder Zeichenketten (%) verwendet werden. Die Syntax von Labelausdrücken ist eine Erweiterung der Syntax von Element– und Attributnamen gegeben durch die Grammatik G_N der Definition 2.1 um die Wildcardzeichen ? und %.

Definition 4.1 (Labelausdruck)
Sei Σ das endliche Unicode–Alphabet und sei die rrkf Grammatik $G_N = (N_N, \Sigma_N, p_N, S_N)$ zur Definition von Element– und Attributnamen gegeben.

Dann ist $G_L = (N_L, T_L, p_L, S_L)$ eine rechtsreguläre kontextfreie Grammatik mit:

- *der endlichen Menge N_L=$\{Label\}$ der Nichtterminale,*
- *der endlichen Menge $T_L = \Sigma_L = \Sigma_N \cup \Sigma_{WC}$ der Terminale,*
- *dem Startsymbol S_L=Label aus N_L und*
- *der Funktion $p_L : N_L \rightarrow RA$, die die Menge N_L der Nichtterminale in die Menge RA der regulären Ausdrücke über $(N_L \cup T_L)$ abbildet:*
  $p_L(Label) = xy^*$
  wobei $x \in \Sigma_L \backslash \{0, \ldots, 9, ., -\}$ und $y \in \Sigma_L$.

Die von G_L definierte Sprache ist $L(G_L) = \{w \in \Sigma_L^* | Label \stackrel{*}{\Rightarrow} w\}$, die Menge der aus dem Startsymbol S_L=Label gemäß p_L ableitbaren Element– und Attributnamen mit optionalen Wildcardzeichen ? und %. □

Im Rahmen von XXL wird ein Variablenname immer durch ein vorangestelltes $–Symbol gekennzeichnet. Zur Definition von Variablennamen verwenden wir die Definition 2.1 von Namen von Elementen und Attributen aus Abschnitt 2.1. Dabei verwenden wir ***Name*** als Symbol für einen Namen aus $L(G_N)$ und setzen es als Terminal in der folgenden kontextfreien Grammatik ein.

Definition 4.2 (Variablennamen)
*Sei Σ das endliche Unicode–Alphabet. Sei **Name** ein Symbol für einen Element– oder Attributnamen aus $L(G_N)$.*

Dann ist $G_V = (N_V, T_V, P_V, S_V)$ eine kontextfreie Grammatik mit:

- *der endlichen Menge N_V=$\{Variable\}$ der Nichtterminale,*
- *der endlichen Menge $T_V = \Sigma_V = \Sigma_N \cup \{\$\}$ der Terminale,*
- *dem Startsymbol S_V=Variable aus N_V und*
- *der Menge der Produktionsregeln $P_V \subseteq N_V \times (N_V \cup T_V)^*$:*

$$P_V = \{ \; Variable \rightarrow \textbf{\$Name} \; \}$$

Die von G_V definierte Sprache ist $L(G_V) = \{w \in \Sigma_V^* | Variable \stackrel{*}{\Rightarrow} w\}$, die Menge der aus dem Startsymbol S_V=Variable gemäß P_V ableitbaren Variablennamen. □

Eine Knotenbedingung besteht entweder aus einem Labelausdruck, einer Variablen oder einem Labelausdruck, dem der unäre Ähnlichkeitsoperator '$\sim \in \Sigma_{OP}$' vorangestellt ist. Dabei bedeutet der Ähnlichkeitsoperator, dass (mit Hilfe der Ontologie) nach semantisch ähnlichen Namen gesucht wird. Eine Variable dient hierbei als Platzhalter für einen n–Knoten des XML–Graphen. Eine Variable, die als Knotenbedingung auftritt wird als *angewandtes Variablenvorkommen* bezeichnet.

Definition 4.3 (Knotenbedingung)
Sei $L(G_L)$ die Menge der Labelausdrücke, $L(G_V)$ die Menge der Variablen und $\sim \in \Sigma_{OP}$ der unäre Ähnlichkeitsoperator. Dann ist $\mathcal{K} = L(G_L) \cup (\{\sim\} \times L(G_L)) \cup L(G_V)$ die Menge der Knotenbedingungen. □

Wie man unschwer sieht, ist jede Knotenbedingung ein regulärer Ausdruck über dem Alphabet Σ.

4.2.2 Pfadbedingungen

Pfadbedingungen sind Bedingungen an die Anordnung von n–Knoten (Struktur) im XML–Graphen. Sie werden als reguläre Ausdrücke über Knotenbedingungen formuliert.

Die Syntax von Pfadbedingungen wird mit Hilfe einer kontextfreien Grammatik definiert. Dazu führen wir für die Knotenbedingungen entsprechende Symbole ***Label***, ***Variable*** und ∼***Label*** gemäß der Definition 4.3 ein. Diese Symbole werden als Terminalsymbole in der folgenden Definition verwendet.

Definition 4.4 (Pfadbedingungen)
Sei Σ das endliche Unicode–Alphabet. Seien ***Label*** *und* ∼***Label*** *Symbole für Labelausdrücke aus $L(G_L)$ bzw. $\{\sim\} \times L(G_L)$ und* ***Variable*** *ein Symbol für eine Variable aus $L(G_V)$.*

Dann ist $G_P = (N_P, T_P, P_P, S_P)$ eine kontextfreie Grammatik mit:

- *der endlichen Menge N_P=*{*Path, Pathexpr, Labeldisj, Labellist, Labelexpr*} *der Nichtterminale,*
- *der endlichen Menge $T_P = \Sigma_P =$* {***Label, Variable,*** ∼***Label***} $\cup\ \Sigma_{META}$ *der Terminale,*
- *dem Startsymbol S_P=Path aus N_P und*
- *der Menge der Produktionsregeln $P_P \subseteq N_P \times (N_P \cup T_P)^*$:*

$$
\begin{aligned}
P_P = \{ & \\
& (1)\ \mathit{Path} \rightarrow \mathit{Pathexpr} \mid \mathit{Pathexpr}/\mathit{Labeldisj} \mid \mathit{Labeldisj} \\
& (2)\ \mathit{Pathexpr} \rightarrow \mathit{Labelexpr} \mid \textbf{\textit{Variable}} \mid \# \mid \\
& (3)\ \quad \mathit{Pathexpr}/\mathit{Pathexpr} \mid \\
& (4)\ \quad (\mathit{Pathexpr})\textbf{?} \mid (\mathit{Pathexpr})+ \mid (\mathit{Pathexpr})\textbf{*} \\
& (5)\ \mathit{Labeldisj} \rightarrow \mathit{(Labelexpr}\ '|'\ \mathit{Labelexpr\ Labellist)} \\
& (6)\ \mathit{Labellist} \rightarrow \varepsilon \mid '|'\ \mathit{Labelexpr\ Labellist} \\
& (7)\ \mathit{Labelexpr} \rightarrow \textbf{\textit{Label}} \mid \sim\textbf{\textit{Label}} \\
\}&
\end{aligned}
$$

Die von G_P definierte Sprache ist $L(G_P) = \{w \in \Sigma_P^* | Path \stackrel{*}{\Rightarrow} w\}$, die Menge der aus dem Startsymbol S_P=Path gemäß P_P ableitbaren Pfadbedingungen. □

Eine *atomare Pfadbedingung* entspricht einer Knotenbedingung. Eine *zusammengesetzte Pfadbedingung* besteht aus einer Menge von atomaren Pfadbedingungen, die mit Hilfe der Metazeichen /, |, ?, +, * und den notwendigen Klammern miteinander verknüpft sind. Die hier verwendeten Metazeichen haben folgende Bedeutungen:

- / dient der Konkatenation von Pfadbedingungen,
- | dient der disjunktiven Verknüpfung von atomaren Pfadbedingungen,
- ? verweist auf das optionale Vorkommen einer Pfadbedingung,
- + verweist auf das einfache oder mehrfache Vorkommen einer Pfadbedingung,
- * verweist auf das optionale oder mehrfache Vorkommen einer Pfadbedingung.

Beispiel 4.3 Die ersten vier Pfadbedingungen sind erlaubt.

```
1. ~play/(act)?/scene
2. ~play/#/scene
3. ~play/$A/scene/$B/title
4. ~play/(act)?/scene/(~title|stage|speech)
```

Im Gegensatz dazu sind die folgenden zwei Pfadbedingungen nicht erlaubt, weil zum einen die Disjunktion nicht am Ende der Pfadbedingung steht und zum anderen die Disjunktion nicht nur über Labelausdrücke sondern über zusammengesetzte Pfadbedingungen geht.

```
1. ~play/(act)?/scene/(~title | stage | speech)/speaker
2. ~play/(act)?/scene/(~title/subtitle | stage | speech/speaker)
```

Der folgende Satz bringt eine strukturelle Besonderheit von Pfadbedingungen zum Ausdruck.

Satz 4.1
Sei $L(G_P)$ die Menge der Pfadbedingungen und sei $\mathcal{K} = L(G_L) \cup (\{\sim\} \times L(G_L)) \cup L(G_V)$ die Menge der Knotenbedingungen. Eine Pfadbedingung $p \in L(G_P)$ ist ein regulärer Ausdruck über Σ, insbesondere aber ein regulärer Ausdruck über Knotenbedingungen aus K. □

Beweis:
Wir zeigen, dass die Konstruktion von Pfadbedingungen gemäß der Produktionen der Grammatik G_P reguläre Ausdrücke (kurz: RA) über Knotenbedingungen hervorbringt.

1. Eine Knotenbedingung $k \in \mathcal{K} = L(G_L) \cup (\{\sim\} \times L(G_L)) \cup L(G_V)$ ist ein RA über Σ. Folglich sind alle Terminalsymbole gemäß der Produktionen (2) und (7) der Grammatik G_P mit Ausnahme des Zeichens '#' RA über Σ.
2. Seien $k, k_1, k_2 \in \mathcal{K}$ Knotenbedingungen, also RA über Σ entsprechend der Definition von regulären Ausdrücken aus [WM97].
   (a) Die Konstruktion k_1/k_2 gemäß der Produktion (3) entspricht der Konkatenation von einem RA und einem zweiten RA und ist somit ein RA über Σ.
   (b) Die Konstruktion $(k)*$ gemäß der Produktion (4) stellt die transitiven Hülle eines RA dar und ist somit ein RA über Σ.
   (c) Das Zeichen $\# \in \Sigma$ gemäß der Produktion (2) dient als abkürzende Schreibweise für $(\%)*$, wobei $\% \in L(G_L)$ ist. Das Zeichen $\# \in \Sigma$ wird also als Wildcardzeichen für einen beliebigen Pfad verwendet und ist aufgrund seiner Definition ein RA über Σ.
   (d) Die Konstruktion $(k)?$ gemäß der Produktion (4) ist ein RA über Σ, weil $(k)? = (k/\varepsilon)$ ist und (k/ε) wegen 2a ein RA ist.
   (e) Die Konstruktion $(k)+$ gemäß Produktion (4) ist ein RA über Σ, weil $(k)+ = (k/(k)*)$ ist und $(k/(k)*)$ wegen 2a und 2b ein RA ist.
   (f) Die Konstruktion $k_1|k_2$ für $k_1, k_2 \in L(G_L) \cup (\{\sim\} \times L(G_L))$ gemäß Produktion (5) entspricht der Disjunktion von Labelausdrücken und ist somit ein RA über Σ.

(g) Die Konstruktion k_1/k_2 für einen RA nach 2a – 2e und einer regulären Labeldisjunktion nach 2f entspricht der Konkatenation eines RA und einer regulären Labeldisjunktion und ist somit ein RA über Σ.

Es gibt keine weiteren Produktionen in G_P, die berücksichtigt werden müssen. Somit ist jede Pfadbedingung $p \in L(G_P)$ ein RA über Σ und insbesondere ein RA über Knotenbedingungen aus $\mathcal{K}$. □

An dieser Stelle möchten wir auf eine Besonderheit hinweisen. Das Wildcardzeichen '#' ist eine Abkürzung für die Pfadbedingung '(%)*' und dient als Platzhalter für einen beliebigen n–Pfad im XML–Graphen. Für den Fall, dass dieser Pfad hinsichtlich seiner Länge und seines Verlaufs für das Gesamtergebnis keine weitere Bedeutung hat, stellt es keine Einschränkung dar, wenn das Wildcardzeichen als atomare Pfadbedingung angesehen wird. In diesem Fall ist eine Pfadbedingung ein regulärer Ausdruck über $\mathcal{K} \cup \{\#\}$.

4.2.3 Stringausdrücke und Inhaltsbedingungen

Eine Pfadbedingung kann an eine Inhaltsbedingung geknüpft sein. Dabei besteht eine *Inhaltsbedingung* entweder aus einem binären Operationszeichen aus Σ_{OP} und einer aussagenlogischen Formel aus Stringkonstanten oder aus einem binären Operationszeichen und einer Variablen. Sie bezieht sich auf den letzten durch eine Pfadbedingung im XML–Graphen gefundenen n–Knoten und vergleicht dessen Element–/Attributwert in dem/den zugehörigen i–Knoten mit der vorgegebenen elementaren Bedingung.

Ein Stringausdruck ist eine aussagenlogische Formel [Sch95], wobei die atomaren Formeln Stringkonstanten über dem Alphabet $\Sigma\backslash\{\&, |, !\}$ sind. Die Syntax der erlaubten Stringkonstanten orientiert sich an der Syntax der Element– und Attributwerte gemäß der Definition 2.2. Zusätzlich werden ? und % als Wildcardzeichen interpretiert. Die Stringkonstanten können zu komplexen Formeln der Aussagenlogik [Sch95] mit Hilfe der Operationszeichen '&' (Konjunktion), '|' (Disjunktion) und '!' (Negation) zusammengesetzt werden.

Zur Definition der Syntax von erlaubten Stringausdrücken stehen uns verschiedene Herangehensweisen zur Verfügung. Zum einen könnten wir verlangen, dass die Stringausdrücke vollständig geklammert sind und somit die Reihenfolge der Auswertung der Operationszeichen feststeht. Aus der Sicht der Anwendung ist es wesentlich flexibler und einfacher, unvollständig geklammerte Stringausdrücke zuzulassen, wobei im Vorfeld für die Operationszeichen Prioritäten festgelegt werden.

Sei ***Wert*** ein Symbol für ein Element– oder Attributwert aus $L(G_I)$, das wir in der nachfolgenden Definition als Terminalsymbol verwenden. Hierbei legen wir fest, dass ein Stringausdruck zur Begrenzung in doppelte Anführungszeichen gesetzt wird. Die Operationszeichen werden mit Prioritäten versehen, dabei hat die Negation die höchste und die Disjunktion die niedrigste Priorität. Die folgende Grammatik spezifiziert die Syntax von Stringausdrücken, die nicht zwingend vollständig geklammert sind.

Definition 4.5 (Stringausdruck)
*Sei Σ das endliche Unicode–Alphabet. Sei **Wert** ein Symbol für einen Wert aus $L(G_W)$.*

Dann ist $G_S = (N_S, T_S, P_S, S_S)$ eine kontextfreie Grammatik mit:

- *der endlichen Menge N_S={String, Stringexpr, Expr1, Expr2} von Nichtterminalen,*

- *der endlichen Menge* $T_S = \Sigma_S = \{$ ***Wert***, **(,)**$\} \cup \Sigma_{BOOL}$ *der Terminale,*
- *dem Startsymbol* S_S*=String aus* N_S *und*
- *der Menge der Produktionsregeln* $P_S \subseteq N_S \times (N_S \cup T_S)^*$*:*

$$
\begin{aligned}
P_S = \{ & \\
& \textit{String} \rightarrow \textit{"Stringexpr"} \\
& \textit{Stringexpr} \rightarrow \textit{Expr1} \mid \textit{Stringexpr}\ '|'\ \textit{Expr1} \\
& \textit{Expr1} \rightarrow \textit{Expr2} \mid \textit{Expr1}\ \&\ \textit{Expr2} \\
& \textit{Expr2} \rightarrow \textbf{\textit{Wert}} \mid \textbf{!}\textit{Expr2} \mid \textbf{(}\textit{Stringexpr}\textbf{)} \\
\} &
\end{aligned}
$$

Die von G_S *definierte Sprache ist* $L(G_S) = \{w \in \Sigma_S^* | \textit{String} \stackrel{*}{\Rightarrow} w\}$*, die Menge der aus dem Startsymbol* S_S*=String gemäß* P_S *ableitbaren Stringausdrücke.* □

Stringausdrücke haben zwei besondere Eigenschaften.

- Eine Stringkonstante darf die Metazeichen der Grammatik G_S enthalten. Wir benötigen also einen Mechanismus, um Metazeichen von Konstantenzeichen zu unterscheiden.
- Die Ableitung eines Stringausdrucks aus dem Startsymbol gemäß obiger Grammatik lässt sich durch einen Operatorbaum darstellen, dessen innere Knoten die Operationszeichen und die Blätter die Stringkonstanten enthalten.

Jeder Stringausdruck hat den gleichen syntaktischen Aufbau wie der Wert eines Elements oder eines Attributes (siehe Definition 2.2 der Grammatik G_W im Abschnitt 2.1). Da die Zeichen ", &, |, !, ? und % Bestandteil dieser Syntax sind, müssen wir einen Mechanismus zur Unterscheidung von diesen Zeichen als Zeichen des Inhalts bzw. als Metazeichen der Sprache XXL einführen. Wir nehmen die Entity–Attribute aus der XML–Spezifikation als Vorlage (siehe Abschnitt 2.2.2) und legen fest, dass für jedes Metazeichen der Sprache XXL, das ein erlaubtes Zeichen für den Inhalt von Elementen und Attributen ist, eine Ersetzungsregel (Entity) vereinbart wird: &quo für ", &amp für &, &or für |, &exc für !, &que für ? und &per für %. Soll also ein Metazeichen als Literal im Stringausdruck vorkommen, wird es dort durch die entsprechende Entity gekennzeichnet und zum Zeitpunkt der Auswertung ersetzt.

Beispiel 4.4 Die folgenden zwei Stringausdrücke sind zulässig:

```
1. !web & (java | xml)
2. (X&amp;Y Systems) & (X&amp;Y Inc.)
```

Zum Zeitpunkt der Auswertung wird dann aus dem zweiten Stringausdruck, der aus zwei konjunktiv verknüpften atomaren Stringkonstanten besteht, durch Auflösung der Entity der eigentliche Stringausdruck "`(X&Y Systems) & (X&Y Inc.)`".

Die Ableitung eines Stringausdruck gemäß obiger Grammatik lässt sich durch einen Syntaxbaum darstellen [WM97]. Betrachten wir dazu folgendes Beispiel.

Beispiel 4.5 Für vier Stringkonstanten $c_1, c_2, c_3, c_4 \in L(G_W)$ ist der Ausdruck $''!(c_1|c_2)\&(c_3|c_4)''$ ein Stringausdruck gemäß G_S, d.h. es gibt eine Ableitung $\textit{String} \stackrel{*}{\Rightarrow} ''!(c_1|c_2)\&(c_3|c_4)''$.

Die syntaktische Struktur einer möglichen Ableitung eines gegebenen Stringausdrucks aus dem Startsymbol entsprechend der Produktionen der Grammatik G_S kann mit Hilfe eines Syntaxbaums dargestellt werden. Ein Syntaxbaum für Stringausdrücke wird dazu wie folgt definiert.

Definition 4.6 (Syntaxbaum für Stringausdrücke)
Sei $G_S = (N_S, T_S, P_S, S_S)$ die kontextfreie Grammatik für Stringausdrücke. Sei S=(V,E) ein geordneter Baum, also ein Baum, in dem die Ausgangskanten jedes Knotens geordnet sind. Dabei sind $v \in T_S \cup \{\varepsilon\}$ die Blätter und $v \in N_S$ die inneren Knoten des Baums. S heißt Syntaxbaum für die Inhaltsbedingung $a \in T_S^*$ und $A \in N_S$ gemäß G_S, wenn gilt:

- *Ist $A' \in V$ ein innerer Knoten, so sind entweder seine Kinder von links nach rechts $A_1, A_2, \ldots, A_k \in N_S \cup T_S$ ($k > 1$) und $A' \rightarrow A_1 A_2 \ldots A_k$ ist eine Produktionsregel in P_S, oder sein einziges Kind $A_1 = \varepsilon$ und $A' \rightarrow \varepsilon$ ist eine Produktionsregel in P_S.*
- *a ist das Terminalwort (Blattwort) von S.*
- *A ist die Wurzel von S.*

□

Ein Syntaxbaum für einen Stringausdruck a und das Startsymbol $String \in N_S$ heisst einfach Syntaxbaum für a.

Beispiel 4.6 In der Abb. 4.3 geben wir den Syntaxbaum passend zum Stringausdruck $expr$=″!$(c_1|c_2)$&$(c_3|c_4)$″ an.

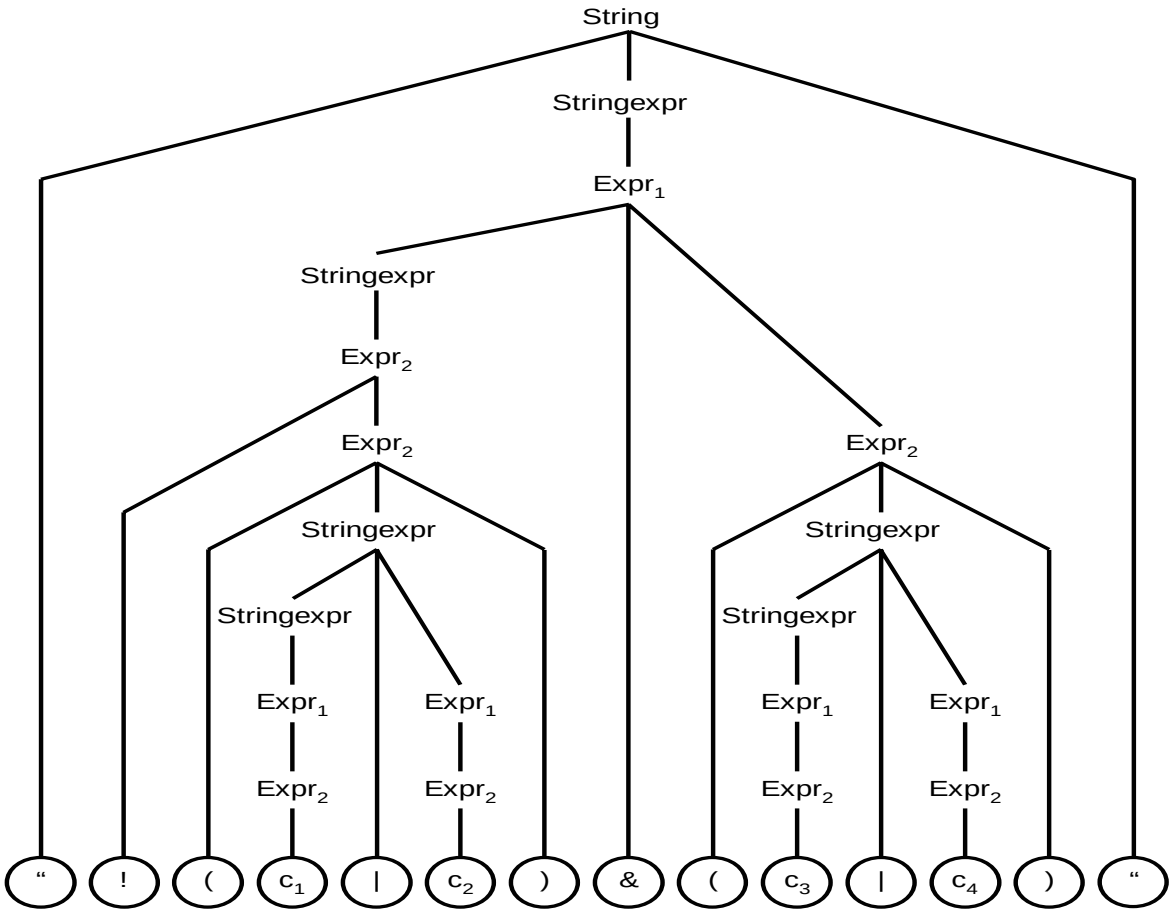


Abbildung 4.3: Syntaxbaum für den Stringausdruck "$!(c_1|c_2)\&(c_3|c_4)$"

Zur Auswertung einer Inhaltsbedingung sind die verwendeten Stringkonstanten und die Reihenfolge der Operationen wichtig. Diese Informationen sind im Syntaxbaum verankert. Allerdings enthält dieser auch Klammern und Nichtterminale, die zum Zeitpunkt der Auswertung des Syntaxbaums keine Bedeutung mehr haben. Aus diesem Grund transformieren wir den Syntaxbaum in einen äquivalenten Operatorbaum, dessen Knoten ausschließlich aus Terminalen der Grammatik G_S bestehen.

Definition 4.7 (Operatorbaum für Stringausdrücke)
Sei G_S die kontextfreie Grammatik für Stringausdrücke. Sei a eine Stringausdruck gemäß G_S und sei $S = (V_S, E_S)$ ein Syntaxbaum für a.

Ein Operatorbaum $T = (V, E)$ für a ist ein reduzierter Syntaxbaum.

Die Knotenmenge $V \subseteq V_S$ enthält nur noch Terminale, aber keine Klammern. Insbesondere sind innere Knoten $u \in \Sigma_{BOOL}$ und Blätter $v \in L(G_I)$.

Seien $a, b, d \in T$. Es gibt immer dann eine Kante $(a, b) \in E$, wenn der Vater A von a ein Vorfahre vom Vater B von b in S ist und kein anderer Vater D eines Knotens d auf dem Pfad $\langle A \ldots B \rangle$ in S liegt. □

Die Konstruktion eines Operatorbaums durch Reduktion des vorgegebenen Syntaxbaums wird in drei Schritten durchgeführt.

1. Es werden die Blätter und deren eingehende Kanten vom Baum gelöscht, die die Anführungszeichen enthalten.

2. Es werden die Blätter und deren eingehende Kanten vom Baum gelöscht, die ein Klammersymbol enthalten.
3. Es werden die Nichtterminale entfernt, die höchstens eine eingehende und eine höchstens eine ausgehende Kante haben und deren Vater und Kind jeweils ebenfalls ein Nichtterminal ist, in dem eine Kante vom Vater zum Kind gesetzt wird und der betrachtete Knoten mit seinen ein– und ausgehenden Kanten aus dem Baum entfernt werden.
4. Jedes Blatt (Terminal) ersetzt seinen Vater (Nichtterminal),dabei wird die Kante zwischen Vater und Blatt entfernt, die eingehenden Kanten des Vaters werden zu eingehenden Kanten des Blattes und der Vater wird entfernt.

Beispiel 4.7 In der Abb. 4.4 werden die drei Schritte der Konstruktion eines Operatorbaums durch die Reduktion des Syntaxbaums aus Abb. 4.3 für den Stringausdruck '$!(c_1|c_2)\&(c_3|c_4)$' demonstriert.

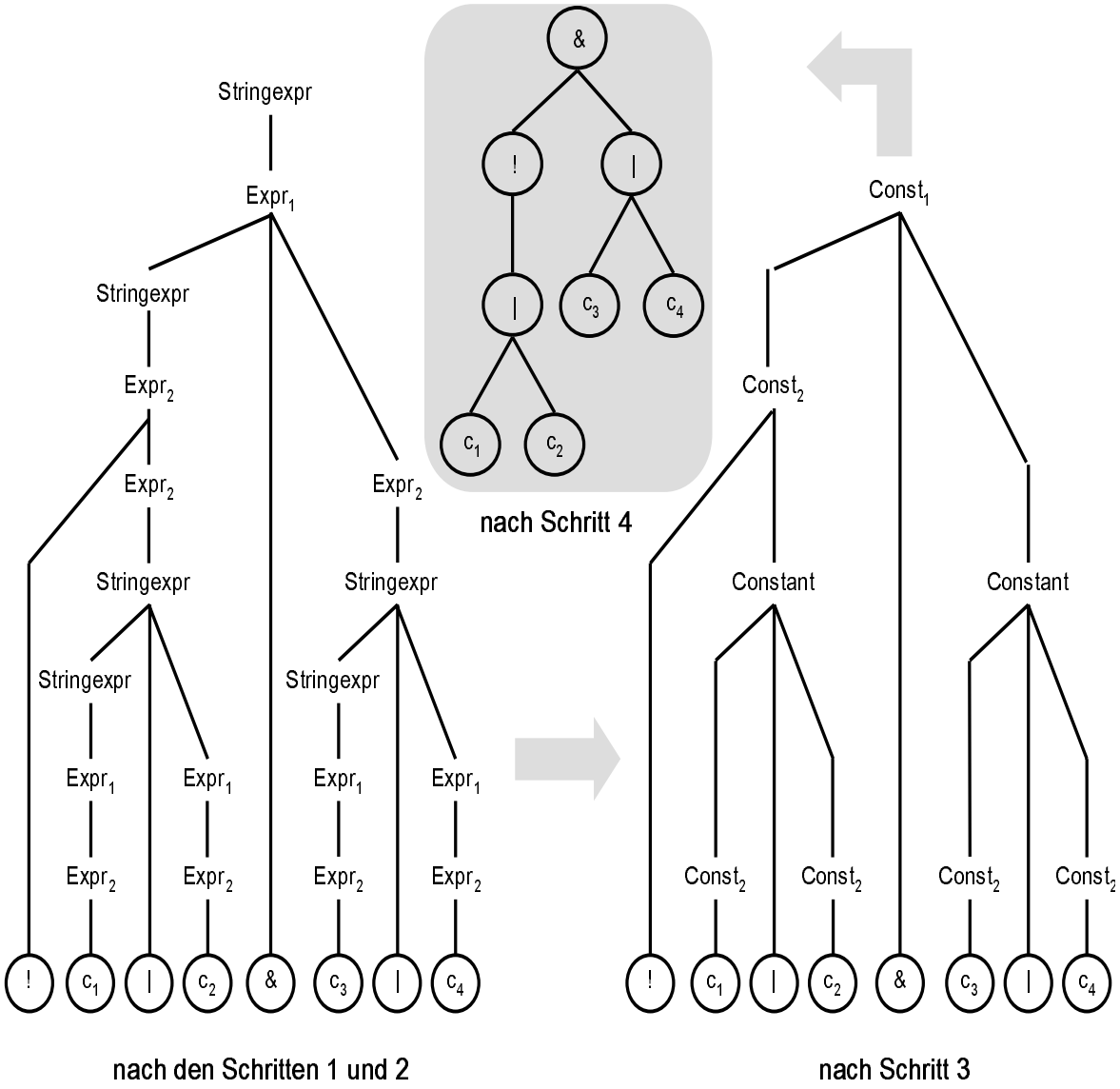


Abbildung 4.4: Konstruktion des Operatorbaums für den Stringausdruck '$!(c_1|c_2)\&(c_3|c_4)$'

Die Reduktion des Syntaxbaums zu einem Operatorbaum verändert die Semantik des Stringausdrucks, d.h. die Reihenfolge der Operationen gegeben durch die Tiefe im Baum, nicht. Die Korrektheit der Konstruktion ergibt sich durch folgende Begründung. Wenn der Stringausdruck als Syntaxbaum vorliegt, dann haben die Anführungszeichen und die Klammern keinen Einfluss mehr auf die Schachtelung der Operationen und können daher gefahrlos ohne Veränderung der Semantik entfernt werden (siehe Schritt 1 und 2). Wenn ein Nichtterminal höchstens einen Vater (Nichtterminal) und ebenso höchstens ein Kind (ebenfalls Nichtterminal) hat, dann hat dieses Nichtterminal keinen Einfluss auf die Struktur des Ausdrucks und kann ohne Veränderung des Blattwortes entfernt werden (siehe Schritt 2). Nach den ersten Schritten bleibt ein Baum übrig, dessen Nichtterminale genau ein Terminal als Kind haben. Diese können daher ebenfalls ohne Veränderung der Schachtelung und damit der Operationsreihenfolge zu einem Terminal zusammengelegt werden.

Besteht ein Stringausdruck aus einer einzelnen Konstanten, dann besteht der zugehörige Operatorbaum aus einem einzelnen Knoten, der diese Konstante enthält.

Im Rahmen dieser Arbeit liegen die Element– und Attributwerte in i–Knoten im XML–Graphen als Zeichenketten vor, über die wir keine Aussage über den Datentyp (Zahl, Datum, Währung, u.ä.) treffen können (siehe Kapitel 2). Aus diesem Grund beschränken wir die hier nutzbaren binären Operationszeichen auf die Menge $\Sigma_{OP} = \{=, \neq\ \mathit{LIKE}, \sim\}$. Damit sind wir in der Lage, die Gleichheit, die Ungleichheit, die lexikalische Ähnlichkeit und die semantische, ontologiebasierte Ähnlichkeit zweier Zeichenketten zu ermitteln.

Eine Inhaltsbedingung wird aus einem binären Operator und entweder aus einem Stringausdruck oder einer Variablen gebildet. Hierbei wird die Verwendung einer Variablen als *Joinbedingung* bezeichnet, da sie als Platzhalter für einen n–Knoten des XML–Graphen steht und somit als Referenz auf dessen Element– oder Attributwerte dient.

Definition 4.8 (Inhaltsbedingung)
Sei $L(G_S)$ die Menge der Stringausdrücke, $L(G_V)$ die Menge der Variablen und Σ_{OP} die Menge der binären Operationszeichen. Dann ist $\mathcal{I} = \Sigma_{OP} \times (L(G_S) \cup L(G_V))$ die Menge der Inhaltsbedingungen. □

4.2.4 Variablenbindung

Eine Pfadbedingung kann an eine Variable gebunden werden. Eine *Variablenbindung* knüpft einen Variablennamen mit Hilfe des reservierten Wortes *'AS'* an den/die letzten Knoten einer Pfadbedingung. Diese Variablenauftritte werden als *definierende Variablenvorkommen* bezeichnet.

Definition 4.9 (Variablenbindung)
Sei $L(G_V)$ die Menge der Variablen und 'AS' ein reserviertes Wort der Sprache XXL. Dann ist $\mathcal{F} = \{AS\} \times L(G_V)$ die Menge der Variablenbindungen.

Der Zweck einer Variablenbindung besteht darin, zum Zeitpunkt der Anfrageauswertung diese Variablenvorkommen mit Knoten des XML–Graphen zu belegen. Gibt es einen n–Pfad im XML–Graphen, der dieser Pfadbedingung genügt, dann wird die Variable mit seinem letzten n–Knoten belegt.

4.2.5 XXL–Suchbedingungen und XXL–Grammatik

Eine XXL–Suchbedingung besteht entweder aus einer Pfadbedingung oder einer Pfadbedingung und einer Inhaltsbedingung oder einer Pfadbedingung und einer Variablenbindung. Zur Definition der Syntax einer XXL–Suchbedingung führen wir die Symbole ***Path*** für eine Pfadbedingung, ***Content*** für eine Inhaltsbedingung bzw. ***VarBind*** für eine Variablenbindung ein, um sie in der folgenden kf Grammatik als Terminale zu verwenden.

Definition 4.10 (XXL–Suchbedingung)
*Sei Σ das endliche Unicode–Alphabet. Seien die Symbole **Path** für eine Pfadbedingung aus $L(G_P)$, **Content** für eine Inhaltsbedingung aus $\mathcal{I}$ sowie **VarBind** für eine Variablenbindung aus $\mathcal{F}$ gegeben.*

Dann ist $G_C = (N_C, T_C, P_C, S_C)$ eine kontextfreie Grammatik mit:

- *der endlichen Menge N_C=$\{Condition\}$ der Nichtterminale,*
- *der endlichen Menge $T_S = \Sigma_C = \{$**Path, Content, VarBind**$\}$ der Terminale,*
- *dem Startsymbol S_C=Condition aus N_C und*
- *der Menge der Produktionsregeln $P_C \subseteq N_C \times (N_C \cup T_C)^*$:*

$$
\begin{aligned}
P_C = \{ & \\
& (8)\ \mathit{Condition} \rightarrow \textbf{\textit{Path}} \mid \textbf{\textit{Path Content}} \mid \textbf{\textit{Path VarBind}} \\
\} &
\end{aligned}
$$

Die von G_C definierte Sprache ist $L(G_C) = \{w \in \Sigma_C^* | Condition \stackrel{*}{\Rightarrow} w\}$, die Menge der aus dem Startsymbol S_C=Condition gemäß P_C ableitbaren XXL–Suchbedingungen. □

Die Where–Klausel einer XXL–Anfrage besteht aus konjunktiv verknüpften XXL–Suchbedingungen. Zur Definition der Syntax einer XXL–Anfrage verwenden wir die Symbole ***Variable*** für eine Variable und ***Condition*** für eine XXL–Suchbedingung als Terminale.

Definition 4.11 (XXL–Grammatik)
*Sei Σ das endliche Unicode–Alphabet. Seien die Symbole **Variable** für eine Variable aus $L(G_V)$ und **Condition** für eine XXL–Suchbedingung aus $L(G_C)$ gegeben.*

Sei $G_{XXL} = (N_{XXL}, T_{XXL}, P_{XXL}, S_{XXL})$ eine kontextfreie Grammatik mit:

- *der endlichen Menge $N_{XXL} = \{Query, S, F, W, V\}$ der Nichtterminale,*
- *der endlichen Menge $T_{XXL} = \Sigma_{XXL} = \Sigma_{OP} \cup \{$**Variable, Condition, SELECT, FROM, WHERE, URLS, *, INDEX, AND, ','**$\}$ der Terminale,*

- *dem Startsymbol S_{XXL}=Query aus N_{XXL} und*
- *der Menge der Produktionsregeln $P_{XXL} \subseteq N_{XXL} \times (N_{XXL} \cup T_{XXL})^*$:*

$$
\begin{aligned}
P_{XXL} = \{ & \\
& (1)\ \mathit{Query} \rightarrow \textbf{\textit{SELECT}}\ S\ \textbf{\textit{FROM}}\ F \mid \\
& \qquad\qquad \textbf{\textit{SELECT}}\ S\ \textbf{\textit{FROM}}\ F\ \textbf{\textit{WHERE}}\ W \\
& (2)\ S \quad\ \rightarrow \textbf{*} \mid \textbf{\textit{URLS}} \mid V \\
& (3)\ V \quad\ \rightarrow \textbf{\textit{Variable}} \mid V,\ \textbf{\textit{Variable}} \\
& (4)\ F \quad\ \rightarrow \textbf{\textit{INDEX}} \\
& (5)\ W \quad\ \rightarrow \textbf{\textit{Condition}} \mid \textbf{\textit{Condition AND}}\ W \\
\} &
\end{aligned}
$$

Die von G_{XXL} definierte Sprache ist $L(G_{XXL}) = \{w \in \Sigma^*_{XXL} | Query \stackrel{*}{\Rightarrow} w\}$, die Menge der aus dem Startsymbol S_{XXL}=Query gemäß P_{XXL} ableitbaren XXL–Anfragen. □

An dieser Stelle ist es notwendig, auf Besonderheiten bei der Verwendung von Variablen in XXL–Anfragen einzugehen. Innerhalb einer XXL–Suchbedingung können Variablen in definierenden (Variablenbindung) oder angewandten Vorkommen (Knotenbedingung, Joinbedingung) auftreten.

Die richtige syntaktische Verwendung von Variablen innerhalb einer XXL–Suchbedingung und innerhalb einer Where–Klausel einer XXL–Anfrage ist ausschlaggebend für eine korrekte semantische Auswertung einer XXL–Anfrage. Diese Bedingungen lassen sich nicht in kontextfreie Grammatiken abbilden und sollen deshalb hier explizit erklärt werden.

1. Für jede in der WHERE–Klausel vorkommende Variable gibt es genau ein definierendes Vorkommen, also eine Variablenbindung mit 'AS'.
2. Innerhalb einer XXL–Suchbedingung darf jede Variable höchstens einmal vorkommen.

Die konjunktive Verknüpfung von XXL–Suchbedingungen wirkt sich nur auf die jeweils verwendeten Variablen aus. Wird eine XXL–Suchbedingung ausgewertet, dann werden die darin vorkommenden Variablen mit n–Knoten des XML–Graphen belegt. Wird dann eine weitere XXL–Suchbedingung ausgewertet, müssen die bestehenden Variablenbelegungen eingehalten werden. Um also eine Where–Klausel vollständig auswerten zu können, darf es zwischen den definierenden Variablenvorkommen (Variablenbindungen) und den angewandten Variablenvorkommen (Knotenbedingungen, Joinbedingungen) keine zyklischen Abhängigkeiten geben. Um ein definierendes Variablenvorkommen auswerten zu können, müssen die in der zugehörigen Pfadbedingung vorkommenden Variablen bereits definiert sein. Dazu definieren wir einen Variablenabhängigkeitsgraph auf der Basis einer "erscheint–vor"–Relation.

Definition 4.12 (Variablenabhängigkeitsgraph)
Seien $W_1, \ldots, W_j$ die j XXL–Suchbedingungen der insgesamt k XXL–Suchbedingungen einer XXL–Anfrage $(0 < j \leq k < \infty)$. Sei die kontextfreie Grammatik G_V=({Variable}, Σ_V, p, Variable) gegeben.

Die Relation '$<$' $\subseteq L(G_V) \times L(G_V)$ ist eine partielle Ordnung auf der Menge der in den Pfadbedingungen der Suchbedingungen W_i vorkommenden Variablen $(1 \leq i \leq j)$, wobei $a < b$ (lies: a "erscheint vor" b) gilt, falls es für zwei Variablen $a, b \in L(G_V)$ eine XXL–Suchbedingung W_i von der Form "...a...AS b" gibt.

Der Variablenabhängigkeitsgraph A=(V,E,<) ist ein gerichteter Graph mit einer Knotenmenge $V \subseteq L(G_V)$, die sich aus der Menge der in den XXL–Suchbedingungen $W_1, \ldots, W_j$ vorkommenden Variablen ergibt, und einer Kantenmenge E. Es gibt immer dann eine gerichtete Kante $e = (a, b) \in E$ zwischen zwei Variablen $a, b \in V$, wenn $a < b$. □

Wenn dieser Variablenabhängigkeitsgraph für eine gegebene XXL–Anfrage azyklisch ist, dann kann diese Anfrage problemlos ausgewertet werden.

Beispiel 4.8 Die folgenden drei XXL–Suchbedingungen sind erlaubt.

```
1. ~play/#/scene/speech/speaker ~ "woman"
2. ~play/#/scene/$A/speech ~ $B
3. ~play/#/scene AS $A AND $A/speech AS $B AND $B/speaker/$A/line LIKE "%King%"
```

Die folgenden zwei XXL–Suchbedingungen sind wegen der Art der Verwendung von Variablen nicht erlaubt. In der ersten XXL–Suchbedingung wird die Variable $\$A$ relativ zur Variablen $\$B$ zyklisch verwendet und in der zweiten XXL–Suchbedingung wird die Variable $\$A$ zweimal mit *'AS'* gebunden.

```
2. ~play/#/scene AS $A AND $A/speech AS $B AND $B/speaker AS $A
3. ~play/#/scene AS $A AND $A/speech AS $B AND $A/speaker AS $B
```

4.3 Transformation

Aus der Theorie der formalen Sprachen und Automaten [WM97] und den Definitionen aus Abschnitt 4.2 sind folgende Zusammenhänge bekannt.

1. Eine reguläre Pfadbedingung beschreibt eine reguläre Sprache, also eine Menge von n–Pfaden im XML–Graphen.
2. Ein nichtdeterministischer endlicher Automat (kurz: NEA) ist eine mathematische Maschine zum Erkennen/Akzeptieren regulärer Sprachen. Mit ihrer Hilfe kann man also prüfen, ob ein n–Pfad im XML–Graphen einer gegebenen Pfadbedingung genügt.
3. Es gibt einen Algorithmus, der zu jedem regulären Ausdruck einen äquivalenten NEA erzeugt. Ein NEA lässt sich durch ein äquivalentes endliches Übergangsdiagramm (einen gerichteten, kantenmarkierten Graphen) darstellen, das dieselbe reguläre Sprache akzeptiert.

Da jede Pfadbedingung ein regulärer Ausdruck über Knotenbedingungen ist, bietet sich eine graphbasierte Darstellung an, die n–Pfade des XML–Graphen akzeptiert, falls sie den gegebenen Knoten– und Kantenbedingungen genügen.

In den folgenden zwei Abschnitten stellen wir ein zweistufiges Transformationsverfahren vor, das aus einer gegebenen Pfadbedingungen einer XXL–Suchbedingungen eine graphbasierte, knotenmarkierte Darstellungen erzeugt. Im Abschnitt 4.3.1 geben wir zunächst die Transformation einer textbasierten Pfadbedingung in ein kantenmarkiertes Übergangsdiagramm an. Im Abschnitt 4.3.2 beschreiben wir ein Verfahren, mit dem man ein kantenmarkiertes Übergangsdiagramm in einen äquivalenten knotenmarkierten Pfadbedingungsgraphen transformiert.

4.3.1 Übergangsdiagramm für eine Pfadbedingung

Jede XXL–Suchbedingung besteht minimal aus einer Pfadbedingung $Q \in L(G_P)$ (siehe Definition 4.4). Ohne Beschränkung der Allgemeinheit nehmen wir an, dass das Wildcardzeichen '#' als atomare Pfadbedingung betrachtet wird und nicht durch den Ausdruck '(%)*' ersetzt wird. Eine Pfadbedingung ist also ein regulärer Ausdruck über der Menge $K \cup \{\#\}$. Folglich ist eine atomare Pfadbedingung entweder eine Knotenbedingung aus K oder das Wildcardzeichen '#' oder das leere Wort.

In diesem Abschnitt geben wir eine Methode an, mit der wir aus einer Pfadbedingung in Textform einen gerichteten, kantenmarkierten Graphen (ein Übergangsdiagramm)

konstruieren können. Wir geben zunächst die Definition eines allgemeinen Übergangsdiagramms für Pfadbedingungen an.

Definition 4.13 (Übergangsdiagramm für eine Pfadbedingung)
Sei $\mathcal{K}$ die Menge der Knotenbedingungen und $L(G_P)$ die Menge der Pfadbedingungen über dem Alphabet $\Sigma_P = \mathcal{K} \cup \Sigma_{META}$. Ein Übergangsdiagramm für eine Pfadbedingung ist ein endlicher, gerichteter, kantenmarkierte Graph $D = (V, E, L(G_P), s, e)$, wobei

- *V eine endliche Menge von Knoten,*
- *$E \subseteq V \times V$ eine endliche Menge von markierten Kanten,*
- *$L(G_P)$ die Menge der möglichen Kantenmarkierungen,*
- *$s \in V$ einen Startknoten und*
- *$e \in V$ einen Endknoten darstellt.*

Die Markierungsfunktion $f_E : E \rightarrow L(G_P)$ weist jeder Kante eine Markierung zu.

Für $a \in L(G_P)$ ist ein a–Pfad in D ein Pfad $p = \langle v_1 \dots v_n \rangle$, so dass die Konkatenation der Kantenmarkierungen dieses Pfades $mark_E(p) = f_E(v_1, v_2)/\dots/f_E(v_{n-1}, v_n) = a$ ist.

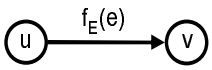
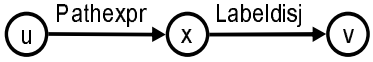
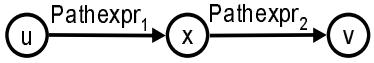
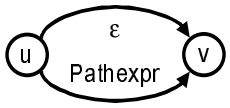
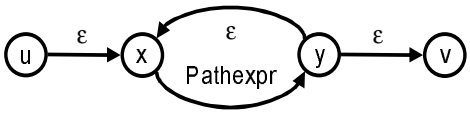
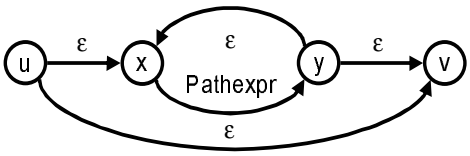
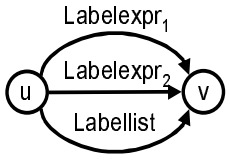
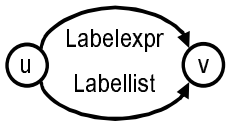
Die von dem Übergangsdiagramm akzeptierte Sprache ist $L(D) = \{a \in L(G_P) \mid \exists a$–Pfad $p = \langle s \dots e \rangle\}$. □

Ein *initiales Übergangsdiagramm* D'_Q für eine Pfadbedingung $Q \in L(G_P)$ besteht aus genau zwei Knoten, dem Start– und dem Endknoten, und einer Kante. Diese Kante ist mit dem regulären Ausdruck Q beschriftet.

Wir bezeichnen ein Übergangsdiagramm als *finales Übergangsdiagramm* D_Q für die Pfadbedingung Q, wenn alle Kanten mit atomaren Pfadbedingungen aus $\mathcal{K} \cup \{\#, \varepsilon\}$ beschriftet sind.

Nun benötigen wir Termersetzungs– und Graphtransformationsregeln, mit deren Hilfe wir das initiale Übergangsdiagramm, das eine einzige mit einer Pfadbedingung beschrifteten Kante enthält, in ein finales Übergangsdiagramm transformieren können, das ausschließlich Kantenbeschriftungen aus $\mathcal{K} \cup \{\#, \varepsilon\}$ enthält. Dazu betrachten wir die Produktionen der Grammatik G_P aus Abschnitt 4.2.2 und definieren die folgenden Ersetzungsregeln.

Definition 4.14 (Ersetzungsregeln)
Es gibt sieben Ersetzungsregeln $t \subseteq (V, E, L(G_P)) \times (V', E', L(G_P))$*, die eine Kante des aktuellen Übergangsdiagramms und seine Beschriftung durch neue Kanten mit neuen Beschriftungen wie in der folgenden Tabelle dargestellt ersetzen.*

Vorgabe	Ersetzungsregel		
Kante e=(u,v) mit $f_E(e) \in L(G_P)$ 	neue Kanten e‘ mit $f_{E'}(e') \in L(G_P)$ und ggf. neue Knoten x,y		
$f_E(e)$	e‘	$f_{E'}(e')$	Graphbasierte Darstellung
(1) Pathexpr/Labeldisj	(u,x) (x,v)	Pathexpr Labeldisj	
(3) Pathexpr_1/Pathexpr_2	(u,x) (x,v)	Pathexpr_1 Pathexpr_2	
(4) (Pathexpr)?	(u,v)	Pathexpr ε	
(4) (Pathexpr)+	(u,x) (x,y) (y,x) (y,v)	ε Pathexpr ε ε	
(4) (Pathexpr)*	(u,x) (x,y) (y,x) (y,v) (u,v)	ε Pathexpr ε ε ε	
(5) (Labelexpr_1 | Labelexpr_2 Labellist)	(u,v)	Labelexpr1 Labelexpr2 Labellist	
(6) | Labelexpr Labellist	(u,v)	Labelexpr Labellist	

□

Die Transformation eines initialen Übergangsdiagramms in ein finales Übergangsdiagramm basiert auf der schrittweisen Anwendung der gegebenen Ersetzungsregeln. Eine Ersetzungsregel wird auf eine Kante und seine Beschriftung dann angewendet, wenn die linke Seite der Regel mit der Beschriftung der Kante übereinstimmt.

In unserem Ersetzungssystem können verzweigende Ersetzungen auftreten. Eine *verzweigende Ersetzung* liegt dann vor, wenn zu einem konkreten Zeitpunkt und einem gegebenen Übergangsdiagramm zwei oder mehrere Ersetzungsregeln angewandt werden können, um mit der Transformation fortzufahren.

Diese Transformation besitzt drei wichtige Eigenschaften.

1. *Korrektheit:* Durch die Anwendung einer beliebigen Ersetzungsregel auf eine Kante und seine Beschriftung, die eine Pfadbedingung gemäß $L(G_P)$ ist, entstehen immer ein oder mehrere neue Kanten, deren Beschriftungen ebenfalls Pfadbedingungen aus $L(G_P)$ sind.
2. *Termination:* Nach endlich facher Anwendung der Ersetzungsregeln entsteht aus einem gegebenen initialen Übergangsdiagramm immer ein finales Übergangsdiagramm.
3. *Konfluenz:* Im Falle von verzweigenden Ersetzungen hat die Reihenfolge der Ersetzungsregeln keinen Einfluss auf das Ergebnis.

im Folgenden werden wir zeigen, dass das hier angegebene Ersetzungssystem tatsächlich korrekt arbeitet und die Terminations– und Konfluenzeigenschaft besitzt. Wir orientieren uns dabei an den Definitionen zur Termination und Konfluenz von zweistelligen Relationen aus [Ave95].

Aus der Definition 4.4 der Grammatik wird ersichtlich, dass die linken Seiten der Ersetzungsregeln und jede Kantenbeschriftung der rechten Seiten dieser Regeln zulässige Pfadbedingungen aus $L(G_P)$ darstellen. Damit ist offensichtlich, dass keine Ersetzungsregel eine unzulässige Zwischendarstellung erzeugt.

Satz 4.2 (Termination)
Die Ersetzungsregeln für die Menge der Übergangsdiagramme sind terminierend. □

Beweis:
Sei eine beliebige Pfadbedingung $Q \in L(G_P)$ gegeben. Sei D'_Q das zugehörige initiale Übergangsdiagramm.

Wir müssen zeigen, dass es keine unendliche Folge von anwendbaren Ersetzungsregeln bei der Transformation beginnend bei D'_Q geben kann.

Jede Ersetzungsregel reduziert die ursprüngliche Kantenbeschriftung durch Zerlegung in Teilpfadbedingungen, die jeweils einer Kante zugeordnet sind bzw. durch Entfernen von Metazeichen, die ihrer Bedeutung gemäß durch entsprechende ε–Kanten umgesetzt werden.

Durch die Endlichkeit der vorgegebenen Pfadbedingung Q, durch die Korrektheit der Ersetzungsregeln und durch die Reduktionseigenschaft dieser Regeln, führt jede Transformation nach endlicher Anwendung der Regeln zu einem finalen Übergangsdiagramm. Insbesondere gibt es nur Ersetzungsregeln für zusammengesetzte Pfadbedingungen, d.h. atomare Pfadbedingungen werden nicht weiter zerlegt. Die Transformation terminiert also genau dann, wenn es im aktuellen Übergangsdiagramm keine Kante mehr gibt, die mit einer zusammengesetzten Pfadbedingung beschriftet ist. Ein Übergangsdiagramm, das ausschließlich Kanten besitzt, die mit atomaren Pfadbedingungen beschriftet sind, hatten wir eingangs als finales Übergangsdiagramm bezeichnet. □

Die Konfluenz eines Ersetzungssystems besagt, dass verzweigende Ersetzungen wieder zusammengeführt werden können, somit also die Reihenfolge der Ersetzungen für das

Ergebnis ohne Belang ist. Dazu benötigen wir den Begriff des kritischen Paares. Zwei Ersetzungsregeln werden als *kritisches Paar* bezeichnet, wenn ihre linken Seiten überlappen. In unserem Ersetzungssystem gibt es kritische Paare, z.B. Regel 3 und Regel 4. Hierbei gibt es eine Besonderheit. Gibt es eine verzweigende Ersetzung, bei der die Wahl zwischen Regeln von kritischen Paaren getroffen werden muss, dann ist die linke Seite der ersten Regel vollständig in der linken Seite der zweiten Regel des kritischen Paares enthalten und die Anwendung der zweiten Regel verändert die in ihr enthaltene linke Seite der ersten Regel nicht.

Beispiel 4.9 Das folgende Beispiel gibt eine verzweigende Ersetzung für ein Übergangsdiagramm an, bei der wahlweise mit Regel 3 oder mit Regel 4 fortgefahren werden kann. Die Kante hat folgende Beschriftung:

```
(act/scene)?
```

Hierbei ist die linke Seite der Regel 3 Bestandteil der linken Seite der Regel 4 und die Anwendung der Regel 4 werden nur die Metazeichen entfernen, aber nicht die Pfadbedingung zwischen den Klammern.

Zur Lösung unserer Konfluenzfrage befassen wir uns zunächst mit der lokalen Konfluenz des hier vorgestellten Ersetzungssystems. Die lokale Konfluenz der Ersetzungsregeln lässt sich mit Hilfe folgendes Lemmas beweisen.

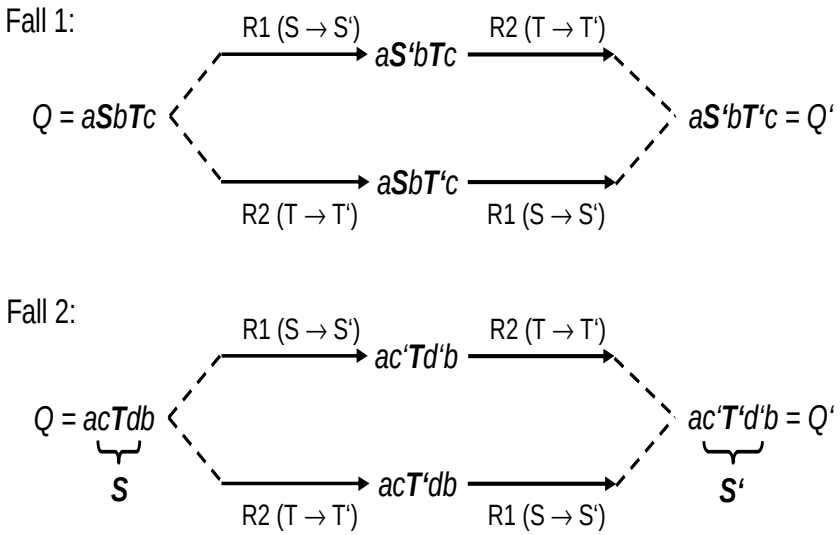
Lemma 4.1 (Lokale Konfluenz)
Die Ersetzungsregeln sind dann lokal konfluent, wenn jedes kritische Paar zusammenführbar ist.

Beweis
Sei $Q \in L(G_P)$ eine beliebige Pfadbedingung und seien R1, R2 zwei auf Q anwendbare Ersetzungsregeln.

Wir zeigen nun, dass bei einer verzweigenden Ersetzung die Anwendung einer Regel nie die Anwendbarkeit der anderen Regel verändert. Dazu unterscheiden wir die folgenden beiden Fälle:

1. die Pfadbedingung $Q = aSbTc$ enthält zwei Teilpfadbedingungen S und T, auf die die Regeln R1 und R2 angewendet werden können (R1 und R2 sind kein kritisches Paar),
2. die Pfadbedingung $Q = aSb$ enthält die Teilpfadbedingung S und die Teilpfadbedingung $S = cTd$ enthält die Teilpfadbedingung T (R1 und R2 sind ein beliebiges kritisches Paar).

Nehmen wir an, dass die Ersetzungsregeln R1 und R2 die Teilpfadbedingungen S und T durch die neuen reduzierten Teilpfadbedingungen S' und T' ersetzen. Für die beiden möglichen verzweigenden Ersetzungen ergeben sich folgende Ergebnisse für die Anwendung der Regel R1 vor bzw. nach der Regel R2:



Es ist offensichtlich, dass die Anwendung der einen Regel die Anwendbarkeit der anderen Regel nicht verändert. Damit ist jede verzweigende Ersetzung (kritisch oder unkritisch) direkt zusammenführbar, also lokal konfluent. □

Da das hier definierte Ersetzungssystem terminierend und lokal konfluent ist, ist es gemäß des Newman–Lemmas [Ave95] auch konfluent. Damit haben wir gezeigt, dass die Reihenfolge der Ersetzungsregeln keinen Einfluss auf die Struktur des finalen Übergangsdiagramms hat.

Mit Hilfe der Ersetzungsregeln sind wir in der Lage, für eine in Textform gegebene Pfadbedingung ein gerichtetes, kantenmarkiertes finales Übergangsdiagramm zu erzeugen.

Beispiel 4.10 Für die XXL–Anfrage aus dem Beispiel 4.1 geben wir in der Abb. 4.5 die zugehörigen finalen Übergangsdiagramme an. Aus Übersichtlichkeitsgründen haben wir hier die Knotenbezeichnungen weggelassen.

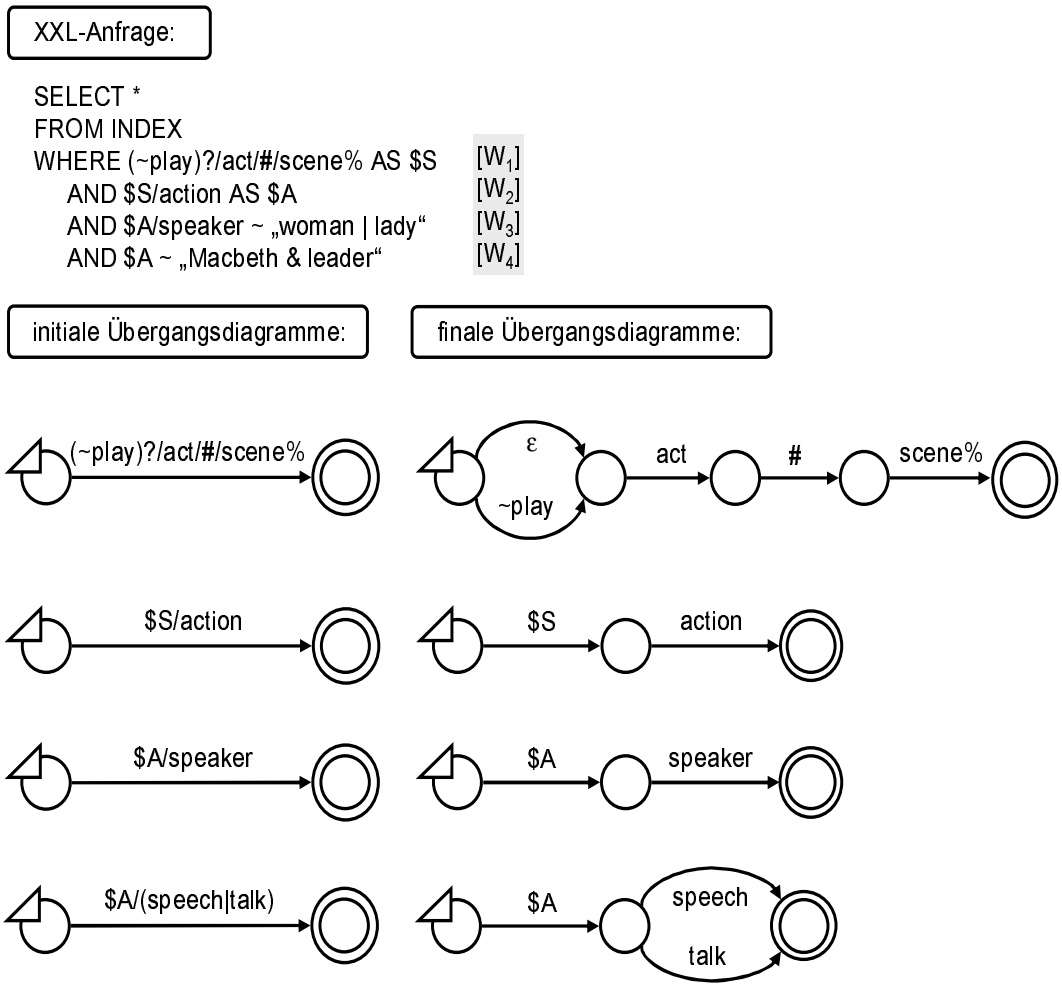


Abbildung 4.5: Finale Übergangsdiagramme D_{Q_1} bis D_{Q_4} zu den vier Pfadbedingungen der vier XXL–Suchbedingungen der gegebenen XXL–Anfrage

4.3.2 Pfadbedingungsgraph für eine Pfadbedingung

Sei Q die Pfadbedingung einer XXL–Suchbedingung und sei D_Q das zugehörige finale Übergangsdiagramm, das die von Q beschriebene reguläre Sprache akzeptiert. In diesem Abschnitt stellen wir ein Verfahren vor, das aus dem gerichteten, kantenmarkierten finalen Übergangsdiagramm ein gerichteten, knotenmarkierten Pfadbedingungsgraphen erzeugt, der dieselbe reguläre Sprache akzeptiert.

Das Verfahren beruht auf der Idee, für jede nicht mit ε beschriftete Kante im Übergangsdiagramm einen Knoten mit gleicher Beschriftung im Pfadbedingungsgraphen anzulegen und die ε–Pfade zwischen diesen Kanten im Übergangsdiagramm durch gerichtete Kanten zwischen den zugehörigen Knoten im Pfadbedingungsgraphen zu ersetzen. Das Ziel besteht darin, die Informationen über Kantenbeschriftungen ungleich ε sowie Start– und Endknoten in den knotenmarkierten Pfadbedingungsgraphen zu übertragen, so dass in diesem keine ε–beschrifteten Knoten vorkommen. Dazu müssen wir folgende Fälle beachten.

1. Für jede nicht mit ε beschriftete Kante wird ein entsprechender Knoten mit gleicher Beschriftung angelegt.
2. Für jeden ε–Pfad zwischen zwei nicht mit ε beschrifteten Kanten wird im Pfadbedingungsgraphen eine direkte Kante zwischen den entsprechenden zugehörigen

Knoten gesetzt.

3. Die Startknoten im Pfadbedingungsgraphen ergeben sich aus den Pfaden vom Startknoten im Übergangsdiagramm.

   - Für jede Kante vom Startknoten im Übergangsdiagramm, die nicht mit ε beschriftet ist, wird der zugehörige Knoten im Pfadbedingungsgraphen zum Startknoten erklärt.
   - Für jeden ε–Pfad im Übergangsdiagramm vom Startknoten zu einer nicht mit ε beschrifteten Kante wird der zugehörige Knoten im Pfadbedingungsgraphen zum Startknoten erklärt.

4. Die Endknoten im Pfadbedingungsgraphen werden in ähnlicher Weise aus den Pfaden zum Endknoten ermittelt.

Ein Pfadbedingungsgraph für eine gegebene Pfadbedingung Q wird aus den Kantenbeschriftungen und Knotentypen eines finalen Übergangsdiagramms gemäß folgender Definition erzeugt.

Definition 4.15 (Pfadbedingungsgraph)
Sei $Q \in L(G_P)$ eine Pfadbedingung und $D_Q = (V_Q, E_Q, L(G_P), s_Q, e_Q)$ ein finales gerichtetes, kantenmarkiertes Übergangsdiagramm, das die Sprache $L(D_Q)$ akzeptiert. Der Pfadbedingungsgraph für Q ist ein gerichteter, knotenmarkierter Graph $C = (V, E, L(G_P), V', V'')$ mit:

- *einer endlichen Menge V von markierten Knoten,*
- *einer endlichen Menge $E \subseteq V \times V$ von gerichteten Kanten,*
- *$L(G_P)$ die Menge der möglichen Kantenmarkierungen,*
- *einer endlichen Menge $V' \subseteq V$ von Startknoten (Quellen) und*
- *einer endlichen Menge $V'' \subseteq V$ von Endknoten (Senken).*

Die Markierungsfunktion $f_V : V \rightarrow L(G_P)$ weist jedem Knoten eine Markierung zu. Die Knoten, Kanten, Start– und Endknoten werden aus den Vorgaben des Übergangsdiagramms wie folgt ermittelt.

1. *Für jede Kante $e = (x, y) \in E_Q$ im Übergangsdiagramm mit der Beschriftung $f_{E_Q}(e) = m$ $(m \neq \varepsilon)$ gibt es einen Knoten $z \in V$ im Pfadbedingungsgraphen mit der gleichen Beschriftung $f_V(z) = m$.*
2. *Für jeden Pfad $p = \langle v_1 \ldots v_k \rangle$ im Übergangsdiagramm mit der Beschriftung $mark_{E_Q}(p) = m_1\ \varepsilon \ldots \varepsilon\ m_2$ und die zugehörigen Knoten $x, y \in V$ im Pfadbedingungsgraphen mit den Beschriftungen $f_V(x) = m_1, f_V(y) = m_2$ gemäß Punkt 1 gibt es eine Kante $(x, y) \in E$ im Pfadbedingungsgraphen.*
3. *Für jede Kante $e = (s_Q, x) \in E_Q$ vom Startknoten s_Q im Übergangsdiagramm mit einer Beschriftung $f_{E_Q}(e) = m$ $(m \neq \varepsilon)$ wird der zugehörige Knoten im Pfadbedingungsgraphen zum Startknoten erklärt und in die Menge V' eingefügt.*
4. *Für jeden Pfad $p = \langle s_Q \ldots v \rangle$ im Übergangsdiagramm beginnend beim Startknoten s_Q mit der Beschriftung $mark_{E_Q}(p) = \varepsilon \ldots \varepsilon\ m$ $(m \neq \varepsilon)$ wird der zugehörige Knoten im Pfadbedingungsgraphen zum Startknoten erklärt und in die Menge V' eingefügt.*

5. *Für jede Kante $e = (x, e_Q) \in E_Q$ zum Endknoten e_Q im Übergangsdiagramm mit einer Beschriftung $f_{E_Q}(e) = m$ $(m \neq \varepsilon)$ wird der zugehörige Knoten im Pfadbedingungsgraphen zum Endknoten erklärt und in die Menge V'' eingefügt.*
6. *Für jeden Pfad $p = \langle v \dots e_Q \rangle$ im Übergangsdiagramm endend beim Endknoten e_Q mit der Beschriftung $mark_{E_Q}(p) = m\ \varepsilon \dots \varepsilon$ $(m \neq \varepsilon)$ wird der zugehörige Knoten im Pfadbedingungsgraphen zum Endknoten erklärt und in die Menge V'' eingefügt.*

Für $a \in \Sigma_P^*$ ist ein a–Pfad in C ein Pfad $p = \langle v_1 \dots v_n \rangle$, so dass die Konkatenation der Knotenmarkierungen dieses Pfades $mark_V(p) = f_V(v_1)/\dots/f_V(v_n) = a$ ist.

Die von dem Pfadbedingungsgraphen akzeptierte Sprache ist $L(C) = \{a \in \Sigma_P^* \mid \exists$ a–Pfad $p = \langle s \dots e \rangle$ wobei $s \in V', e \in V''\}$. □

Sei p ein Pfad von einem Start– zu einem Endknoten in einem gegebenen Pfadbedingungsgraphen. im Folgenden werden wir Pfade wie p als *s/e–Pfade* bezeichnen. Ein Pfadbedingungsgraph akzeptiert also solche Pfade, die mit einem seiner s/e–Pfad übereinstimmen.

Beispiel 4.11
In der folgenden Abbildung 4.6 zeigen wir die Pfadbedingungsgraphen für die Pfadbedingungen der gegebenen XXL–Suchbedingungen der XXL–Anfrage aus Abb. 4.1.

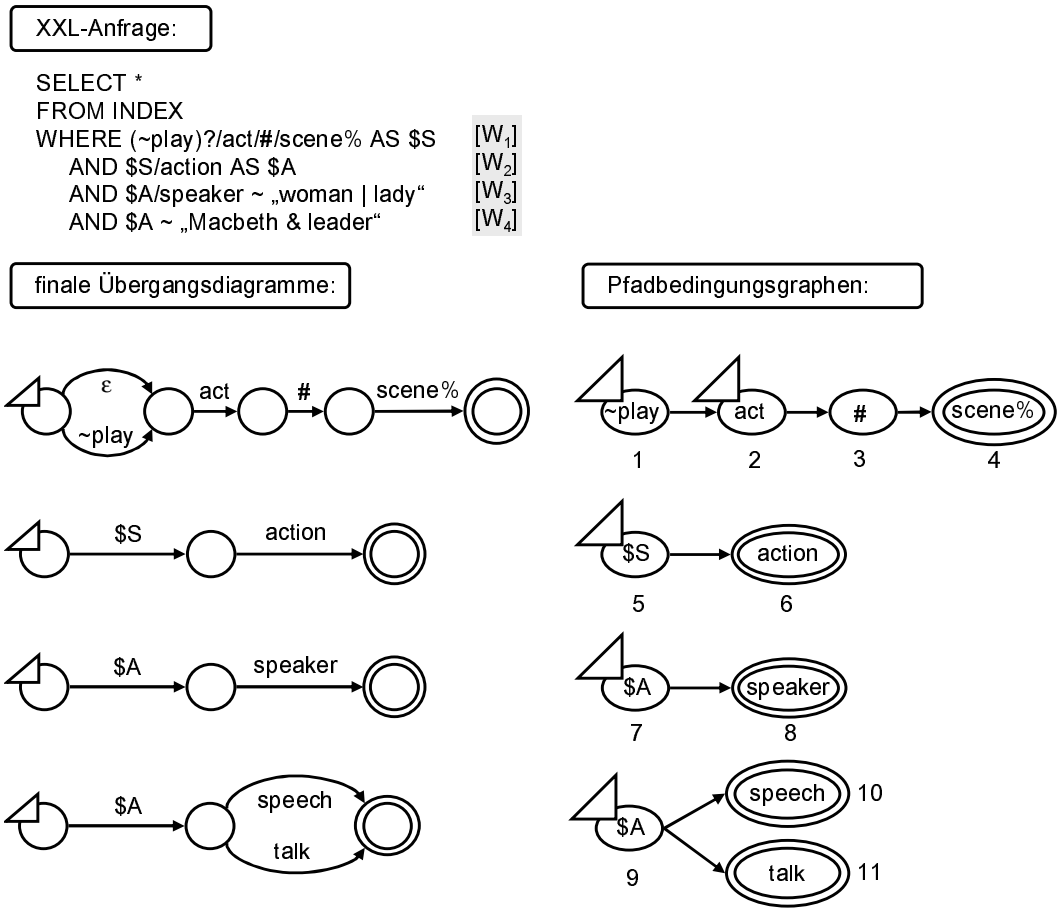


Abbildung 4.6: Konstruktion der Pfadbedingungsgraphen aus den finalen Übergangsdiagrammen der gegebenen Pfadbedingungen

An dieser Stelle müssen wir den Beweis führen, dass die von einem so konstruierten Pfadbedingungsgraphen akzeptierte Sprache äquivalent zu der Sprache ist, die vom gegebenen Übergangsdiagramm akzeptiert wird. Dazu zeigen wir, dass aus einem Pfad p im Übergangsdiagramm mit der Kantenbeschriftung $mark_{E_Q}(p)$ ein Pfad p' im Pfadbedingungsgraphen mit der Knotenbeschriftung $mark_V(p')$ konstruiert wird, für den gilt $mark_V(p') = mark_{E_Q}(p)$.

Satz 4.3
Sei $D_Q = (V_Q, E_Q, \Sigma_D, s_Q, e_Q)$ das Übergangsdiagramm für eine Pfadbedingung Q. Sei $C = (V, E, \Sigma_P^*, V', V'')$ der Pfadbedingungsgraph für Q. Dann gilt für die jeweils akzeptierten Sprachen $L(D_Q) = L(C)$. □

Beweis:
Teil 1: $L(D_Q) \subseteq L(C)$, d.h. $a \in L(D_Q) \Rightarrow a \in L(C)$

Sei $a \in \Sigma_P^*$ und $a \in L(D_{E_Q})$.

Dann gibt es gemäß Definition 4.13 einen Pfad $p = \langle s_Q \dots e_Q \rangle$ mit der Kantenbeschriftung $mark_{E_Q}(p) = a$.

Sei $a = a_1 a_2 \dots a_k$ die Konkatenation der Kantenbeschriftungen $a_i \in \Sigma_D$ mit $a_i \neq \varepsilon$. Seien weiterhin $e_1, e_2, \dots, e_k$ die zugehörigen Kanten, also $mark_{E_Q}(e_i) = a_i$. Die Reihenfolge der Kanten $e_i = (u_i, v_i)$ auf dem Pfad p ist durch die Reihenfolge der Kantenbeschriftungen a_i fest vorgegeben.

Wir zeigen nun, dass zum einen der gemäß Definition 4.15 erzeugte Pfadbedingungsgraph keine anderen Knotenbeschriftungen außer den gegebenen a_i haben kann und zum anderen, dass die Reihenfolge der Knotenbeschriftungen identisch zur Reihenfolge der Kantenbeschriftungen ist.

Entsprechend der Punkte 1 und 2 der Definition 4.15 gibt es im Pfadbedingungsgraphen die mit ε beschrifteten Start– und Endknoten sowie die mit a_i beschrifteten Knoten x_i. Es gibt keine weiteren Knoten mit anderen Beschriftungen.

Gemäß dem Punkt 4 der Definition 4.15 gibt es im Pfadbedingungsgraphen eine Kante vom Startknoten zum Knoten x_1, da es lt. Voraussetzung einen Teilpfad $\langle s_Q \dots u_1 v_1 \rangle$ von p mit der Beschriftung a_1 gibt.

Analog gibt es im Pfadbedingungsgraphen eine Kante vom Knoten x_k zum Endknoten, da es lt. Voraussetzung einen Teilpfad $\langle u_k v_k \dots e_Q \rangle$ von p mit der Beschriftung a_k gibt.

Insbesondere gibt es entsprechend des Punktes 3 der Definition 4.15 jeweils eine Kante von x_i nach x_{i+1}, da es lt. Voraussetzung einen Teilpfad $\langle u_i v_i \dots u_{i+1} v_{i+1} \rangle$ in p mit der Beschriftung $a_i a_{i+1}$ gibt.

Da es auf einem Pfad zwischen zwei Knoten höchstens eine Kante gibt, kann es im Pfadbedingungsgraphen keine weiteren außer den hier genannten Kanten geben. Somit entsteht ein vorläufiger Bedingungsgraph bestehend aus einem Pfad p' mit der Beschriftung $\varepsilon a_1 \dots a_k \varepsilon$. Also ist $a \in L(C)$.

Teil 2: $L(D_Q) \supseteq L(C)$, d.h. $a \in L(C) \Rightarrow a \in L(D_Q)$

Sei $a \in \Sigma_P^*$ und $a \in L(C)$.

Dann gibt es gemäß Definition 4.15 einen Pfad $p = \langle s \dots e \rangle$ mit der Knotenbeschriftung $mark_E(p) = \varepsilon a \varepsilon = a$.

Sei $a = a_1 a_2 \dots a_k$ die Konkatenation der Knotenbeschriftungen $a_i \in \Sigma_D$ mit $a_i \neq \varepsilon$ der Knoten x_i.

Wir zeigen, dass das ursprüngliche Übergangsdiagramm einen Pfad p' mit der Kantenmarkierung a enthalten haben muss.

Das Übergangsdiagramm besteht minimal aus dem Start– und dem Endknoten.

Wegen Punkt 2 der Definition 4.15 gab es für jeden Knoten mit der Beschriftung a_i eine Kante im Übergangsdiagramm mit derselben Beschriftung. Sei $e_i = (u_i, v_i)$ diese Kante.

Da es eine Kante (s, x_1) im Pfadbedingungsgraphen gibt, muss es im Übergangsdiagramm einen ε–Weg vom Startknoten s_Q zum Knoten u_1 gegeben haben, oder $s_Q = u_1$.

Analog muss es im Übergangsdiagramm einen ε–Weg vom Knoten v_k zum Endknoten oder $v_k = e_Q$ gegeben haben.

Für jede Kante (x_i, x_{i+1}) im Pfadbedingungsgraphen muss es im Übergangsdiagramm einen Teilpfad $\langle e_i e_{i+1} \rangle$ oder einen Teilpfad der Form $\langle e_i p' e_{i+1} \rangle$ gegeben haben, wobei p' ein ε–Weg ist.

Folglich gab es im Übergangsdiagramm einen Pfad p' mit der Beschriftung a, wobei die ε–Wege keinen Einfluss auf die Kantenmarkierung haben sondern lediglich darauf hinweisen, dass die Rückverfolgung des Ursprungs nicht eindeutig in seiner Struktur wohl aber in seiner Beschriftung ist. □

4.4 Semantik

Klassisches Boolesches Information Retrieval in XML–Dokumenten produziert eine unsortierte Liste von relevanten Treffern (XML–Dokumente bzw. Ausschnitte aus XML–Dokumenten). Dabei gibt es nur die beiden Möglichkeiten, ein Treffer ist relevant/zutreffend oder er ist nicht relevant/zutreffend. Dieser Ansatz ist für eine begrenzte Dokumentkollektion durchaus ausreichend. Für die Informationssuche in sehr umfangreichen, vielschichtigen Mengen von XML–Dokumenten besteht verstärkt das Interesse daran, auch partiell relevante Treffer zu finden.

Im Rahmen dieser Arbeit definieren wir die Semantik von XXL–Anfragen mit dem Ziel, ranglistenbasiertes Information Retrieval in XML–Dokumenten zu unterstützen. Hierbei wird als Ergebnis einer Anfrage eine Rangliste von Treffern produziert (XML–Dokumente bzw. Ausschnitte aus XML–Dokumenten), die nach ihrer Relevanz absteigend sortiert ist.

Im Mittelpunkt der Semantikdefinition steht damit die Relevanzbewertung von Treffern, wobei die Relevanz eines Treffers inhärent subjektiv und vage und sehr schwer automatisch zu erfassen ist.

Die XML–Dokumente, in denen nach Informationen gesucht werden soll, werden durch den gerichteten XML–Graphen $X = (V_X, E_X, \Sigma^*)$ repräsentiert (Kapitel 2). Eine XXL–Anfrage der Form `SELECT S FROM F WHERE W1 AND ... AND Wk` besteht aus k konjunktiv verknüpften XXL–Suchbedingungen (Abschnitt 4.2). Eine XXL–Suchbedingung besteht dabei entweder aus einer Pfadbedingung oder aus einer Pfadbedingung und einer Inhaltsbedingung oder aus einer Pfadbedingung mit einer Variablenbindung.

Die Semantik einer XXL–Anfrage legt induktiv fest, wann Knoten, Pfade bzw. Teilgraphen im XML–Graphen den Pfadbedingungen, Inhaltsbedingungen und Variablenbindungen der gegebenen konjunktiv XXL–Suchbedingungen genügen. Dazu gehen wir wie folgt vor.

- Im Abschnitt 4.4.1 definieren wir den *lokalen Relevanzwert* eines n–Knotens, der sich aus dem Vergleich seines Element–/Attributnamens mit einer gegebenen atomaren Pfadbedingung ergibt.
- Im Abschnitt 4.4.2 definieren wir *relevante Pfade*, die sich aus einer Menge von relevanten n–Knoten zusammensetzen und einer gegebenen Pfadbedingung genügen.
- Ist eine Pfadbedingung an eine Variable gebunden, dann wird diese Variable mit dem letzten n–Knoten eines relevanten Pfades belegt. Die zugehörige Definition einer Variablenbelegungen wird im Abschnitt 4.4.3 angegeben.
- Ist eine Pfadbedingung an eine Inhaltsbedingung geknüpft, bedeutet es, dass der zum letzten n–Knoten eines dazu relevanten n–Pfades gehörende Element– oder Attributwert dieser Bedingung genügen muss. Dazu werden die Bedeutungen der binärer Operatoren und der Vergleichsobjekte (Variable oder Stringausdruck) im Abschnitt 4.4.4 festgelegt.
- Im Abschnitt 4.4.5 definieren wir den *Resultatgraph* als einzelnen Treffer für eine gegebene XXL–Anfrage. Dieser setzt sich aus je einem relevanten Pfad pro XXL–Suchbedingung zusammen, wobei diese einer gemeinsamen Variablenbelegung genügen müssen. Die Relevanz des Resultatgraphen ergibt sich aus den lokalen Relevanzwerten der beteiligten Knoten.

Das Ergebnis einer gegebenen XXL–Anfrage ist somit eine Rangliste von Resultatgraphen, die nach ihrer Relevanz absteigend sortiert sind. Auf der Basis der Select–Klausel wird daraus das tatsächlich gewünschte Ergebnis extrahiert und an den Benutzer zurückgegeben.

4.4.1 Lokaler Relevanzwert eines n–Knoten

Jede XXL–Suchbedingung W enthält eine Pfadbedingung, die durch den zugehörigen Pfadbedingungsgraphen $C = (V, E, L(G_P), V', V'')$ repräsentiert wird. Ein n–Pfad im XML–Graphen $X = (V_X, E_X, \Sigma^*)$ wird von dem Pfadbedingungsgraphen akzeptiert, genügt also der gegebenen Pfadbedingung, wenn er mit einem s/e–Pfad (Start/End–Pfad) des Pfadbedingungsgraphen übereinstimmt.

Die Übereinstimmung eines n–Pfades mit einem s/e–Pfad beruht dabei auf der Übereinstimmung der Knoten und Kanten. In diesem Abschnitt definieren wir die Übereinstimmung von Knoten auf der Basis des Vergleichs von Element– bzw. Attributnamen mit gegebenen atomaren Pfadbedingungen. Der Grad der Übereinstimmung soll dabei durch den lokalen Relevanzwert ausgedrückt werden.

Sei p ein s/e–Pfad des gegebenen Pfadbedingungsgraphen C. Sei $x \in V$ ein Knoten in p. Sei weiterhin $y \in V_X$ ein n–Knoten im XML–Graphen. Die Relevanz des n–Knotens y für die gegebene XXL–Suchbedingung ergibt sich aus dem Vergleich seines Element– bzw. Attributnamens mit der atomaren Pfadbedingung des Knotens x. Dazu definieren wir die Funktion $\pi : V \times V_X \rightarrow [0, 1]$, die einem n–Knoten des XML–Graphen X in Abhängigkeit von der gegebenen atomaren Pfadbedingung durch einen Knoten des Pfadbedingungsgraphen einen *lokalen Relevanzwert* zwischen 0 und 1 zuweist.

Gemäß der Definition 4.4 der Grammatik G_P können atomare Pfadbedingungen in vier Formen auftreten,

1. als das #–Zeichen.

2. als eine Variable,
3. als ein Labelausdruck (inkl. der möglichen Wildcardzeichen) oder
4. als ein Labelausdruck mit vorangestelltem Ähnlichkeitsoperator $\sim$,

Für die oben genannten vier Fälle definieren wir nun den lokalen Relevanzwert des n–Knotens y bzgl. der gegebenen atomaren Pfadbedingung durch den Knoten x.

Fall 1: Sei x mit dem #–Zeichen beschriftet. Der n–Knoten y des XML–Graphen erfüllt die gegebene atomare Pfadbedingung unabhängig von seiner Beschriftung und erhält einen lokalen Relevanzwert $\pi(x, y) = 1.0$.

Fall 2: Sei x mit einem Variablennamen gemäß der Definition 4.2 beschriftet, x steht also für ein angewandtes Variablenvorkommen. Dann muss es vorher ein definierendes Vorkommen dieser Variablen gegeben haben, das sie mit einem n–Knoten des XML–Graphen belegt. Ist die Variable mit dem n–Knoten y des XML–Graphen belegt, dann ist die atomare Pfadbedingung erfüllt und erhält einen lokalen Relevanzwert $\pi(x, y) = 1.0$.

Fall 3: Sei x mit einem Labelausdruck gemäß der Definition 4.1 beschriftet, der u.U. die Wildcardzeichen % bzw. ? enthält. Der n–Knoten y des XML–Graphen, dessen Beschriftung lexikalisch mit dem vorgegebenen Labelausdruck unter Berücksichtigung der Wildcardzeichen übereinstimmt, erfüllt diese atomare Pfadbedingung und erhält einen lokalen Relevanzwert $\pi(x, y) = 1.0$.

Fall 4: Sei y mit einem Labelausdruck l gemäß der Definition 4.1 beschriftet, dem der unäre Ähnlichkeitsoperator $\sim$ vorangestellt ist. Diese atomare Pfadbedingung wird schrittweise mit Hilfe der Ontologie wie folgt ausgewertet.

- Der Labelausdruck l wird mit Hilfe der Ontologie einem oder mehreren Konzepten $c(l)$ zugeordnet. (Die Notation $c(l)$ steht für das Konzept c, in dessen Synset das Wort l vorkommt (siehe Abschnitt 3).)
- Zu jedem Konzept $c(l)$ des Labelausdrucks l können nun mit Hilfe der Ontologie semantisch ähnliche Konzepte c' sowie deren Ähnlichkeitswerte $sim(c(l), c')$ ermittelt werden.
- Die Synonyme $t_1, t_2, t_3, \ldots$ der gefundenen semantisch ähnlichen Konzepte werden zur Expansion der atomaren Pfadbedingung "$\sim l$" zum Ausdruck "$l|t_1|t_2|t_3| \ldots$" verwendet.
- Der n–Knoten y, dessen Beschriftung mit einem der gegebenen Wörter $t \in \{l, t_1, t_2, t_3, \ldots\}$ exakt übereinstimmt, erfüllt die ursprünglich gegebene atomare Pfadbedingung "$\sim l$" und erhält einen lokalen Relevanzwert auf der Basis des ontologiebasierten Ähnlichkeitswertes der zugehörigen Konzepte $c(l)$ und $c'(t)$, also $\pi(x, y) = sim(c(l), c'(t))$.

Fall 5: Erfüllt die Beschriftung des n–Knotens y die gegebene atomare Pfadbedingung nicht, dann wird ihm der lokale Relevanzwert $\pi(x, y) = 0$ zugewiesen.

Beispiel 4.12 In der Abb. 4.7 ist links der XML–Graph und rechts die Pfadbedingungsgraphen einer gegebenen XXL–Anfrage dargestellt. Zur Vereinfachung haben wir die jeweils eindeutigen Knotenbezeichnungen durch römische bzw. arabische Ziffern ausgedrückt.

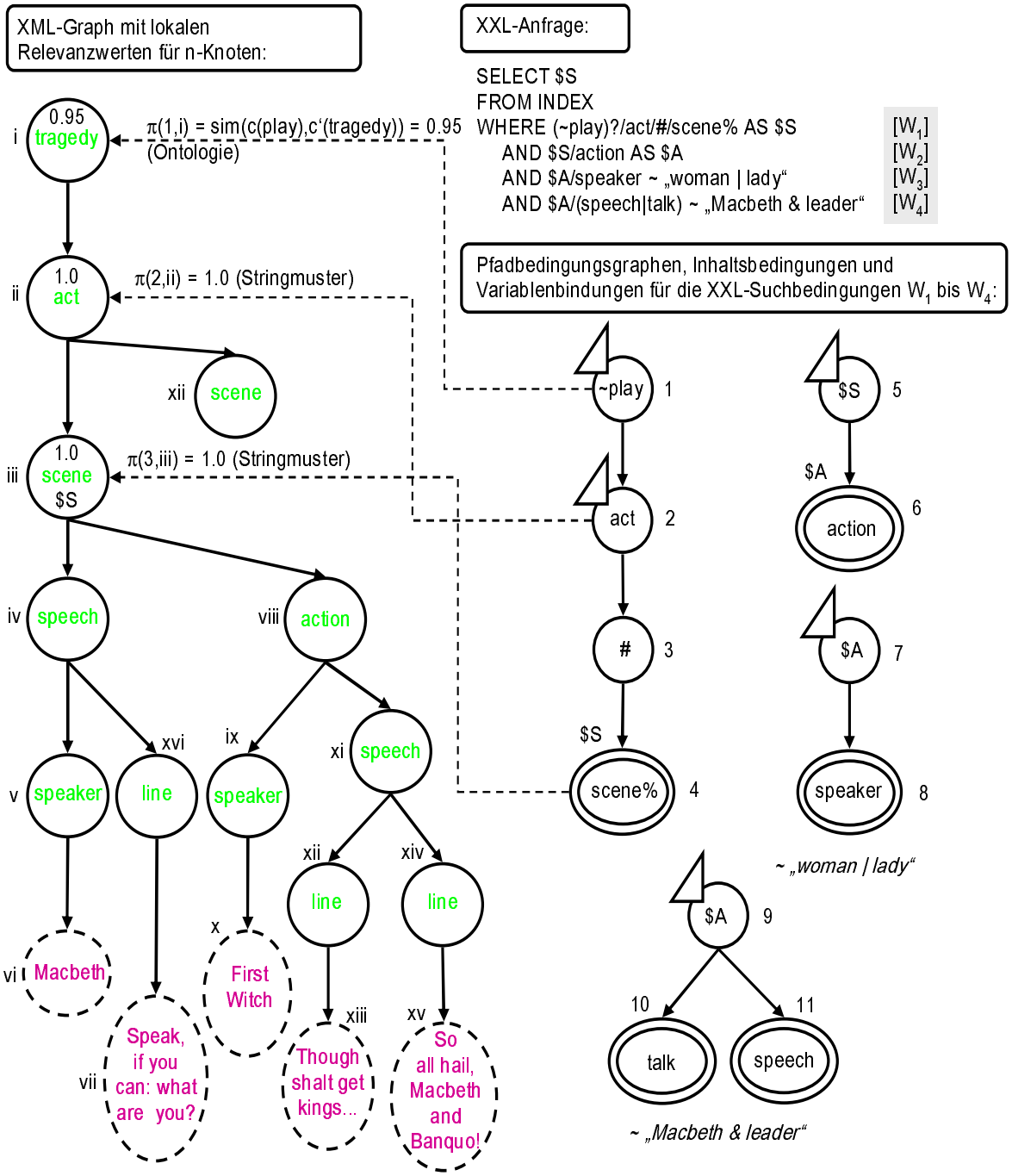


Abbildung 4.7: Lokale Relevanzwerte für n–Knoten auf der Basis des s/e–Pfades p=⟨*1/2/3/4*⟩ des ersten Pfadbedingungsgraphen

Den Wert für die ontologiebasierte Ähnlichkeit der Konzepte von `play` und `tragedy` haben wir hier zur Illustration willkürlich gewählt.

Der Grad der Übereinstimmung eines n–Knotens y im XML–Graphen mit einer atomaren Pfadbedingung eines Knotens x eines s/e–Pfades eines Pfadbedingungsgraphen einer XXL–Suchbedingung wird durch den lokalen Relevanzwert $\pi(x, y) \in [0, 1]$ ausgedrückt.

4.4.2 Relevante Pfade

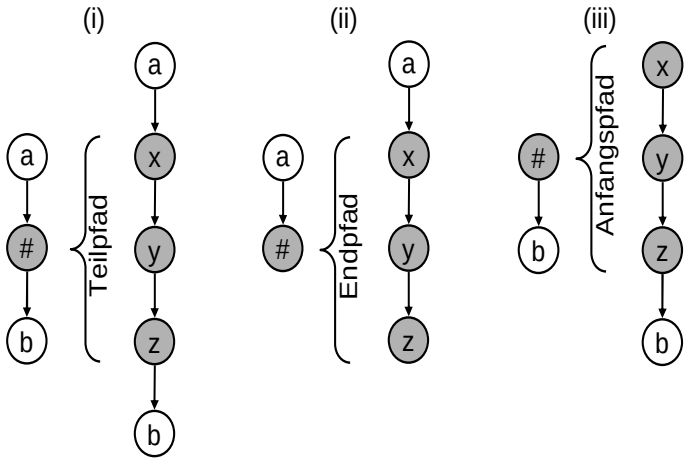
Jede XXL–Suchbedingung W enthält eine Pfadbedingung, die durch den zugehörigen Pfadbedingungsgraphen $C = (V, E, L(G_P), V', V'')$ repräsentiert wird. Ein n–Pfad im XML–Graphen $X = (V_X, E_X, \Sigma^*)$ wird von dem Pfadbedingungsgraphen akzeptiert, genügt also der gegebenen Pfadbedingung, wenn er mit einem s/e–Pfad (Start/End–Pfad) des Pfadbedingungsgraphen übereinstimmt.

Die Übereinstimmung eines n–Pfades mit einem s/e–Pfad wird anhand der Gestalt (Anordnung der beteiligten Knoten) und den Knotenbeschriftungen gemessen. Dazu greifen wir die Idee der Graphisomorphie auf und passen ihn an die gegebene Situation an.

Definition 4.16 (Graphisomorphie)
Zwei Graphen $G = (V, E)$ und $G' = (V', E')$ sind isomorph, wenn es eine bijektive Abbildung $g : V \to V'$ gibt, so dass $(u, v) \in E$ genau dann, wenn $(g(u), g(v)) \in E'$. □

Der Vergleich eines n–Pfades q des XML–Graphen mit einem s/e–Pfad p eines Pfadbedingungsgraphen einer gegebenen XXL–Suchbedingung muss zwei Besonderheiten berücksichtigen.

1. Die Übereinstimmung eines n–Knotens $y \in V_X$ mit einem Knoten $x \in V$ ist dann gegeben, wenn der lokale Relevanzwert $\pi(x, y) > 0$ ist.
2. Die atomare Pfadbedingung # steht als Wildcardzeichen für eine beliebige Folge von n–Knoten. Dabei entspricht # entweder einem Teilpfad oder dem Anfangspfad oder dem Endpfad des betrachteten n–Pfades. Insbesondere haben alle Knoten dieses Teilpfades, Anfangspfades bzw. Endpfades definitionsgemäß 1.0 als lokalen Relevanzwert. In der folgenden Abbildung sind diese drei Fälle dargestellt, wobei jeweils links der s/e–Pfad p und rechts der n–Pfad q steht.



Die *relevanzbasierte Graphisomorphie* berücksichtigt bei dem Vergleich eines n–Pfades q mit einem s/e–Pfad p die Übereinstimmung der n–Knoten, also die lokalen Relevanzwerte, sowie das Vorkommen des Wildcardzeichens #, dass für einen beliebigen Teilpfad, Anfangspfad oder Endpfad steht. Dabei müssen wir beachten, dass # auch für einen Pfad der Länge 0 stehen kann.

Definition 4.17 (Relevanzbasierte Graphisomorphie)
Seien X der XML–Graph und C ein Pfadbedingungsgraph. Sei $q = (V', E')$ ein n–Pfad in X und sei $p = (V, E)$ ein s/e–Pfad des Pfadbedingungsgraphen. Die Funktion

$\pi : V \times V' \to [0,1]$ weist den n–Knoten ihre lokalen Relevanzwerte zu. Der Pfad q ist relevanzbasiert isomorph zum Pfad p, wenn es eine bijektive Abbildung $g : V \to V'$ gibt, so dass folgendes gilt:

- *$(u,v) \in E$ genau dann, wenn $(g(u), g(v)) \in E'$ und $\pi(u, g(u)) > 0$ und $\pi(v, g(v)) > 0$*
- *$(u,\#), (\#,v) \in E$ genau dann, wenn $q' = \langle a_0 \dots a_n \rangle$ ein Teilpfad von q ist, für den gilt: $\pi(\#, a_j) = 1.0$ $(0 \leq j \leq n)$ und $(g(u), a_0) \in E'$ und $(a_n, g(v)) \in E'$*
- *$(u,\#) \in E$ genau dann, wenn $q' = \langle a_0 \dots a_n \rangle$ ein Endpfad von q ist, für den gilt: $\pi(\#, a_j) = 1.0$ $(0 \leq j \leq n)$ und $(g(u), a_0) \in E'$*
- *$(\#,v) \in E$ genau dann, wenn $q' = \langle a_0 \dots a_n \rangle$ ein Anfangspfad von q ist, für den gilt: $\pi(\#, a_j) = 1.0$ $(0 \leq j \leq n)$ und $(a_n, g(v)) \in E'$*

Der n–Pfad q heisst relevanter Pfad mit dem Relevanzwert $rsv(q) \in [0,1]$, der durch Multiplikation der lokalen Relevanzwerte aller beteiligten n–Knoten bestimmt wird. □

Ein n–Pfad im XML–Graphen wird also als relevant bezeichnet, wenn alle seine n–Knoten einen lokalen Relevanzwert größer als 0 haben und die Gestalt des n–Pfades mit der eines s/e–Pfades des gegebenen Pfadbedingungsgraphen übereinstimmt.

Beispiel 4.13 Betrachten wir noch einmal die Abbildung 4.7. Die n–Knoten *i, ii* und *iii* bilden einen relevanten Pfad q mit dem Relevanzwert $\pi(q) = 0.95$ für den ersten Pfadbedingungsgraphen.

Insbesondere gilt die oben definierte relevanzbasierte Graphisomorphie ohne Beschränkung der Allgemeinheit auch für Pfadbedingungen, die das Wildcardzeichen # mehrfach enthalten.

4.4.3 Variablenbelegung

Wir betrachten eine XXL–Suchbedingung, die aus einer Pfadbedingung mit einer Variablenbindung besteht. Die Belegung einer Variablen geschieht durch ihr definierendes Vorkommen. Dabei wird die Variable mit dem letzten n–Knoten eines für diese Pfadbedingung relevanten Pfades belegt. Für die Auswertung bzw. Relevanzbewertung eines angewandten Variablenvorkommen wird die zugehörige Variablenbelegung herangezogen.

Die *Variablenbelegung* für eine XXL–Anfrage setzt sich aus den Belegungen aller vorkommenden Variablen zusammen. Dabei wird jede Variable mit genau einem n–Knoten des XML–Graphen belegt.

Definition 4.18 (Variablenbelegung)
Sei $VAR \subseteq L(G_V)$ die Menge der in einer gegebenen XXL–Anfrage vorkommenden Variablen. Sei $X = (V_X, E_X, \Sigma^*)$ der XML–Graph. Eine Variablenbelegung $\phi : VAR \to V_X$ weist jeder vorkommenden Variablen einen n–Knoten des XML–Graphen zu. □

Sei ϕ eine Variablenbelegung und sei C der Pfadbedingungsgraph für eine Pfadbedingung Q. Ein n–Pfad q ist ein relevanter Pfad für einen s/e–Pfad p aus C und genügt

der Variablenbelegung ϕ, wenn die angewandten Variablenvorkommen gemäß ϕ ausgewertet wurden.

Beispiel 4.14 Erinnern wir uns an die Abbildung 4.7. Der erste Pfadbedingungsgraph enthält eine Variablenbindung für die Variable $\$S$. $\$S$ wird zum Beispiel mit dem letzten Knoten der relevanten Pfades $q = \langle i/ii/iii \rangle$ belegt. Diese Belegung findet dann Verwendung bei der Auswertung des zweiten Pfadbedingungsgraphen. Jeder für den zweiten Pfadbedingungsgraphen relevante Pfad muss der Variablenbelegung für $\$S$ Folge leisten.

4.4.4 Lokaler Relevanzwert von i–Knoten

Wir betrachten hier eine XXL–Suchbedingung, die aus einer Pfadbedingung mit einer Inhaltsbedingung besteht. Sei $C = (V, E, L(G_P), V', V'')$ der Pfadbedingungsgraph und sei $X = (V_X, E_X, \Sigma^*)$ der XML–Graph. Sei q ein relevanter Pfad in X, der vom Pfadbedingungsgraphen C akzeptiert wird. Sei y der letzte n–Knoten in q. Die gegebene Inhaltsbedingung $ib \in \mathcal{I}$ muss also mit dem Element– oder Attributwert verglichen werden, der zu dem n–Knoten $y \in V_X$ gehört.

Hierbei müssen wir auf eine Besonderheit achten (siehe Kapitel 2). Elemente können in ihrem Elementinhalt Subelemente und Elementwerte enthalten. Wenn y ein Attribut repräsentiert, dann steht der zugehörige Wert in einem einzelnen i–Knoten. Wenn y ein Element repräsentiert, dann müssen wir im Hinblick auf gemischtem und geschachteltem Elementinhalt festlegen, was mit der gegebenen Inhaltsbedingung verglichen werden soll.

Im Rahmen dieser Arbeit bezeichnen wir als *Wert* des n–Knotens y entweder den zugehörigen Attributwert bzw. oder die Konkatenation aller nachfolgenden i–Knoten. Die nachfolgenden i–Knoten enthalten gemäß der XML–typischen dokumentinternen Elementhierarchie die Elementwerte von y und die Elementwerte aller Subelemente, aller Subsubelemente, usw. in der richtigen Reihenfolge, was durch die lokale Ordnung des XML–Graphen sichergestellt ist.

Der lokale Relevanzwert eines Wertes drückt den Grad der Übereinstimmung dieses Wertes mit einer gegebenen Inhaltsbedingung aus. Da wir einen oder mehrere i–Knoten mit der gegebenen Inhaltsbedingung vergleichen, weisen wir den lokalen Relevanzwert dem n–Knoten y zu. Dazu verwenden wir eine zweite Funktion $\pi' : \mathcal{I} \times V_X \rightarrow [0, 1]$, die einem n–Knoten des XML–Graphen $X = (V_X, E_X, \Sigma^*)$ in Abhängigkeit von der gegebenen Inhaltsbedingung einen *lokalen Relevanzwert* zwischen 0 und 1 zuweist.

Hierbei müssen wir gemäß der Definition 4.8 eine Vielzahl von Fällen unterscheiden. Unabhängig vom gegebenen Operator müssen wir zunächst beachten, dass entweder die Inhaltsbedingung einen Variablennamen oder einen Stringausdruck enthält.

Fall 1: Besteht die Inhaltsbedingung $ib \in \mathcal{I}$ aus einem binären Operator $o \in \Sigma_{OP}$ und einem Variablennamen $v \in L(G_V)$, dann muss die zugehörige Variablenbelegung $\phi(v) = y' \in V_X$ zum Vergleich herangezogen werden. Für die Art des Vergleichs und die Art der Relevanzbewertung ist letztlich der binäre Operator o maßgebend.

Im Rahmen dieser Arbeit erlauben wir Joinbedingungen nur mit einem binären Operator aus der Menge $\Sigma_{OP} \backslash \{\sim\}$. Der Vergleich führt dann immer zu einem lokalen

Relevanzwert $\pi'(ib, y) = 1.0$, falls der Wert von y mit dem Wert von y' gemäß o übereinstimmt.

Wir erlauben hier keinen ontologiebasierten Ähnlichkeitsvergleich, da a priori nicht klar ist, wie die semantische Ähnlichkeit zweier Texte mit Hilfe einer Ontologie bewertet werden soll.

Fall 2: Besteht die Inhaltsbedingung aus einem binären Operator und einem Stringausdruck (einer aussagenlogischen Formel über Stringkonstanten), dann muss der Wert von y mit dieser Formel verglichen werden. Für die Art des Vergleichs und die Art der Relevanzbewertung ist letztlich der binäre Operator ausschlaggebend. Ist der Operator aus der Menge $\Sigma_{OP}\backslash\{\sim\}$, dann führt der Vergleich entweder zu dem lokalen Relevanzwert $\pi'(ib, y) = 1.0$, falls die Inhaltsbedingung gemäß der geforderten Gleichheit/Ungleichheit/lexikalischen Ähnlichkeit erfüllt ist. Andernfalls ist der lokale Relevanzwert $\pi'(ib, y) = 0$. Ist der binäre Operator das Symbol für die ontologiebasierte semantische Ähnlichkeit, dann wird wie folgt vorgegangen.

1. Die aussagenlogische Formel über atomaren Stringkonstanten (kurz: Keywords) wird in einen Operatorbaum transformiert, dessen innere Knoten die Operatoren und dessen Blätter die Keywords enthalten.
2. Mit Hilfe der Ontologie kann jedes Blatt (jedes Keyword) analog zur ontologiebasierten Relevanzbewertung von n–Knoten disjunktiv expandiert werden.
3. Für jedes Keyword eines Blattes gibt es einen Relevanzwert auf der Basis seines ontologiebasierten Ähnlichkeitswertes und seines statistischen Termgewichts gemäß seiner Häufigkeit im gegebenen Wert.
4. Die Relevanz eines Blattes für einen gegebenen Wert ergibt sich aus der disjunktiven Verknüpfung der stichwortbasierten Relevanzwerte.
5. Die Relevanzbewertung des Wertes des n–Knotens y bzgl. des gegebenen Stringausdrucks ergibt sich aus der Auswertung des Operatorbaums.

Die nächsten Abschnitte befassen sich mit den einzelnen Punkten des dritten Falles etwas genauer. Dazu erläutern wir zunächst im Abschnitt 4.4.4.1 die Gewichtung von Keywords auf der Basis ihrer Vorkommen in einem gegebenen Wert. Anschließend verwenden wir im Abschnitt 4.4.4.2 die Zerlegung eines Stringausdrucks in einen Operatorbaum, um die Relevanzbewertung seiner Blätter sowie die letztendliche Relevanzbewertung des gegebenen Wertes auf der Basis der Auswertung des gesamten Operatorbaums vorzunehmen.

4.4.4.1 TF/IDF–basierte Ähnlichkeit

Das Vektorraummodell (kurz. VRM) ist ein klassisches Information Retrieval Modell [BR99, MS01], um die inhaltliche Ähnlichkeit eines Dokumentes bzgl. einer gegebenen Anfrage zu bewerten. Dazu werden Dokument und Anfrage durch Vektoren von Termen (Stichwörtern) repräsentiert. Den Termen des Dokuments werden Termgewichte zugewiesen, mit deren Hilfe der Grad der Ähnlichkeit zur gegebenen Anfrage berechnet werden kann. Diese Termgewichte können auf verschiedene Art und Weise ermittelt werden. Ein Standardansatz, den wir verwenden werden, basiert auf der Termfrequenz und der inversen Dokumentfrequenz. Die Termfrequenz (tf–Faktor) liefert ein Maß dafür, wie gut ein Term den Dokumentinhalt beschreibt. Die inverse Frequenz eines Terms unter allen Dokumenten der gegebenen Dokumentkollektion (idf–Faktor) soll

berücksichtigen, dass ein Term, der in vielen Dokumenten vorkommt, nicht besonders gut zur Unterscheidung von relevanten und nicht relevanten Dokumenten geeignet ist. Eine möglichst effektive Termgewichtung wird dann erreicht, wenn diese beiden Effekte geeignet balanciert werden.

Sei $D = \{d_1, \ldots, d_n\}$ eine feste Menge von Dokumenten. Das Gewicht eines Terms t_j bzgl. eines Dokumentes d_i ergibt sich aus dem Produkt $w_{ij} = tf_{ij} \cdot idf_j$. Dabei haben die verwendeten Parameter folgende Bedeutung:

w_{ij}	– Gewicht des Terms t_j im Dokument d_i
tf_{ij}	– Termfrequenz (Häufigkeit des Terms t_j im Dokument d_i)
N	– Anzahl der Dokumente insgesamt
df_j	– Dokumentfrequenz (Anzahl der Dokumente, in denen der Term t_j mindestens einmal vorkommt
$idf_j = log\ (N\ /\ df_j)$	– inverse Dokumenthäufigkeit

Im Rahmen dieser Arbeit interessieren uns jedoch die Häufigkeiten von Termen in Element– und Attributwerten, nicht in ganzen (XML–)Dokumenten. Dazu wenden wir obigen Ansatz auf Element– und Attributwerte (kurz: Werte) an, in dem wir diese als Dokumente betrachten, die Dokumentfrequenz *df* als Elementfrequenz *ef* und die inverse Dokumenthäufigkeit *idf* als inverse Elementhäufigkeit *ief* interpretieren.

Zur Vereinfachung sprechen wir von Elementen, meinen aber immer Elemente und Attribute. Sei $\mathcal{E} = \{e_1, \ldots, e_n\}$ eine feste Menge von Elementen. Das Gewicht eines Terms t_j bzgl. des Wertes des n–Knotens y, der für ein konkretes Element steht, ergibt sich aus dem Produkt $w_{ij} = tf_{ij} \cdot ief_j$. Dabei haben die verwendeten Parameter folgende Bedeutung:

w_{ij}	– Gewicht des Terms t_j bzgl. des Wertes des Elements y
tf_{ij}	– Termfrequenz (Häufigkeit des Terms t_j im Wert $e_i.value$
N	– Anzahl der Elemente insgesamt
ef_j	– Elementfrequenz (Anzahl der Elemente, in deren Wert der Term t_j mindestens einmal vorkommt)
$ief_j = log\ (N\ /\ ef_j)$	– inverse Elementhäufigkeit

Die statistischen Informationen über Termfrequenzen, Elementfrequenzen und inverse Elementfrequenzen werden typischer Weise bei der Zerlegung der XML–Dokumente in den XML–Graphen gesammelt und in einem invertierten File gespeichert [BR99]. Dieses Verfahren nimmt postuliert die Unabhängigkeit der Terme voneinander.

Im Rahmen dieser Arbeit geben wir das Gewicht eines Terms t auf der Basis der Termfrequenz und der inversen Elementfrequenz durch die Funktion *score:* $\Sigma^* \times V_X \rightarrow [0, 1]$ an, d.h. $score(t, y) = w_{yt}$.

4.4.4.2 Operatorbaum für Stringausdrücke

Im Abschnitt 4.2.3 haben wir die Syntax von Stringausdrücken mit Hilfe der Grammatik G_S festgelegt. Insbesondere ist ein Stringausdruck eine aussagenlogische Formel über Stringkonstanten, der sich durch einen Operatorbaum gemäß der Definition 4.7 darstellen lässt.

Der Relevanzwert $\pi'(ib, y)$ für den Wert eines n–Knotens y bzgl. einer gegebenen Inhaltsbedingung ib können wir nun mit Hilfe der Ontologie und dem Operatorbaum für den Stringausdruck der Inhaltsbedingung ermitteln. Im Rahmen dieser Arbeit wird dabei jede Stringkonstante eines Blattknotens, die keinen NOT-Knoten als Vorgänger hat, expandiert. Die Stringkonstanten unterhalb von NOT–Knoten bleiben unverändert.

Wir greifen das Beispiel aus Abschnitt 4.2.3 auf und rufen uns den zum gegebenen Stringausdruck zugehörigen Operatorbaum in Erinnerung (siehe Abb. 4.8). Nun wird für jedes Blatt des Operatorbaums ein lokaler Relevanzwert $\pi'(c_i, y)$ ermittelt. Wir müssen hierbei unterscheiden, ob das Blatt mit Hilfe der Ontologie expandiert wurde oder nicht.

Fall 1: Der binäre Operator o ist aus der Menge $\Sigma_{OP}\backslash\{\sim\}$.

Dann bleiben die Blätter des Operatorbaums unverändert und werden gemäß des Operators o ausgewertet. Damit erhält jedes Blatt einen lokalen Relevanzwert $\pi'(c_i, y) = 1.0$, wenn der Vergleich positiv ist. Andernfalls ist $\pi'(c_i, y) = 0$.

Fall 2: Der binäre Operator ist $o =' \sim'$.

In diesem Fall werden alle Knoten, die keinen NOT–Knoten als Vorfahren haben, mit Hilfe der Ontologie expandiert (siehe Abb. 4.8).

Beispiel 4.15 In der Abb. 4.8 geben wir den Operatorbaum passend zum Stringausdruck $''!(c_1|c_2)\&(c_3|c_4)''$ an. Die Stringkonstanten c_3 und c_4 haben keinen NOT–Knoten als Vorgänger und werden dem entsprechend mit Hilfe der Ontologie expandiert.

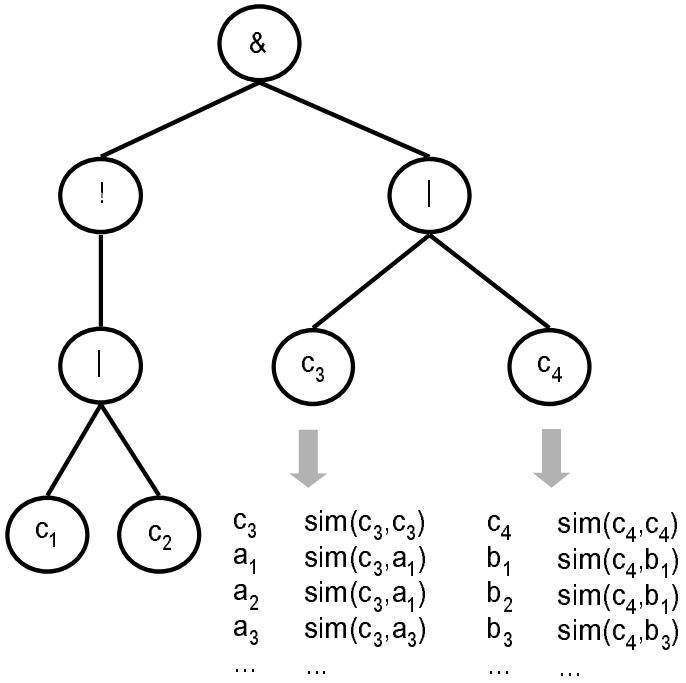


Abbildung 4.8: Operatorbaum für den Stringausdruck "$!(c_1|c_2)\&(c_3|c_4)$"

Der ontologiebasierte Ähnlichkeitswert wird hier zur besseren Illustration zum Ausdruck der semantischen Ähnlichkeit der gegebenen und gefundenen Stringkonstanten angegeben. In Wirklichkeit ergibt sich der Ähnlichkeitswert jeweils auf der Basis der semantischen Ähnlichkeit der zugehörigen Konzepte (siehe Abschnitt 3).

Wurde das Blatt des Operatorbaums expandiert, dann spielen zum einen die ontologiebasierten Ähnlichkeitswerte und die Vorkommen jeder einzelnen Stringkonstante eine Rolle für den lokalen Relevanzwert des Blattes. Wir gehen dabei wie folgt vor. Für die Stringkonstante c_i und jeden semantisch ähnlichen Term t_j wird der lokale Ähnlichkeitswert $\pi'(c_i, y) = sim(c_i, c_i) * score(c_i, y)$ bzw. $\pi'(t_j, y) = sim(c_i, t_j) * score(t_j, y)$

berechnet. Da die Expansion auf der disjunktiven Verknüpfung der Terme beruht, müssen die lokalen Relevanzwerte zur Bestimmung des lokalen Relevanzwertes eines Blattes ebenfalls disjunktiv verknüpft werden. Das muss paarweise rekursiv mit Hilfe der folgenden Rekursion erfolgen:

$$\pi_1 = \pi(r_1)$$
$$\pi_{j+1} = \pi_j + \pi(r_{j+1}) - \pi_j \cdot \pi(r_{j+1})$$

Damit ergibt sich der lokale Relevanzwert $\pi'(c_i|t_1|t_2|t_3|\ldots, y)$ für ein expandiertes Blatt aus den disjunktiv verknüpften lokalen Relevanzwerten für die einzelnen Terme.

Wurde ein Blatt c_j des Operatorbaums nicht expandiert, dann hat es einen NOT–Vorgänger. Damit wird der lokale Relevanzwert $\pi'(c_j, y) = 1.0$ gesetzt, falls die Stringkonstante c_j im Wert von y vorkommt, d.h. $score(c_j, y) > 0$ ist. Andernfalls ist der lokale Relevanzwert $\pi'(c_j, y) = 0$.

Für beide Fälle bleibt nun die Aufgabe, die lokalen Relevanzwerte der Blätter des Operatorbaums zu einem lokalen Relevanzwert $\pi(ib, y)$ für den n–Knoten y bzgl. der ursprünglich gegebenen Inhaltsbedingung zusammenzuführen. Der Relevanzwert $\pi'(ib, y)$ für den Wert des n–Knotens y bzgl. der gegebenen Inhaltsbedingung wird nun mit Hilfe des Operatorbaums und den lokalen Relevanzwerten an den Blättern bottom up (also von unten nach oben) bei den Blättern beginnend berechnet. Dazu werden die Relevanzwerte wie aus der Wahrscheinlichkeitstheorie bekannt wie folgt berechnet:

$$\pi(a\&b) = \pi(a) * \pi(b)$$
$$\pi(a|b) = \pi(a) + \pi(b) - \pi(a) * \pi(b)$$
$$\pi(!a) = 1 - \pi(a)$$

Ist die Wurzel des Operatorbaums erreicht, dann ist der lokale Relevanzwert $\pi'(ib, y)$ für einen gegebenen Wert und den n–Knoten y mit Hilfe oder ohne Hilfe der Ontologie ermittelt worden.

Der Relevanzwert $\pi'(ib, y)$ für den Wert eines n–Knotens y bzgl. einer gegebenen Inhaltsbedingung ib wird entsprechend des gegebenen Vergleichsoperators ermittelt. Im Falle von $\sim$ als Vergleichsoperation werden ontologiebasiert semantisch ähnliche Stichwörter berücksichtigt. Der zugehörige lokale Relevanzwert basiert zum einen auf ontologiebasierten Ähnlichkeitswerten und zum anderen auf tf*ief–basierte Termgewichten.

4.4.5 Resultatgraphen

Sei $X = (V_X, E_X, \Sigma^*)$ der XML–Graph. Seien $W_1, \ldots, W_k$ die k XXL–Suchbedingungen einer gegebenen XXL–Anfrage. Eine XXL–Suchbedingung kann entweder eine Pfadbedingung oder eine Pfadbedingung mit einer Inhaltsbedingung oder eine Pfadbedingung mit einer Variablenbindung enthalten.

1. Enthält eine XXL–Suchbedingung nur eine Pfadbedingung, dann erfüllt ein relevanter Pfad q diese XXL–Suchbedingung mit dem Relevanzwert $rsv(q)$, wobei sich der Relevanzwert aus dem Produkt der lokalen Relevanzwerte der beteiligten n–Knoten ergibt.
2. Enthält eine XXL–Suchbedingung eine Pfadbedingung mit einer Inhaltsbedingung ib, dann erfüllt der relevante Pfad q diese XXL–Suchbedingung mit dem Relevanzwert $rsv(q)$, wobei sich der Relevanzwert für den relevanten Pfad zunächst

aus dem Produkt lokalen Relevanzwerte der beteiligten n–Knoten ergibt, der dann aber noch mit dem Relevanzwert $\pi'(ib, y)$ für den letzten n–Knoten y von q multipliziert wird.

3. Enthält eine XXL–Suchbedingung eine Pfadbedingung mit einer Variablenbindung an die Variable $\$A$, dann erfüllt der relevante Pfad q diese XXL–Suchbedingung mit dem Relevanzwert $rsv(q)$, wobei sich der Relevanzwert aus dem Produkt der lokalen Relevanzwerte der beteiligten n–Knoten ergibt. Die Variable $\$A$ wird mit dem letzten Knoten y von q belegt, also $\phi(\$A) = y$.

Ein Ergebnis für die gegebene XXL–Anfrage besteht aus je einem relevanten Pfad pro XXL–Suchbedingung unter Berücksichtigung der oben zusammengefassten drei Möglichkeiten, die alle einer gemeinsamen Variablenbelegung ϕ genügen.

Definition 4.19 (Resultatgraph)
Sei ϕ eine Variablenbelegung und seien $q_1, \ldots, q_k$ relevante Pfade, wobei jeder n–Pfad q_i vom Pfadbedingungsgraphen C_i akzeptiert wird, ggf. die vorhandene Inhaltsbedingung erfüllt und der Variablenbelegung ϕ genügt.

Ein Resultatgraph $R_\phi = (V_R, E_R, \Sigma^*)$ ist ein Teilgraph des XML–Graphen $X = (V, E, \Sigma^*)$, der sich aus der Vereinigung der Knoten und Kanten der relevanten Pfade $q_1, q_2, \ldots, q_k$ ergibt. Der Relevanzwert $rsv(R_\phi) \in [0, 1]$ des Resultatgraphen setzt sich aus der Multiplikation der lokalen Relevanzwerte π und π' aller beteiligten n–Knoten zusammen. □

Zu Beginn des Abschnitts haben wir einen Überblick über die Definitionen von Syntax und Semantik anhand von Beispielen gegeben. In der Abbildung 4.2 ist ein Resultatgraph basierend auf den relevanten Pfaden für die einzelnen Pfadbedingungsgraphen unter Berücksichtigung einer gegebenen Variablenbelegung grau schattiert.

An dieser Stelle möchten wir die wichtigsten Aspekte dieses Kapitels zusammenfassen. Wir haben hier die Syntax und Semantik der XML–Anfragesprache XXL definiert. Hierbei lassen sich elementare Bedingungen an die Namen und Werte von Elementen und Attributen formulieren. Die strukturbasierte Suche erfolgt durch Pfadbedingungen bestehend aus einzelnen Knotenbedingungen und die inhaltsorientierte Suche erfolgt durch entsprechende Inhaltsbedingungen. Die in Textform gegebenen Pfadbedingungen der XXL–Suchbedingungen einer XXL–Anfrage werden zwecks Auswertung in eine Menge von äquivalenten Pfadbedingungsgraphen transformiert. Als Ergebnis einer XXL–Anfrage werden Mengen von Resultatgraphen bestimmt, die Teilgraphen des XML–Graphen sind. Die Relevanz jedes Ergebnisses beruht auf der lokalen Relevanz der einzelnen Knoten unter Berücksichtigung der ontologiebasierten Ähnlichkeit von einzelnen Knoten, sofern sie eine entsprechende Suchbedingung beinhalten.

Kapitel 5

Anfrageverarbeitung

In den vorherigen Kapiteln haben wir die formalen Grundlagen für die effektive Informationssuche in XML–Dokumenten geschaffen. XML–Dokumente werden durch einen gerichteten XML–Graphen $X = (V_X, E_X, \Sigma^*)$ repräsentiert (Kapitel 2). Mit Hilfe von XXL–Anfragen der Form `SELECT S FROM F WHERE` $\mathtt{W}_1$ `AND ...AND` $\mathtt{W}_k$ $(0 \leq k < \infty)$ kann gezielt nach Informationen in den Struktur– und Inhaltsdaten des XML–Graphen gesucht werden (Abschnitt 4.2: XXL–Syntax). Das Ergebnis einer XXL–Anfrage ist eine absteigend sortierte Rangliste von relevanten Resultatgraphen (Abschnitt 4.4: XXL–Semantik).

In diesem Kapitel befassen wir uns mit dem Problem, wie ein/mehrere/alle relevanten Resultatgraphen zu einer gegebenen XXL–Anfrage berechnet werden können, so dass als Ergebnis eine absteigend sortierte Rangliste von relevanten Resultatgraphen zur Verfügung steht, aus der dann gemäß der Select–Klausel die angestrebte Ergebnismenge extrahiert werden kann.

Dazu betrachten wir zunächst die Teilergebnisse und das Endergebnis einer XXL–Beispielanfrage.

Beispiel 5.1 Die Abbildung 5.1 zeigt einen XML–Graphen und ein Beispiel für eine XXL–Anfrage. Jeder Knoten des XML–Graphen ist mit einem eindeutigen Objektbezeichner (kurz: OID) und dem entsprechenden Namen bzw. Wert eines Elements/Attributes beschriftet. Die gegebene XXL–Anfrage besteht aus drei konjunktiv verknüpften XXL–Suchbedingungen W_1, W_2 und W_3.

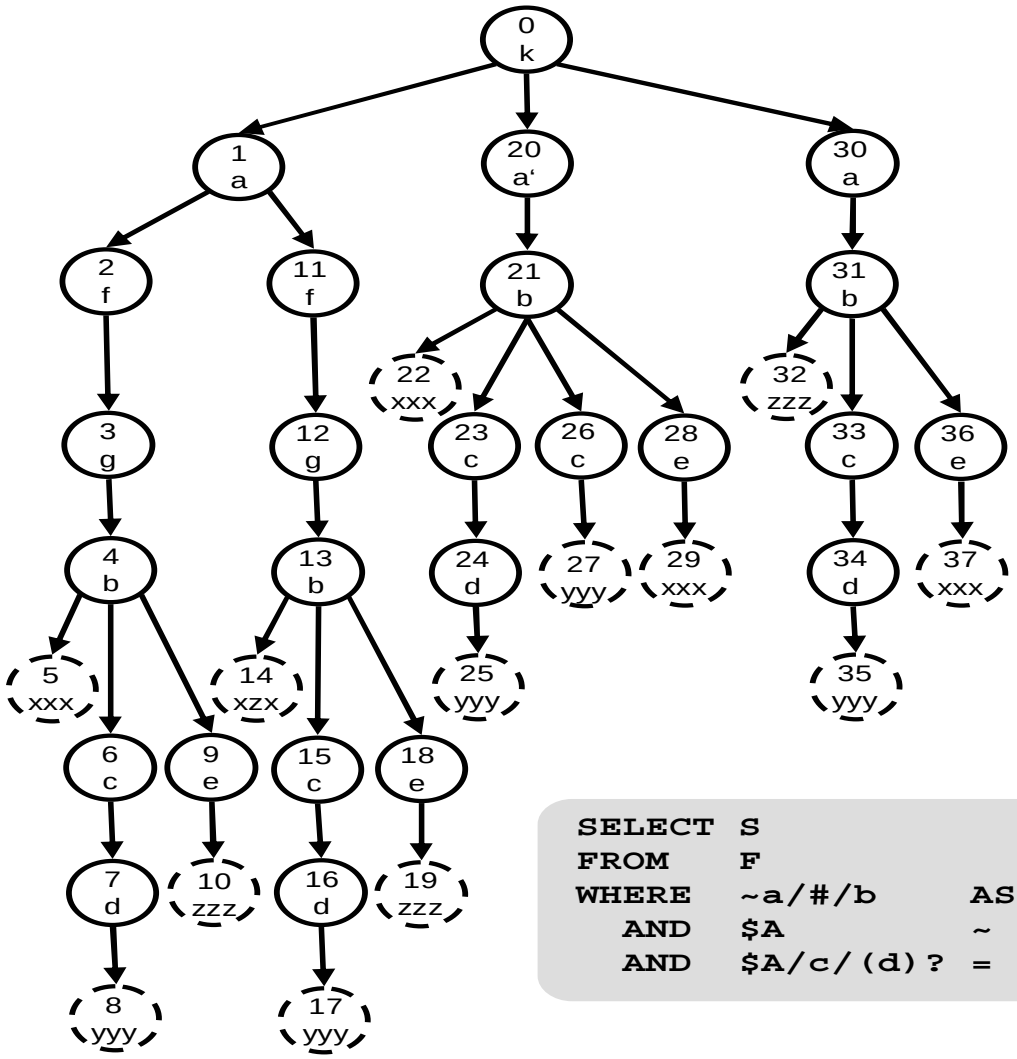


Abbildung 5.1: XML–Beispielgraph und XXL–Beispielanfrage

In den folgenden drei Abbildungen 5.2, 5.3 und 5.4 sind die vollständigen Teilergebnisse der einzelnen XXL–Suchbedingungen schattiert dargestellt. Hierbei besteht ein Teilergebnis definitionsgemäß immer aus einem relevanten Pfad mit einer dazugehörenden Variablenbelegung. Aus Übersichtlichkeitsgründen haben wir auf die Angaben der Relevanzwerte verzichtet.

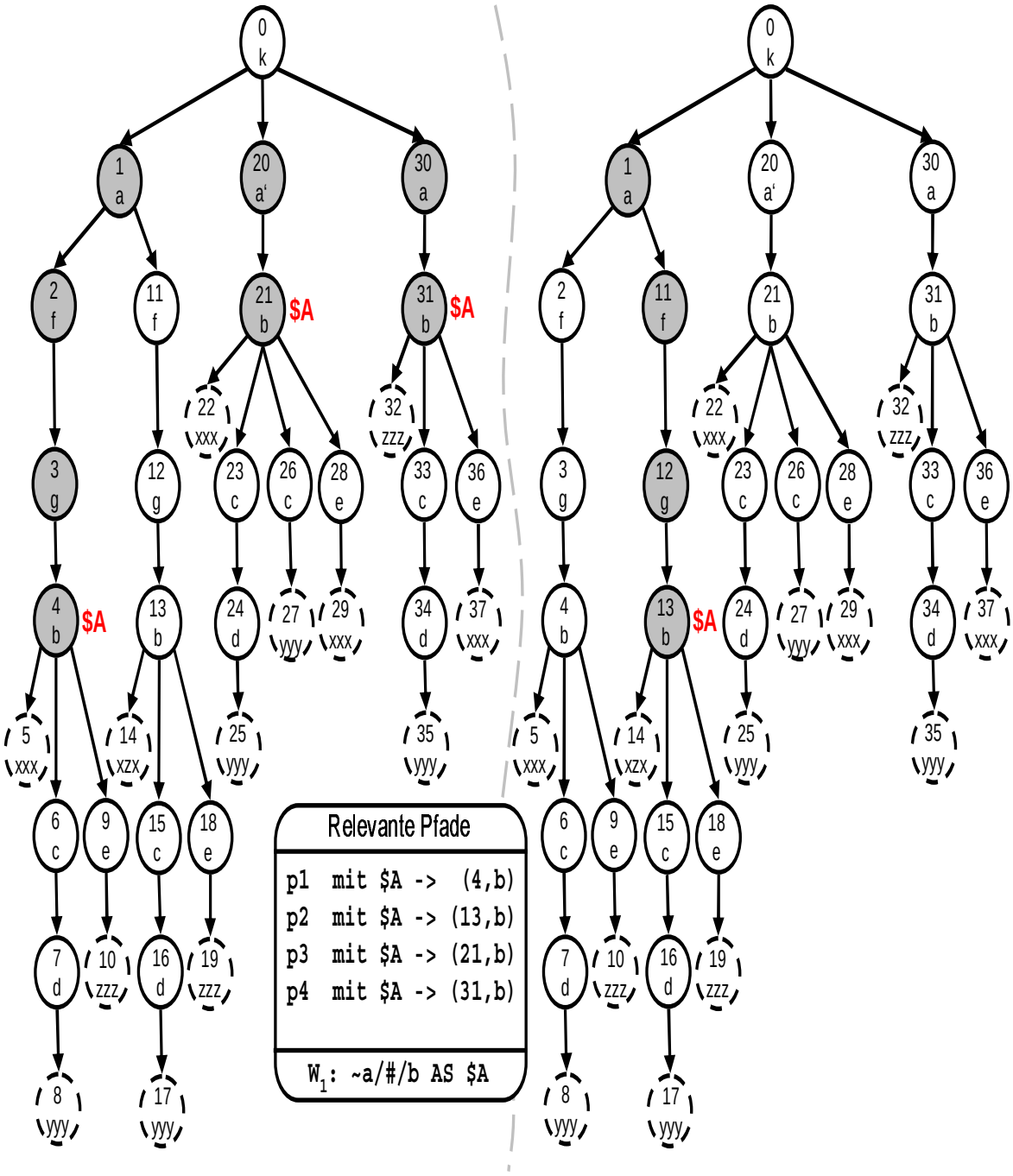


Abbildung 5.2: Relevante Pfade für die XXL–Suchbedingung W_1

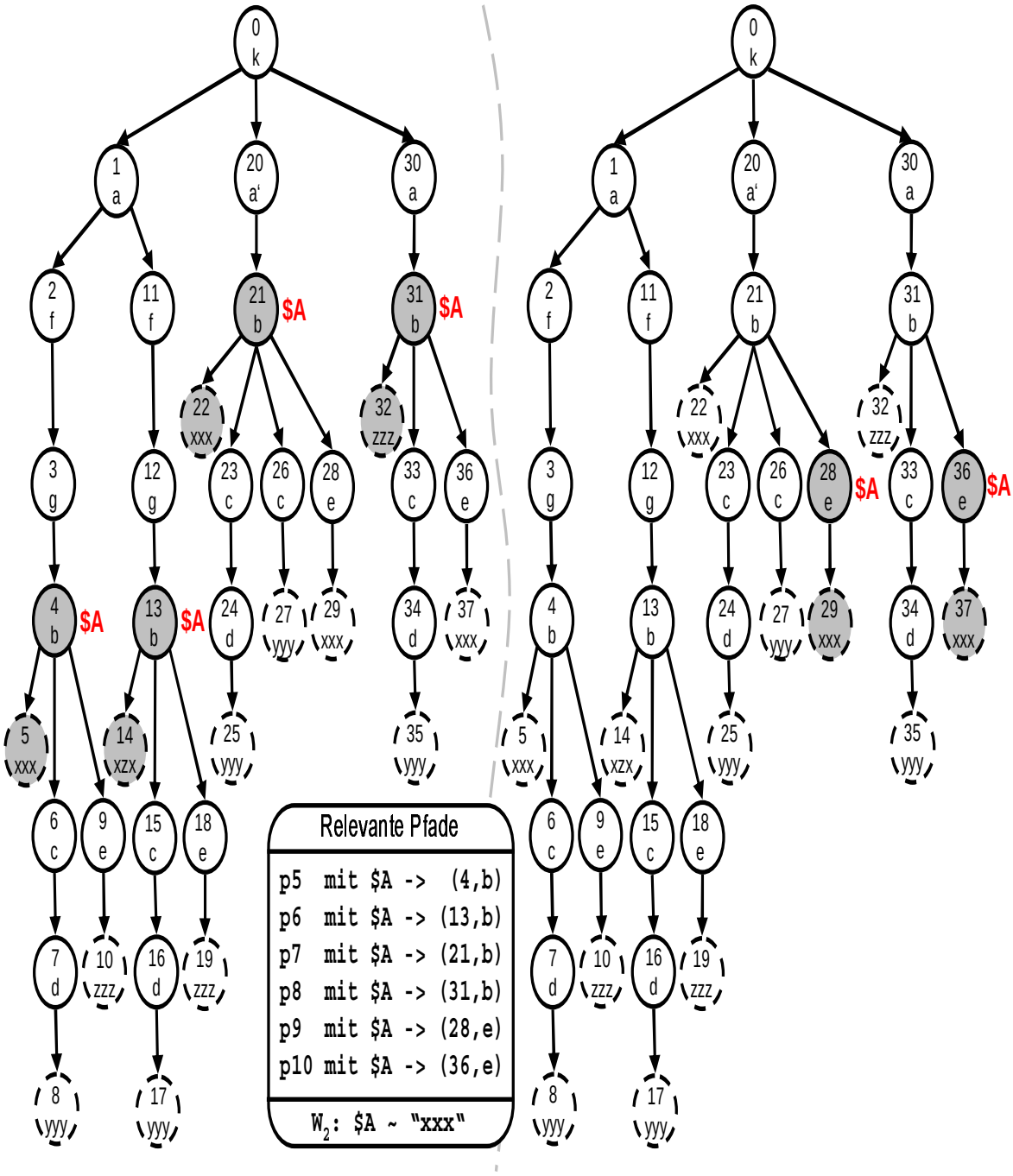


Abbildung 5.3: Relevante Pfade für die XXL–Suchbedingung W_2

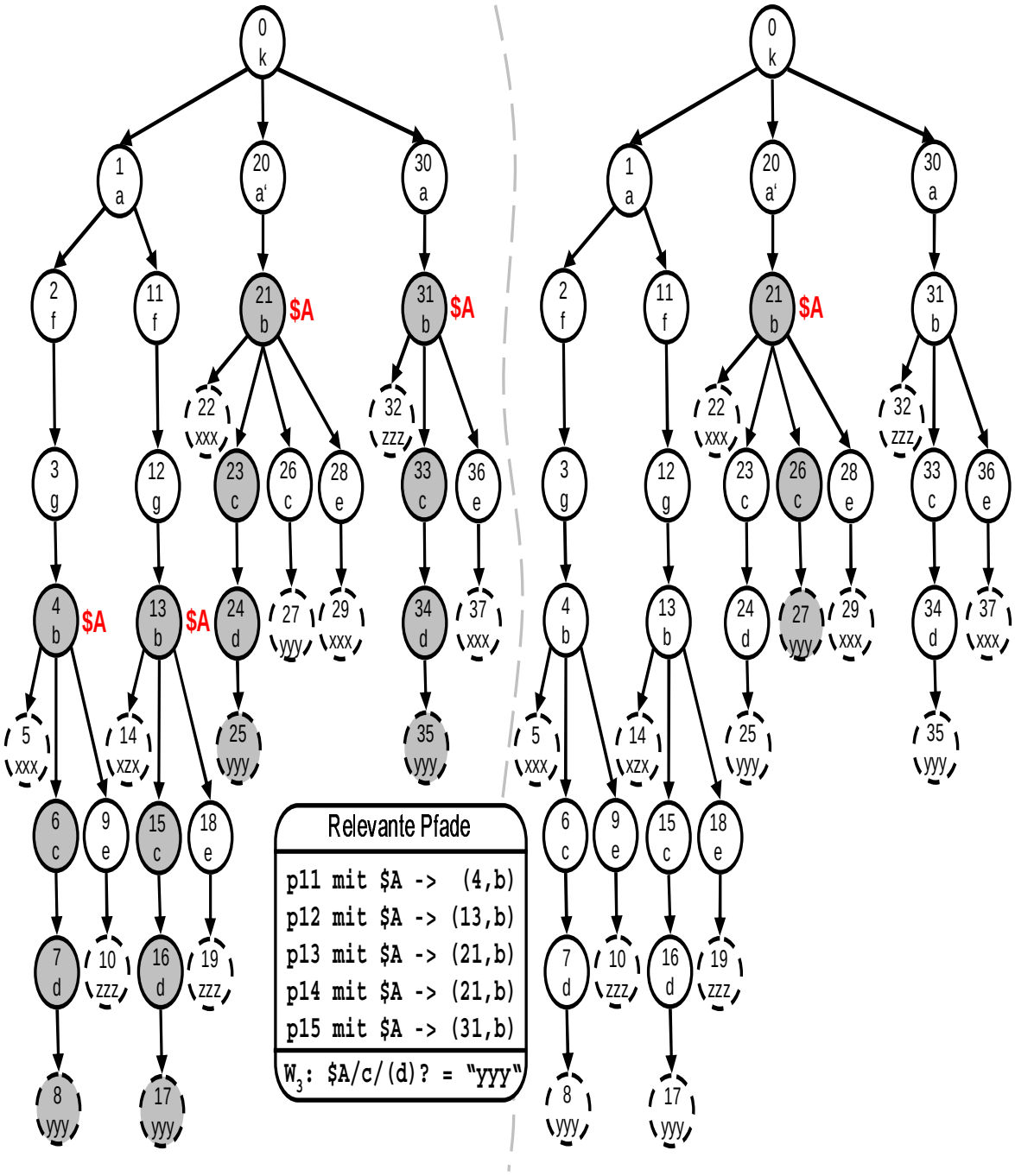


Abbildung 5.4: Relevante Pfade für die dritte XXL–Suchbedingung W_3

Unter Berücksichtigung der drei Teilergebnisse mit den ermittelten Variablenbelegungen besteht das Endergebnis der gegebenen XXL–Beispielanfrage aus fünf verschiedenen Resultatgraphen, die in der Abb. 5.5 dargestellt sind.

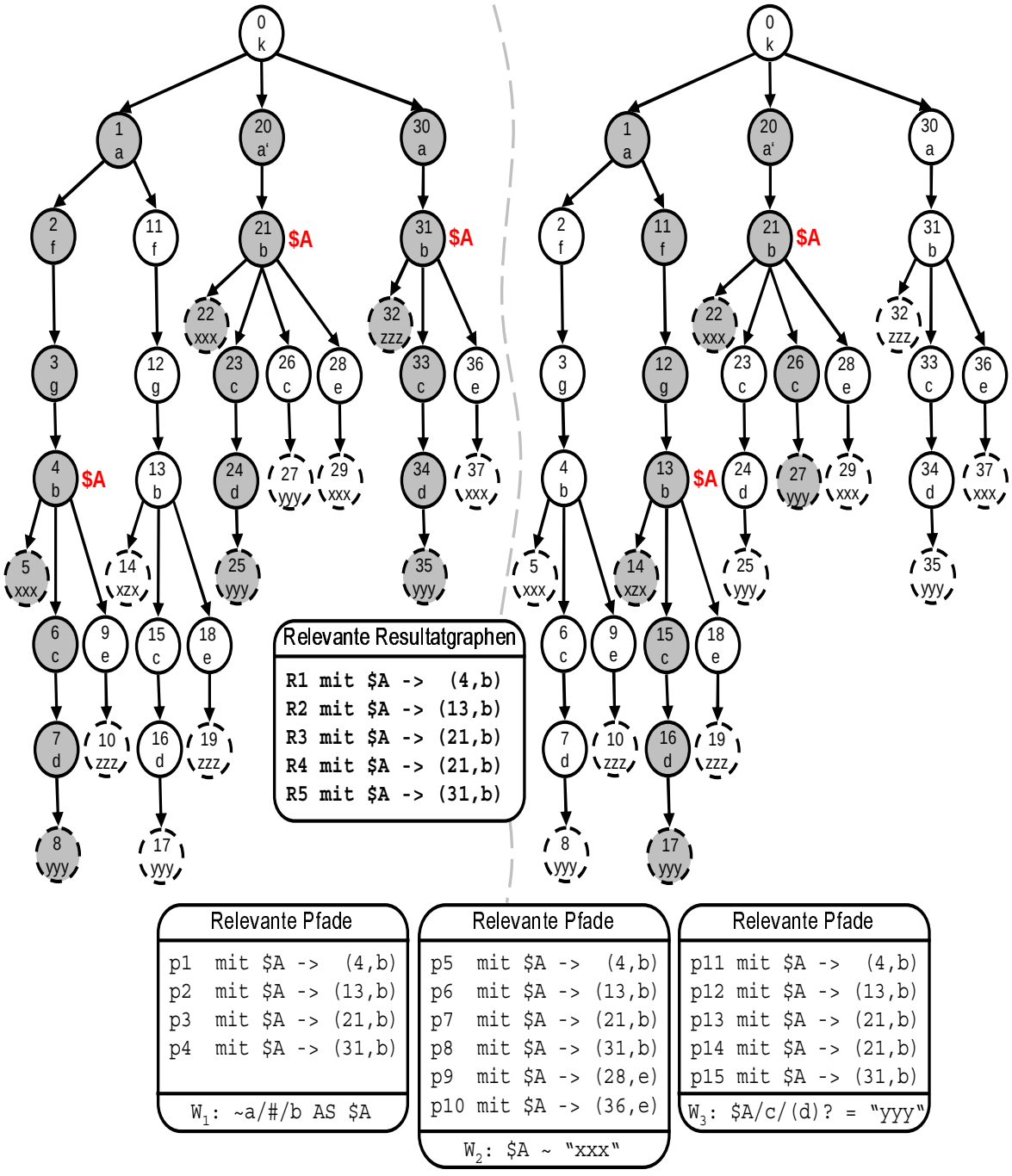


Abbildung 5.5: Relevante Resultatgraphen

Offensichtlich besteht das Problem, einen/mehrere/alle relevanten Resultatgraphen zu einer gegebenen XXL–Anfrage zu berechnen, aus zwei Teilproblemen.

1. Wie werden ein/mehrere/alle relevanten Pfade mit ihren Variablenbelegungen zu einer gegebenen XXL–Suchbedingung berechnet?
2. Wie werden aus den Teilergebnissen der XXL–Suchbedingungen ein/mehrere/alle relevanten Resultatgraphen ermittelt?

Im Abschnitt 5.1 befassen wir uns mit dem ersten Teilproblem und gehen detailliert auf die Verarbeitung einer einzelnen XXL–Suchbedingung ein. Im Abschnitt 5.2 widmen wir uns dem zweiten Teilproblem und erläutern die Verarbeitung der Where–Klausel einer gegebenen XXL–Anfrage, also aller XXL–Suchbedingungen.

5.1 Verarbeitung einer XXL–Suchbedingung

Gegeben seien der XML–Graph $X = (V_X, E_X, \Sigma^*)$ aus Kapitel 2 als Repräsentant einer Menge von beliebigen XML–Dokumenten und eine XXL–Anfragen der Form `SELECT S FROM F WHERE` $\mathtt{W}_1$ `AND ...AND` $\mathtt{W}_k$ $(0 \leq k < \infty)$ aus Kapitel 4. Eine XXL–Suchbedingung $W \in \{W_1, \ldots, W_k\}$ besteht entweder aus einer Pfadbedingung oder einer Pfadbedingung mit einer Inhaltsbedingung oder einer Pfadbedingung mit einer Variablenbindung, wobei die Pfadbedingung durch den zugehörigen Pfadbedingungsgraphen $C = (V, E, L(G_P), V', V'')$ gegeben ist.

Das Ziel der Verarbeitung einer gegebenen XXL–Suchbedingung besteht darin, einen/mehrere/alle relevanten Pfade im XML–Graphen X mit ihren zugehörigen Variablenbelegungen zu finden, die von dem zur XXL–Suchbedingung gehörenden Pfadbedingungsgraphen unter Berücksichtigung bestehender Variablenbindungen und Inhaltsbedingungen akzeptiert werden. Hierbei gilt ein n–Pfad des XML–Graphen als relevant, wenn er relevanzbasiert isomorph zu einem s/e–Pfad (Start/End–Pfad) des gegebenen Pfadbedingungsgraphen unter Berücksichtigung der zusätzlich gegebenen Variablenbindung bzw. Inhaltsbedingung ist (Abschnitt 4.4: XXL–Semantik). D.h. die beiden Pfade stimmen in ihrer Gestalt überein und die n–Knoten des n–Pfades sind lokal relevant.

Im Grundsatz entspricht diese Aufgabe dem Problem, Teilgraphisomorphismen zu finden (subgraph isomorphism detection) bzw. Graphübereinstimmungen zu testen (graph matching) [Har95]. Die Berechnung aller Teilgraphisomorphismen ist insbesondere NP–vollständig.

Im Rahmen dieser Arbeit basiert die Verarbeitung einer XXL–Suchbedingung auf der Verarbeitung des zugehörigen Pfadbedingungsgraphen unter Berücksichtigung der zusätzlich bestehenden Inhaltsbedingungen oder Variablenbindungen. Die Verarbeitung des Pfadbedingungsgraphen beinhaltet die Verarbeitung jedes seiner s/e–Pfade und die Berechnung der zugehörigen relevanten Pfade. Die Verarbeitung eines s/e–Pfades beruht auf der Traversierung dieses Pfades, wobei Schritt für Schritt zu den atomaren Pfadbedingungen der Knoten des s/e–Pfades relevante Knoten im XML–Graphen gesucht und zu einem relevanten Pfad zusammengesetzt werden.

Für die Traversierung des s/e–Pfades gibt es aus der Graphentheorie drei Ansätze, die *Top–Down–Auswertung* vom Start– zum Endknoten, die *Bottom–Up–Auswertung* vom End– zum Startknoten sowie die *hybride Auswertung* als Kombination aus Top–Down– und Bottom–Up–Auswertung. Wir werden uns hier auf die Betrachtung von Top–Down– und Bottom–Up–Auswertung beschränken.

5.1.1 Top–Down–Verarbeitung des Pfadbedingungsgraphen

Die Top–Down–Verarbeitung des Pfadbedingungsgraphen $C = (V, E, L(G_P), V', V'')$ beginnt bei den Startknoten $V' \subseteq V$. In einem Verarbeitungsschritt wird der aktuelle Knoten *act* eines aktuellen s/e–Pfades (Start/End–Pfades) unter Berücksichtigung des vorherigen Knotens *prev* und der bereits produzierten relevanten Pfade $q \in result$

ausgewertet. Neue relevante Knoten werden an die Enden der bisherigen relevanten Pfade angefügt und die Belegung der Variablen wird in *varbind* zentral gespeichert. Der Verarbeitungsschritt wird rekursiv für die Kinder des aktuellen Knotens *act* fortgesetzt, bis ein Endknoten erreicht wird. Nach der Verarbeitung eines Endknotens beinhaltet die Resultatmenge *result* die relevanten Pfade.

Die rekursive Top–Down–Verarbeitung des Pfadbedingungsgraphen ist im folgenden Algorithmus in Pseudocode–Notation dargestellt.

```
processTopDown(result, varbind, prev, act)

   // top–down evaluation of the atomic path condition of node act
(1)  if (act contains a label expression) then
(2)     (result, varbind) = evalLabelTopDown(result, varbind, prev,        // evaluate 'act' by its label
        act)
(3)  if (act contains a variable name) then
(4)     (result, varbind) = evalVarTopDown(result, varbind, prev,          // evaluate 'act' by its var
        act)
(5)  if (act contains a wildcard) then
(6)     (result, varbind) = evalWildcardTopDown(result, varbind,           // evaluate 'act' by its wc
        prev, act)
     // traverse path condition graph downward
(7)  if (act is an end node) then
(8)     foreach (child ∈ Children(act)) do                                 // children of 'act'
(9)        localresult = processTopDown(result, varbind, act, child)       // recursion step
(10)       result.append(localresult)                                      // append new results
(11)    od
(12)    return (result)                                                    // return relevant paths
(13) else
(14)    tmpresult = ∅                                                      // new temp. resultset
(15)    foreach (child ∈ Children(act)) do                                 // children of 'act'
(16)       localresult = processTopDown(result, varbind, act, child)       // recursion step
(17)       tmpresult.append(localresult)                                   // append new results
(18)    od
(19)    return (tmpresult)                                                 // return relevant paths
(20) fi
```

In Abhängigkeit von der Beschriftung des aktuellen Knotens *act* wird in den Zeilen (1)–(6) die entsprechende Top–Down–Auswertung der atomaren Pfadbedingung vorgenommen. Die Zeilen (7)–(12) befassen sich mit dem Fall, dass *act* ein Endknoten ist. In diesem Fall enthält *result* die Menge der relevanten Pfade. Falls *act* Kinder hat, dann ist er zusätzlich ein Nicht–Endknoten auf einem anderen s/e–Pfad und *result* enthält die Anfänge von relevanten Pfaden, die noch zu verlängern und zu ergänzen sind. Die Zeilen (13)–(20) widmen sich dem Fall, dass *act* kein Endknoten ist. Dann enthält *result* die Anfänge von relevanten Pfaden, die mit Hilfe jedes Kindes von *act* verlängert und ergänzt werden können. Alle Kinder werden dabei auf der Basis der aktuellen Ergebnismenge *result* verarbeitet.

In diesem Algorithmus haben wir aus Übersichtlichkeitsgründen auf die Ausformulierung der Abbruchkriterien verzichtet.

Die Top–Down–Auswertung atomarer Pfadbedingungen für einen aktuellen Knoten *act* unter Berücksichtigung des vorherigen Knotens *prev* und der bisher berechneten relevanten Pfade in der Menge *result* beruht auf folgender Idee. Zu jedem relevanten Pfad $q \in result$ werden relevante n–Knoten $y \in V_X$ im XML–Graphen gesucht, also $\pi(y, act) > 0$, die in Abhängigkeit von der Beschriftung des vorherigen Knotens *prev* Kinder bzw. Nachfolger des letzten Knotens $q.last()$ des Pfades q sind. Dazu benötigen wir Methoden, mit denen wir gezielt nach relevanten n–Knoten im XML–Graph suchen können. Für die Top–Down–Auswertung von atomaren Pfadbedingungen (Labelausdrücke, Variablen oder Wildcardzeichen) und Inhaltsbedingungen stehen uns folgende Methoden zur Verfügung.

1. *findNode (oid)*
   Es wird der n–Knoten gesucht, deren eindeutiger Bezeichner (OID) mit der gegebenen *oid* übereinstimmt.
2. *findNodeByName (name)*
   Es werden n–Knoten gesucht, deren Beschriftung mit dem gegebenen Element– bzw. Attributnamen *name* übereinstimmen. Davon kann es mehrere geben.
3. *findChildren (oid)*
   Zu einem durch seinen eindeutigen Bezeichner *oid* gegebenen n–Knoten werden alle Kindknoten gesucht.
4. *findChildrenByName (oid, name)*
   Zu einem durch seinen eindeutigen Bezeichner *oid* gegebenen n–Knoten werden alle Kindknoten gesucht, deren Beschriftung mit dem gegebenen Element– bzw. Attributnamen *name* übereinstimmen.
5. *testEdge (oid1, oid2)*
   Für zwei durch ihre jeweils eindeutigen Bezeichner *oid1, oid2* benannte n–Knoten wird überprüft, ob es von dem ersten zum zweiten eine gerichtete Kante gibt.
6. *findDescendants (oid)*
   Zu einem durch seinen eindeutigen Bezeichner *oid* gegebenen n–Knoten werden alle Nachfolgerknoten gesucht.
7. *findDescendantsByName (oid, name)*
   Zu einem durch seinen eindeutigen Bezeichner *oid* gegebenen n–Knoten werden alle Nachfolgerknoten gesucht, deren Beschriftung mit dem gegebenen Element– bzw. Attributnamen *name* übereinstimmen.
8. *testPath (oid1, oid2)*
   Für zwei durch ihre jeweils eindeutigen Bezeichner *oid1, oid2* benannte n–Knoten wird überprüft, ob es von dem ersten zum zweiten einen Pfad gibt.
9. *findNode (operator, condition)*
   Es werden n–Knoten gesucht, deren Element– bzw. Attributwerte in den zugehörigen i–Knoten bzgl. der gegebenen Inhaltsbedingung relevant sind. Davon kann es mehrere geben.
10. *findNodeByName (name, operator, condition)*
    Es werden n–Knoten gesucht, deren Element– bzw. Attributwerte in den zugehörigen i–Knoten bzgl. der gegebenen Inhaltsbedingung relevant sind und deren Beschriftungen mit dem gegebenen Element– bzw. Attributnamen *name* übereinstimmen. Davon kann es mehrere geben.
11. *testContent (oid, operator, condition)*
    Es wird der Element– bzw. Attributwert eines gegebenen n–Knotens bzgl. einer ebenfalls vorgegebenen Inhaltsbedingung ausgewertet und der Knoten mit dem lokalen Relevanzwert zurückgegeben.

Mit Hilfe der hier genannten Methoden (1)–(4), (6)–(7), (9)–(10) werden entsprechend der Semantik–Definitionen aus Abschnitt 4.4 relevante n–Knoten im XML–Graphen gesucht und zusammen mit ihren lokalen Relevanzwerten zurückgegeben. Die Methoden (5) und (8) werden benötigt, um zu überprüfen, ob zwischen zwei gegebenen n–Knoten eine Kante bzw. eine Verbindung besteht.

Die Top–Down–Auswertung einer atomaren Pfadbedingung beinhaltet die Auswertung der atomaren Pfadbedingung unter Berücksichtigung bestehender Variablenbelegungen. Darüber hinaus werden Inhaltsbedingungen und Variablenbindungen der aktuellen XXL–Suchbedingung verarbeitet. Für die drei Methoden evalLabelTopDown(),

evalVarTopDown() und evalWildcardTopDown() geben wir nun die Algorithmen in Pseudocode–Notation und verwenden darin zur Vereinfachung a, b als Labelausdrücke, $\$A$, $\$B$ als Variablennamen und # als Wildcardzeichen für beliebige Pfade.

Die Auswertung eines Labelausdrucks mit Hilfe der Methode evalLabelTopDown() beruht auf der Überprüfung der elementaren Bedingung an den Element–/Attributnamen. Zusätzlich gibt die vorherige elementare Pfadbedingung an, ob der neue n–Knoten ein Kind oder ein Nachfolger des letzten Knotens des aktuell betrachteten relevanten Pfades ist. Im folgenden Algorithmus sind die Fälle zusammengefasst, die auftreten können.

evalLabelTopDown(*result*, *varbind*, *prev*, *act*)

```
    // find relevant nodes for each given path of the result set
(1)  foreach (q ∈ result) do
    // act has no content condition
(2)    if (prev/act is labeled with null/b) then
(3)      N = findNodesByName(b)
(4)    if (prev/act is labeled with a/b) then
(5)      N = findChildrenByName(q.last(), b)
(6)    if (prev/act is labeled with $A/b) then
(7)      N = findChildrenByName(q.last(), b)
(8)    if (prev/act is labeled with #/b) then
(9)      N = findDescendantsByName(q.last(), b)
    // act is an end node and has some content condition
(10)   if (prev/act is labeled with null/b and act is an end node with condition (op, cond)) then
(11)     N = findNodesByName(b, op, cond)
(12)   if (prev/act is labeled with a/b and act is an end node with condition (op, cond)) then
(13)     N' = findNodesByName(b, op, cond)
(14)     foreach (y ∈ N') do
(15)       if (testEdge(q.last(), y)) then
(16)         N = N ∪ {y}
(17)   if (prev/act is labeled with $A/b and act is an end node with condition (op, cond)) then
(18)     N' = findNodesByName(b, op, cond)
(19)     foreach (y ∈ N') do
(20)       if (testEdge(q.last(), y)) then
(21)         N = N ∪ {y}
(22)   if (prev/act is labeled with #/b and act is an end node with condition (op, cond)) then
(23)     N' = findNodesByName(b, op, cond)
(24)     foreach (y ∈ N') do
(25)       if (testPath(q.last(), y)) then
(26)         N = N ∪ {y}
    // create a new path for the given path and each new node
(27)   result.remove(q)
(28)   foreach (y ∈ N) do
(29)     if (prev is labeled with #) then
(30)       q' := q/#/y
(31)     else
(32)       q' := q/y
(33)     result.insert(q')
    // add variable binding to each path and to global varbind information
(34)   if (act is an end node with variable binding $A)) then
(35)     foreach (q ∈ result) do
(36)       q.varbind(($A, q.last()))
(37)       varbind.add(($A,q.last()))
(38) od
(39)  return (result, varbind)
```

Für die Top–Down–Verlängerung von relevanten Pfaden und die Ergänzung der Ergebnismenge *result* (Zeilen 27–30) müssen alle neuen relevanten Knoten der Mengen N separat berücksichtigt werden. Aus einem relevanten Pfad q und jedem Knoten $y \in N$ wird je ein relevanter Pfad q/y erzeugt und der Menge *result* hinzugefügt. Im Falle einer Variablenbindung wird zum Abschluss (Zeilen 31–34) zu jedem relevanten Pfad $q \in result$ die Variablenbelegung bestehend aus dem Variablennamen und dem letzten Knoten $q.last()$ des relevanten Pfades eingetragen. Diese Variablenbelegungen

werden zusätzlich in der Menge der zentral verfügbaren Variablenbelegungen *varbind* eingetragen.

Mit Hilfe der Methode evalVarTopDown() werden atomare Pfadbedingungen ausgewertet, die aus einem Variablennamen bestehen. Dazu müssen die bisherigen Variablenbelegungen hinsichtlich ihrer aktuellen Relevanz überprüft werden. Der folgende Algorithmus präsentiert die möglichen Fälle.

```
    evalVarTopDown(result, varbind, prev, act)

    // find relevant nodes for each given path of the result set
(1)  foreach (q ∈ result) do
    // act has no content condition
(2)    if (prev/act is labeled with null/$B) then
(3)      N = {y|(x, y) ∈ varbind und x = $B}
(4)    if (prev/act is labeled with a/$B) then
(5)      foreach (y ∈ V_X|(x, y) ∈ varbind und x = $B) do
(6)        if (testEdge(q.last(), y)) then
(7)          N = N ∪ {y}
(8)    if (prev/act is labeled with $A/$B) then
(9)      foreach (y ∈ V_X|(x, y) ∈ varbind und x = $B) do
(10)       if (testEdge(q.last(), y)) then
(11)         N = N ∪ {y}
(12)   if (prev/act is labeled with #/$B) then
(13)     foreach (y ∈ V_X|(x, y) ∈ varbind und x = $B) do
(14)       if (testPath(q.last(), y)) then
(15)         N = N ∪ {y}
    // act is an end node and has some content condition
(16)   if (prev/act is labeled with null/$B and act is an end node with condition (op, cond)) then
(17)     foreach (y ∈ V_X|(x, y) ∈ varbind und x = $B) do
(18)       if (testContent(y, op, cond)) then
(19)         N = N ∪ {y}
(20)   if (prev/act is labeled with a/$B and act is an end node with condition (op, cond)) then
(21)     foreach (y ∈ V_X|(x, y) ∈ varbind und x = $B) do
(22)       if (testContent(y, op, cond) AND testEdge(q.last(), y)) then
(23)         N = N ∪ {y}
(24)   if (prev/act is labeled with $A/$B and act is an end node with condition (op, cond)) then
(25)     foreach (y ∈ V_X|(x, y) ∈ varbind und x = $B) do
(26)       if (testContent(y, op, cond) AND testEdge(q.last(), y)) then
(27)         N = N ∪ {y}
(28)   if (prev/act is labeled with #/$B and act is an end node with condition (op, cond)) then
(29)     foreach (y ∈ V_X|(x, y) ∈ varbind und x = $B) do
(30)       if (testContent(y, op, cond) AND testPath(q.last(), y)) then
(31)         N = N ∪ {y}
    // create a new path for the given path and each new node ...
    // add variable binding to each path and to global varbind information ...
(32)  od
(33)  return (result, varbind)
```

Die Methode evalWildcardTopDown() untersucht primär, ob das Wildcardzeichen # für einen beliebigen Pfad vor einer weiteren atomaren Pfadbedingung steht oder bereits ausgewertet werden kann. Der nachfolgende Algorithmus illustriert die verschiedenen Fälle.

evalWildcardTopDown(*result*, *varbind*, *prev*, *act*)

```
    // find relevant nodes for each given path of the result set
(1) foreach (q ∈ result) do
    // act has no content condition
(2)   if (prev/act is labeled with null/#) then
(3)     //do nothing, wait for the next atomic path condition
(4)   if (prev/act is labeled with a/#) then
(5)     N = findDescendants(q.last())
(6)   if (prev/act is labeled with $A/#) then
(7)     N = findDescendants(q.last())
    // act is an end node and has some content condition
(8)   if (prev/act is labeled with null/# and act is an end node with condition (op, cond)) then
(9)     N = findNodes(op, cond)
(10)  if (prev/act is labeled with a/# and act is an end node with condition (op, cond)) then
(11)    N' = findNodes(op, cond)
(12)    foreach (y ∈ N') do
(13)      if (testPath(q.last(), y)) then
(14)        N = N ∪ {y}
(15)  if (prev/act is labeled with $A/# and act is an end node with condition (op, cond)) then
(16)    N' = findNodes(op, cond)
(17)    foreach (y ∈ N') do
(18)      if (testPath(q.last(), y)) then
(19)        N = N ∪ {y}
    // create a new path for the given path and each new node ...
    // add variable binding to each path and to global varbind information ...
(20) od
(21) return (result, varbind)
```

Beispiel 5.2 An dieser Stelle demonstrieren wir nun die Top–Down–Verarbeitung einer XXL–Suchbedingung anhand der ersten XXL–Suchbedingung " `a/#/b AS $A`" der aktuellen XXL–Beispielanfrage. Dazu betrachten wir den XML–Graphen aus Abb. 5.1. In diesem Beispiel geben wir einen relevanten n–Knoten durch seinen eindeutigen Bezeichner (z.B. `1`), seinen Namen (z.B. `a`) und seinen lokalen Relevanzwert (z.B. `0.75`) in der Form `(1,a)[0.75]` an und verwenden '/' zur Konkatenation solcher Knoten zu relevanten Pfaden. Die Menge *result* enthält zu jedem relevanten Pfad die zugehörige Variablenbelegung.

```
result = {}, varbind = {}
// beginne Top-Down-Verarbeitung bei einem Startknoten
processTopDown(result, null, ~a)
  evalLabelTopDown(result, null, ~a)
    // werte '~a' aus
    findNodeByName(~a)
    // fuege Ergebnisse ein (relevante Pfade der Laenge 1)
    result = {((1,a)[1.0], {}),
              ((20,a')[0.9], {}),
              ((30,a)[1.0], {})}
    // setzte Top-Down-Verarbeitung bei den Kindern fort
    processTopDown(result, ~a, #)
      evalWildcardTopDown(result, ~a, #)
        // werte '~a/#' aus, d.h. tue hier nichts
        // setzte Top-Down-Verarbeitung bei den Kindern fort
        processTopDown(result, #, b)
          evalLabelTopDown(result, #, b)
```

```
            // werte '#/b' aus
            findDescendantsByName(q.last(), b)
              // fuege Ergebnisse ein
              result = {((1,a)[1.0]/#[1.0]/(4,b)[1.0], {}),
                        ((1,a)[1.0]/#[1.0]/(13,b)[1.0], {}
                        ((20,a')[0.9]/#[1.0]/(21,b)[1.0], {}),
                        ((30,a)[1.0]/#[1.0]/(31,b)[1.0], {})}
              // trage Variablenbelegungen fuer die Variablenbindung ein
              result = {((1,a)[1.0]/#[1.0]/(4,b)[1.0], {($A -> (4,b))}),
                        ((1,a)[1.0]/#[1.0]/(13,b)[1.0], {($A -> (13,b))}
                        ((20,a')[0.9]/#[1.0]/(21,b)[1.0], {($A -> (21,b))}),
                        ((30,a)[1.0]/#[1.0]/(31,b)[1.0], {($A -> (31,b))})}
              varbind = {($A -> (4,b),(13,b),(21,b),(31,b))}

RETURN (result, varbind)
```

Die Verarbeitung der ersten XXL–Suchbedingung liefert die erwarteten vier relevanten Pfade (vergleiche Abb. 5.2).

5.1.2 Bottom–Up–Verarbeitung des Pfadbedingungsgraphen

Die Bottom–Up–Verarbeitung eines Pfadbedingungsgraphen verläuft ähnlich zur Top–Down–Verarbeitung. Die wesentlichen Unterschiede ergeben sich aus folgenden Aspekten.

- Die Bottom–Up–Verarbeitung des Pfadbedingungsgraphen $C = (V, E, L(G_P), V', V'')$ beginnt bei den Endknoten und traversiert den Graphen entgegen der Kantenrichtung von einem Knoten zu seinen Eltern bis zu den Startknoten.
- Zur gezielten Suche von relevanten n–Knoten im XML–Graphen werden Methoden wie findParents() benötigt.
- Die Auswertung atomarer Pfadbedingungen verlängert und ergänzt relevante Pfade an deren Anfang, wobei hier mit dem aktuellen Knoten *act* und dem vorherigen Knoten *prev* (act/prev) gearbeitet wird.

Die rekursive Bottom–Up–Verarbeitung des Pfadbedingungsgraphen ist im folgenden Algorithmus in Pseudocode–Notation dargestellt.

```
processBottomUp(result, varbind, prev, act)

    // bottom up evaluation of the atomic path condition of node act
(1) if (act contains a label expression) then
(2)    (result, varbind) = evalLabelBottomUp(result, varbind, prev, act)      // evaluate 'act' by its label
(3) if (act contains a variable name) then
(4)    (result, varbind) = evalVarBottomUp(result, varbind, prev, act)        // evaluate 'act' by its var
(5) if (act contains a wildcard) then
(6)    (result, varbind) = evalWildcardBottomUp(result, varbind, prev, act)   // evaluate 'act' by its wc
    // traverse path condition graph upward
(7) if (act is a start node) then
(8)    foreach (parent ∈ Parents(act)) do                                     // parents of 'act'
(9)       localresult = processBottomUp(result, varbind, act, child)          // recursion step
(10)      result.append(localresult)                                          // append new results
(11)   od
(12)    return (result)                                                       // return relevant paths
(13) else
(14)   tmpresult = ∅                                                         // new temp. resultset
(15)   foreach (parent ∈ Parents(act)) do                                    // parents of 'act'
(16)      localresult = processBottomUp(result, varbind, act, child)         // recursion step
(17)      tmpresult.append(localresult)                                      // append new results
(18)   od
(19)    return (tmpresult)                                                   // return relevant paths
(20) fi
```

Zur gezielten Suche von n–Knoten im XML–Graphen benötigen wir zur Bottom–Up–Verarbeitung eines Pfadbedingungsgraphen die folgenden Methoden. Einen Teil der Methoden können wir unverändert aus dem Abschnitt über die Top–Down–Verarbeitung übernehmen (daher die Numerierung).

1. *findNode (oid)*
   Es wird der n–Knoten gesucht, deren OID mit der gegebenen *oid* übereinstimmt.

2. *findNodeByName (name)*
   Es werden n–Knoten gesucht, deren Beschriftung mit dem gegebenen Element– bzw. Attributnamen *name* übereinstimmen. Davon kann es mehrere geben.

12. *findParents (oid)* [Bottom–Up–Auswertung]
    Zu einem durch seinen eindeutigen Bezeichner *oid* gegebenen n–Knoten werden alle Elternknoten gesucht.

13. *findParentsByName (oid, name)* [Bottom–Up–Auswertung]
    Zu einem durch seinen eindeutigen Bezeichner *oid* gegebenen n–Knoten werden alle Elternknoten gesucht, deren Beschriftung mit dem gegebenen Element– bzw. Attributnamen *name* übereinstimmen.

5. *testEdge (oid1, oid2)*
   Für zwei durch ihre jeweils eindeutigen Bezeichner *oid1, oid2* benannte n–Knoten wird überprüft, ob es von dem ersten zum zweiten eine gerichtete Kante gibt.

14. *findAncestors(oid)* [Bottom–Up–Auswertung]
    Zu einem durch seinen eindeutigen Bezeichner *oid* gegebenen n–Knoten werden alle Vorgängerknoten gesucht.

15. *findAncestorsByName (oid, name)* [Bottom–Up–Auswertung]
    Zu einem durch seinen eindeutigen Bezeichner *oid* gegebenen n–Knoten werden alle Vorgängerknoten gesucht, deren Beschriftung mit dem gegebenen Element– bzw. Attributnamen *name* übereinstimmen.

8. *testPath (oid1, oid2)*
   Für zwei durch ihre jeweils eindeutigen Bezeichner *oid1, oid2* benannte n–Knoten wird überprüft, ob es von dem ersten zum zweiten einen Pfad gibt.

9. *findNode (operator, condition)*
   Es werden n–Knoten gesucht, deren Element– bzw. Attributwerte in den zugehörigen i–Knoten bzgl. der gegebenen Inhaltsbedingung relevant sind. Davon kann es mehrere geben.

10. *findNodeByName (name, operator, condition)*
    Es werden n–Knoten gesucht, deren Element– bzw. Attributwerte in den zugehörigen i–Knoten bzgl. der gegebenen Inhaltsbedingung relevant sind und deren Beschriftungen mit dem gegebenen Element– bzw. Attributnamen *name* übereinstimmen. Davon kann es mehrere geben.

11. *testContent (oid, operator, condition)*
    Es wird der Element– bzw. Attributwert eines gegebenen n–Knotens bzgl. einer ebenfalls vorgegebenen Inhaltsbedingung ausgewertet und der Knoten mit dem lokalen Relevanzwert zurückgegeben.

Mit Hilfe der hier genannten Methoden (1)–(4), (6)–(7), (9)–(10) werden entsprechend der Semantik–Definitionen aus Abschnitt 4.4 relevante n–Knoten im XML–Graphen gesucht und zusammen mit ihren lokalen Relevanzwerten zurückgegeben.

Die Bottom–Up–Auswertung einer atomaren Pfadbedingung beinhaltet die Auswertung der atomaren Pfadbedingung unter Berücksichtigung bestehender Variablenbelegungen. Darüber hinaus werden Inhaltsbedingungen und Variablenbindungen der aktuellen XXL–Suchbedingung verarbeitet. Für die drei Methoden evalLabelBottomUp(), evalVarBottomUp() und evalWildcardBottomUp() geben wir nun die Algorithmen in Pseudocode–Notation und verwenden darin zur Vereinfachung a, b als Labelausdrücke, $\$A$, $\$B$ als Variablennamen und # als Wildcardzeichen für beliebige Pfade.

Die Auswertung eines Labelausdrucks mit Hilfe der Methode evalLabelBottomUp() beruht auf der Überprüfung der elementaren Bedingung an den Element–/Attributnamen. Zusätzlich gibt die vorherige elementare Pfadbedingung an, ob der neue n–Knoten ein Elternknoten oder ein Vorgänger des ersten Knotens des aktuell betrachteten relevanten Pfades ist. Im folgenden Algorithmus sind die Fälle zusammengefasst, die auftreten können.

```
    evalLabelBottomUp(result, varbind, prev, act)

    // find relevant nodes for each given path of the result set
(1)  foreach (q ∈ result) do
    // act has no content condition
(2)    if (act/prev is labeled with b/null) then
(3)      N = findNodesByName(b)
(4)    if (act/prev is labeled with b/a) then
(5)      N = findParentsByName(q.last(), b)
(6)    if (act/prev is labeled with b/$A) then
(7)      N = findParentsByName(q.last(), b)
(8)    if (act/prev is labeled with b/#) then
(9)      N = findAncestorsByName(q.last(), b)
    // act is an end node and has some content condition
(10)   N = findNodesByName(b, op, cond)
    // create a new path for the given path and each new node
(11)   result.remove(q)
(12)   foreach (y ∈ N) do
(13)     if (prev is labeled with #) then
(14)       q' := y/#/q
(15)     else
(16)       q' := y/q
(17)     result.insert(q')
    // add variable binding to each path and to global varbind information
(18)   if (act is an end node with variable binding $A)) then
(19)     foreach (q ∈ result) do
(20)       q.varbind(($A, q.last()))
(21)       varbind.add(($A,q.last()))
(22)  od
(23)  return (result, varbind)
```

Für die Bottom–Up–Verlängerung von relevanten Pfaden und die Ergänzung der Ergebnismenge *result* (Zeilen 11–17) müssen alle neuen relevanten Knoten der Mengen N separat berücksichtig werden. Aus einem relevanten Pfad q und jedem Knoten $y \in N$ wird je ein relevanter Pfad y/q bzw. $y/\#/q$ erzeugt und der Menge *result* hinzugefügt. Im Falle einer Variablenbindung wird zum Abschluss (Zeilen 18–21) zu jedem relevanten Pfad $q \in result$ die Variablenbelegung bestehend aus dem Variablennamen und dem letzten Knoten $q.last()$ des relevanten Pfades eingetragen. Diese Variablenbelegungen werden zusätzlich in der Menge der zentral verfügbaren Variablenbelegungen *varbind* eingetragen.

Mit Hilfe der Methode evalVarBottomUp() werden atomare Pfadbedingungen ausgewertet, die aus einem Variablennamen bestehen. Dazu müssen die bisherigen Variablenbelegungen hinsichtlich ihrer aktuellen Relevanz überprüft werden. Der folgende Algorithmus präsentiert die möglichen Fälle.

```
evalVarBottomUp(result, varbind, prev, act)

    // find relevant nodes for each given path of the result set
(1)  foreach (q ∈ result) do
    // act has no content condition
(2)    if (act/prev is labeled with $B/null) then
(3)      N = {y|(x, y) ∈ varbind und x = $B}
(4)    if (act/prev is labeled with $B/a) then
(5)      foreach (y ∈ V_X|(x, y) ∈ varbind und x = $B) do
(6)        if (testEdge(y, q.first())) then
(7)          N = N ∪ {y}
(8)    if (act/prev is labeled with $B/$A) then
(9)      foreach (y ∈ V_X|(x, y) ∈ varbind und x = $B) do
(10)       if (testEdge(y, q.first())) then
(11)         N = N ∪ {y}
(12)   if (act/prev is labeled with $B/#) then
(13)     foreach (y ∈ V_X|(x, y) ∈ varbind und x = $B) do
(14)       if (testPath(y, q.first())) then
(15)         N = N ∪ {y}
    // act is an end node and has some content condition
(16)   foreach (y ∈ V_X|(x, y) ∈ varbind und x = $B) do
(17)     if (testContent(y, op, cond)) then
(18)       N = N ∪ {y}
    // create a new path for the given path and each new node ...
    // add variable binding to each path and to global varbind information ...
(19) od
(20)  return (result, varbind)
```

Die Methode evalWildcardBottomUp() untersucht primär, ob das Wildcardzeichen # für einen beliebigen Pfad vor einer weiteren atomaren Pfadbedingung steht oder bereits ausgewertet werden kann. Der nachfolgende Algorithmus illustriert die verschiedenen Fälle.

```
evalWildcardBottomUp(result, varbind, prev, act)

    // find relevant nodes for each given path of the result set
(1)  foreach (q ∈ result) do
    // act has no content condition
(2)    if (act/prev is labeled with #/null) then
(3)      //do nothing, wait for the next atomic path condition
(4)    if (act/prev is labeled with #/a) then
(5)      N = findAncestors(q.first())
(6)    if (act/prev is labeled with #/$A) then
(7)      N = findAncestors(q.first())
    // act is an end node and has some content condition
(8)    N = findNodes(op, cond)
    // create a new path for the given path and each new node ...
    // add variable binding to each path and to global varbind information ...
(9)  od
(10)  return (result, varbind)
```

Beispiel 5.3 An dieser Stelle demonstrieren wir nun die Bottom–Up–Verarbeitung einer XXL–Suchbedingung anhand der ersten XXL–Suchbedingung "`~a/#/b AS $A`" der aktuellen XXL–Beispielanfrage. Dazu betrachten wir den XML–Graphen aus Abb. 5.1. In diesem Beispiel geben wir einen relevanten n–Knoten durch seinen eindeutigen

Bezeichner (z.B. `1`), seinen Namen (z.B. `a`) und seinen lokalen Relevanzwert (z.B. `0.75`) in der Form `(1,a)[0.75]` an und verwenden '/' zur Konkatenation solcher Knoten zu relevanten Pfaden. Die Menge *result* enthält zu jedem relevanten Pfad die zugehörige Variablenbelegung.

```
result = {}, varbind = {}
// beginne Bottom-Up-Verarbeitung bei einem Endknoten
processBottomUp(result, null, b)
  evalLabelBottomUp(result, null, b)
    // werte 'b' aus
    findNodeByName(b)
    // fuege Ergebnisse ein (relevante Pfade der Laenge 1)
    result = {((4,b)[1.0], {}),
              ((13,b)[1.0], {}),
              ((21,b)[1.0], {}),
              ((31,b)[1.0], {})}
    // setzte Bottom-Up-Verarbeitung bei den Eltern fort
    processBottomUp(result, b, #)
      evalWildcardBottomUp(result, b, #)
        // werte '#/b' aus, d.h. tue hier nichts
        // setzte Bottom-Up-Verarbeitung bei den Eltern fort
        processBottomUp(result, #, ~a)
          evalLabelTopDown(result, #, ~a)
            // werte '~a/#' aus
            findAncestorsByName(q.first(), ~a)
              // fuege Ergebnisse ein
              result = {((1,a)[1.0]/#[1.0]/(4,b)[1.0], {}),
                        ((1,a)[1.0]/#[1.0]/(13,b)[1.0], {}
                        ((20,a')[0.9]/#[1.0]/(21,b)[1.0], {}),
                        ((30,a)[1.0]/#[1.0]/(31,b)[1.0], {})}
              // trage Variablenbelegungen fuer die Variablenbindung ein
              result = {((1,a)[1.0]/#[1.0]/(4,b)[1.0], {($A -> (4,b))}),
                        ((1,a)[1.0]/#[1.0]/(13,b)[1.0], {($A -> (13,b))}
                        ((20,a')[0.9]/#[1.0]/(21,b)[1.0], {($A -> (21,b))}),
                        ((30,a)[1.0]/#[1.0]/(31,b)[1.0], {($A -> (31,b))})}
              varbind = {($A -> (4,b),(13,b),(21,b),(31,b))}

RETURN (result, varbind)
```

Die Verarbeitung der ersten XXL–Suchbedingung liefert die erwarteten vier relevanten Pfade (vergleiche Abb. 5.2).

5.2 Verarbeitung einer Where–Klausel

Eine XXL–Anfrage der Form `SELECT S FROM F WHERE W1 AND ...AND Wk` ($0 \leq k < \infty$) enthält in der Where–Klausel k konjunktiv verknüpfte XXL–Suchbedingungen. Die Verarbeitung der Where–Klausel umfasst folgende Schritte:

1. Zerlegung der Where–Klausel in die XXL–Suchbedingungen $W_1, \ldots, W_k$
2. Festlegung der globalen Auswertungsreihenfolge für die Verarbeitung der k XXL–Suchbedingungen
3. Festlegung der lokalen Auswertungsstrategie für die Verarbeitung jeder XXL–Suchbedingung

4. Verarbeitung der k XXL–Suchbedingungen gemäß der globalen Auswertungsreihenfolge und Zusammensetzung der Teilergebnisse zu relevanten Resultatgraphen

Wir betrachten nun die oben genannten vier Schritte etwas genauer.

1. Zerlegung der Where–Klausel

Die XXL–Syntaxdefinition aus Abschnitt 4.2 legt fest, dass das reservierte Wort 'AND' nur zur konjunktiven Verknüpfung von XXL–Suchbedingungen verwendet wird. Die Zerlegung der Where–Klausel in die k XXL–Suchbedingungen auf der Basis des reservierten Wortes 'AND' ist ohne weiteres möglich. Eine XXL–Suchbedingung besteht dabei entweder aus einer Pfadbedingung oder einer Pfadbedingung und einer Inhaltsbedingung oder einer Pfadbedingung mit einer Variablenbindung. Entsprechend der Vorgaben in Abschnitt 4.3 wird zusätzlich für jede XXL–Suchbedingung zu jeder Pfadbedingung der zugehörige Pfadbedingungsgraph erzeugt.

2. Globale Auswertungsreihenfolge

Die *globale Auswertungsreihenfolge* legt fest, in welcher Reihenfolge die gegebenen XXL–Suchbedingungen abgearbeitet werden. Dazu gibt es prinzipiell zwei Möglichkeiten, entweder die XXL–Suchbedingungen werden unabhängig voneinander oder abhängig voneinander ausgewertet.

Im ersten Fall spielt die Reihenfolge der Verarbeitung der k XXL–Suchbedingungen keine Rolle. Wurden für alle XXL–Suchbedingungen alle Teilergebnisse bestehend aus relevanten Pfaden und den zugehörigen Variablenbelegungen ermittelt, können auf der Basis gemeinsamer Variablenbelegungen die relevanten Resultatgraphen zusammengestellt werden (siehe Beispiel zu Beginn dieses Kapitels). Der Hauptvorteil dieses Ansatzes beruht auf der Möglichkeit, die Verarbeitung der k XXL–Suchbedingungen parallel durchzuführen. Der Hauptnachteil dieser Methode besteht in der Größe der Teilergebnisse, die aufgrund ihrer Unabhängigkeit tendentiell wesentlich mehr potentiell relevante Treffer bereithalten, als für das Endergebnis tatsächlich relevant sind.

Im zweiten Fall werden die XXL–Suchbedingungen nacheinander ausgewertet. Im Rahmen dieser Arbeit verlangen wir, dass diese Reihenfolge den Variablenabhängigkeitsgraph aus Definition 4.12 berücksichtigt, so dass jede XXL–Suchbedingung zum Zeitpunkt ihrer Verarbeitung vollständig ausgewertet werden kann, ohne auf Variablenbelegungen warten zu müssen. Idealer Weise sollte diese Reihenfolge so gewählt sein, dass die Verarbeitung der Where–Klausel effizient mit kleinen Teilergebnissen möglich ist. Hierzu gibt es eine Vielzahl von Heuristiken auf der Basis der Syntax der XXL–Suchbedingungen und den statistischen Informationen über Häufigkeiten von Element– und Attributnamen und über Häufigkeiten von Stichwörtern in Element– und Attributwerten im XML–Graphen, die man zu dieser Entscheidungsfindung heranziehen kann.

Auf der Basis von Selektivitätsschätzungen zu den einzelnen XXL–Suchbedingungen kann eine solche Reihenfolge festgelegt werden. Im Rahmen dieser Arbeit schätzen wir die Selektivität einer XXL–Suchbedingung mit dem Maximum der Selektivitäten der atomaren Pfadbedingungen und der Inhaltsbedingung ab. Die XXL-Suchbedingungen werden gemäß iher Selektivitäten aufsteigend sortiert. Die globale Auswertungsreihenfolge wird dann auf der Basis dieser Sortierung unter Berücksichtigung des Variablenabhängigkeitsgraphen festgelegt.

3. Lokale Auswertungsstrategie

Die *lokale Auswertungsstrategie* legt die Art und Weise fest, in der die betrachtete XXL–Suchbedingung verarbeitet wird. Gemäß des vorherigen Abschnitts 5.1 können XXL–Suchbedingungen top–down oder bottom–up verarbeitet werden. Hierzu gibt es

eine Vielzahl von Heuristiken, die die Strategie auf der Basis der XXL–Syntax und den Daten im XML–Graphen auswählen.

Im Rahmen dieser Arbeit schätzen wir die Selektivitäten der Start– und der Endknoten und legen darauf hin die lokale Auswertungsstrategie fest.

4. Verarbeitung der k XXL–Suchbedingungen

Für diesen Schritt nehmen wir an, dass die k XXL–Suchbedingungen nacheinander in Abhängigkeit voneinander ausgewertet werden.

Ein Teilergebnis ergibt sich aus der Auswertung einer XXL–Suchbedingung und setzt sich aus einer Menge von relevanten Pfaden und den zugehörigen Variablenbelegungen zusammen. Die konjunktive Abhängigkeit der gegebenen XXL–Suchbedingungen definiert sich ausschließlich über die gemeinsam verwendeten Variablen (siehe Abschnitt 4.4). Somit genügt es, die ermittelten bisher zulässigen Variablenbelegungen weiterzugeben.

Beispiel 5.4 Betrachten wir noch einmal das Beispiel zu Beginn dieses Kapitels. Werden die Variablenbelegungen aus der Abb. 5.2 zur Auswertung der XXL–Suchbedingung W_2 verwendet, dann umfasst das Teilergebnis zu W_2 nur noch die relevanten Pfade p5, p6, p7 und p8.

Zur Berechnung einer Menge von Resultatgraphen ist die Handhabung von Teilergebnissen maßgebend. Grundsätzlich unterscheiden wir die speicherintensive Breitensuche und die laufzeitintensive Tiefensuche. Im ersten Fall werden bei der Auswertung einer XXL–Suchbedingung alle potentiellen Variablenbelegungen und alle potentiellen relevanten Pfade ermittelt. Die potentiellen Variablenbelegungen werden dann an die nächste XXL–Suchbedingung weitergegeben. Diese verwirft einen Teil der erhaltenen Variablenbelegungen und fügt neue hinzu. Auf diese Weise entstehen große Mengen von Zwischenergebnissen. Der zweite Ansatz basiert darauf, dass für eine XXL–Suchbedingung ein relevanter Pfad mit einer Variablenbelegung ermittelt wird. Die nächste XXL–Suchbedingung wird gemäß den bisher belegten Variablen ausgewertet, wobei u.U. noch nicht belegte Variablen eine Belegung erhalten. Gibt es für jede XXL–Suchbedingung zur aktuellen Variablenbelegung einen relevanten Pfad, dann wurde ein Ergenis gefunden. Andernfalls muss mit einer anderen Variablenbelegung fortgefahren werden. Diese Berechnung ist deshalb sehr zeitaufwendig, weil Teilergebnisse unter Umständen mehrfach berechnet werden.

Wir verwenden im Rahmen dieser Arbeit die Breitensuche zur Ermittlung der relevanten Resultatgraphen.

Kapitel 6

Indexstrukturen für XML–Daten

Um XXL–Anfragen auf einer großen Menge von XML–Dokumenten (also einem großen XML–Graphen) möglichst effizient im Sinne der Antwortzeit auswerten zu können, kann man durch die Wahl der globalen und lokalen Auswertungsstrategien aber auch durch die gezielte, zugriffsoptimierte Aufbereitung der zu durchsuchenden Informationen mit Hilfe von geeigneten *Indexstrukturen* Einfluss nehmen.

Im Rahmen dieser Arbeit bestehen die zu indizierenden Informationen aus den Struktur– und den Inhaltsdaten von XML–Dokumenten. Gegeben seien ein XML–Graph $X = (V, E, \Sigma^*)$ gemäß Definition 2.6 und eine beliebige XXL–Anfrage gemäß der Syntax– und Semantikdefinitionen aus Abschnitt 4.4 (siehe Abb. 6.1). Die Struktur der XML–Dokumente wird auf die n–Knoten und die Kanten zwischen ihnen abgebildet; die Inhaltsdaten der XML–Dokumente werden in den i–Knoten gespeichert, wobei es eine Kante vom zugehörigen n–Knoten gibt.

Die Semantik einer XXL–Anfrage wird auf der Basis von Resultatgraphen definiert, die Teilgraphen des XML–Graphen sind. Hierbei umfasst ein Resultatgraph zu einer gegebenen Variablenbelegung zu jeder XXL–Suchbedingung einen relevanten Pfad, der dieser Belegung genügt. Ein solcher Pfad und sein Relevanzwert für eine XXL–Suchbedingung wird durch Auswertung der gegebenen strukturbasierten Pfad– und der gegebenen datenorientierten Inhaltsbedingung bestimmt.

In diesem Abschnitt untersuchen wir die Möglichkeiten, die Namen und Werte von Elementen und Attributen in geeigneten Indexstrukturen so abzulegen, dass der Zugriff auf die Namen und Werte zur Auswertung elementarer Bedingungen möglichst effizient erfolgen kann. Bei dem Entwurf von Indexstrukturen muss immer sorgfältig auf das Verhältnis von Laufzeitgewinn und zusätzlichem Speicherplatzbedarf geachtet werden.

Beispiel 6.1 In der Abbildung 6.1 sind drei XML–Dokumente durch den zugehörigen XML–Graphen repräsentiert. Darüber hinaus ist eine XXL–Anfrage angegeben, die typische elementare Bedingungen, also Knoten– und Inhaltsbedingungen enthält.

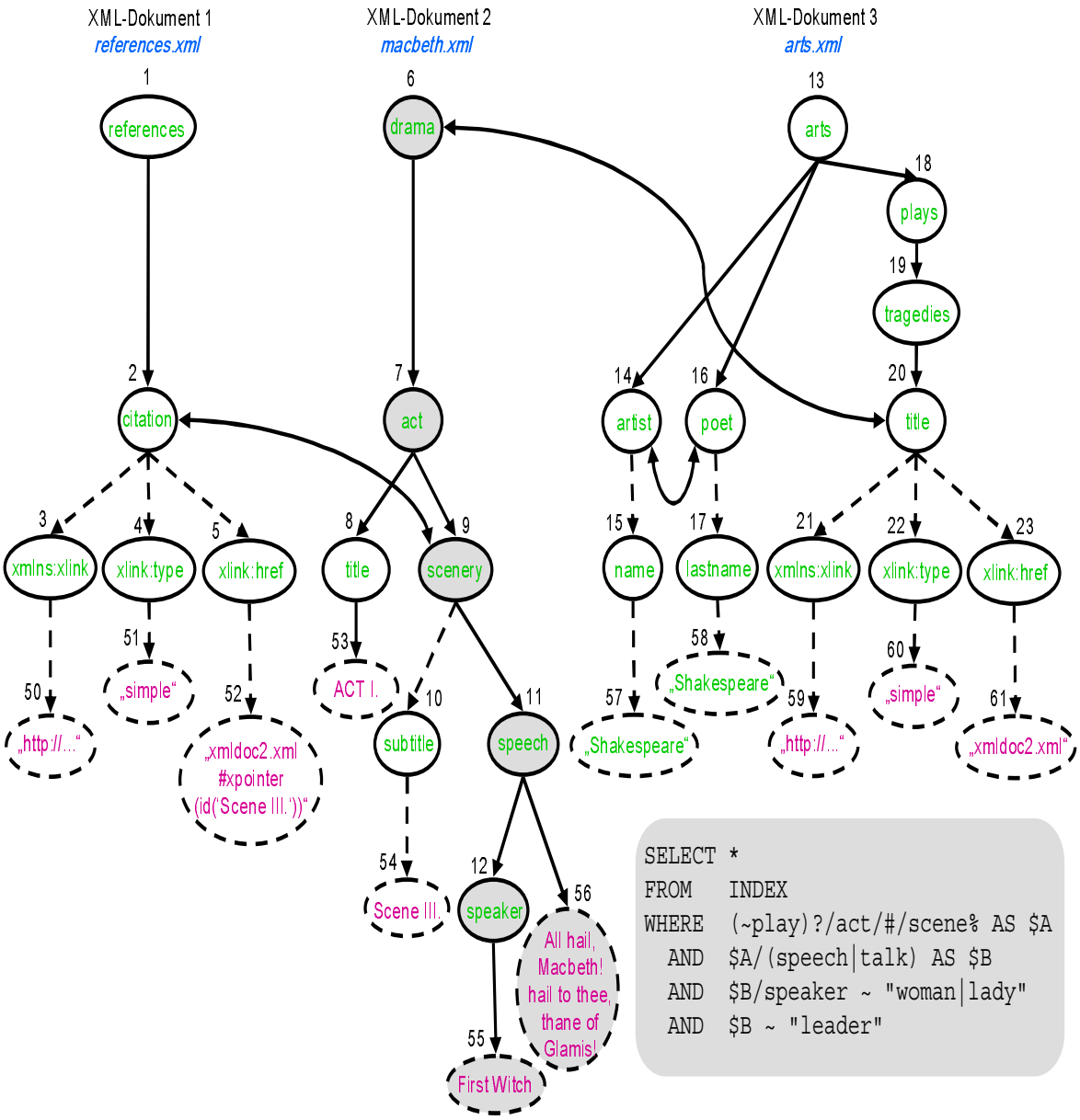


Abbildung 6.1: XML–Graphen und XXL–Anfrage

Die Knoten des XML–Graphen sind mit den Namen und Werten von Elementen und Attributen beschriftet. Jeder Knoten ist darüber hinaus durch eine eindeutige OID (object identifier) identifizierbar. In diesem Beispiel sind die n–Knoten von 1 bis 23 und die i–Knoten von 50 bis 61 durchnummeriert. Diese Nummern verwenden wir als OID der Knoten. Die schattierten Knoten des XML–Graphen stellen einen Resultatgraphen für die gegebene Anfrage dar.

Im Kapitel 5 haben wir ausführlich die naive Verarbeitung von XXL–Suchbedingungen einer gegebenen XXL–Anfrage auf der Basis eines gegebenen XML–Graphen und einer Ontologie beschrieben. Dazu haben wir 15 Funktionen eingeführt, die den Zugriff auf den XML–Graphen und somit die Auswertung jedes beliebigen XXL–Bedingungsgraphen top–down oder bottom–up unterstützen. Zur Erinnerung geben wir die Funktionen in der folgenden Tabelle noch einmal an.

Gruppe	Nr.	Funktion	Verwendung top–down	Verwendung bottom–up
1	1	*findNode(oid)*	x	x
	2	*findNodeByName(name)*	x	x
	3	*findChildren(oid)*	x	
	4	*findChildrenByName(oid, name)*	x	
	5	*testEdge()*	x	x
	6	*findDescendants(oid)*	x	
	7	*findDescendantsByName(oid, name)*	x	
	8	*testPath()*	x	x
	12	*findParents(oid)*		x
	13	*findParentsByName(oid, name)*		x
	14	*findAncestors(oid)*		x
	15	*findAncestorsByName(oid, name)*		x
2	9	*findNode(operator, condition)*	x	x
	10	*findNodeByName(name, operator, condition)*	x	x
	11	*testContent(oid, operator, condition)*	x	x

Tabelle 6.1: Funktionen

Die Funktionen der ersten Gruppe greifen dabei ausschließlich auf n–Knoten des XML–Graphen zu. Die Funktionen der zweiten Gruppe vergleichen die Werte von i–Knoten unter Berücksichtigung der zugehörigen n–Knoten. Wir benötigen damit einerseits einen Index zur Erkennung von Strukturmustern des XML–Graphen bestehend aus Kanten, Pfaden und Verbindungen zwischen gegebenen n–Knoten und zum anderen einen Index zur Auswertung von Inhaltsbedingungen, wobei die Vergleichsbasis unstrukturierter Text von i–Knoten darstellt.

Zu diesem Zwecke teilen wir den XML–Graphen in den *XML–Strukturgraphen* $X_S = (V_S, E_S, \Sigma^*)$ und den *XML–Datengraphen* $X_D = (V_D, E_D, \Sigma^*)$ auf (siehe Abb. 6.2). Dabei enthält der XML–Strukturgraph ausschließlich n–Knoten und die zugehörigen Kanten. Der XML–Datengraph enthält alle i–Knoten und den zu jedem i–Knoten gehörenden n–Knoten, sowie die entsprechenden Kanten.

Beispiel 6.2 In der Abb. 6.2 ist oben der XML–Strukturgraph X_S und unten der XML–Datengraph X_D zum gegebenen XML–Graphen X aus Abb. 6.1 jeweils schwarz dargestellt. Die grau gezeichneten Knoten gehören jeweils nicht mehr zu dem entsprechenden Teilgraphen.

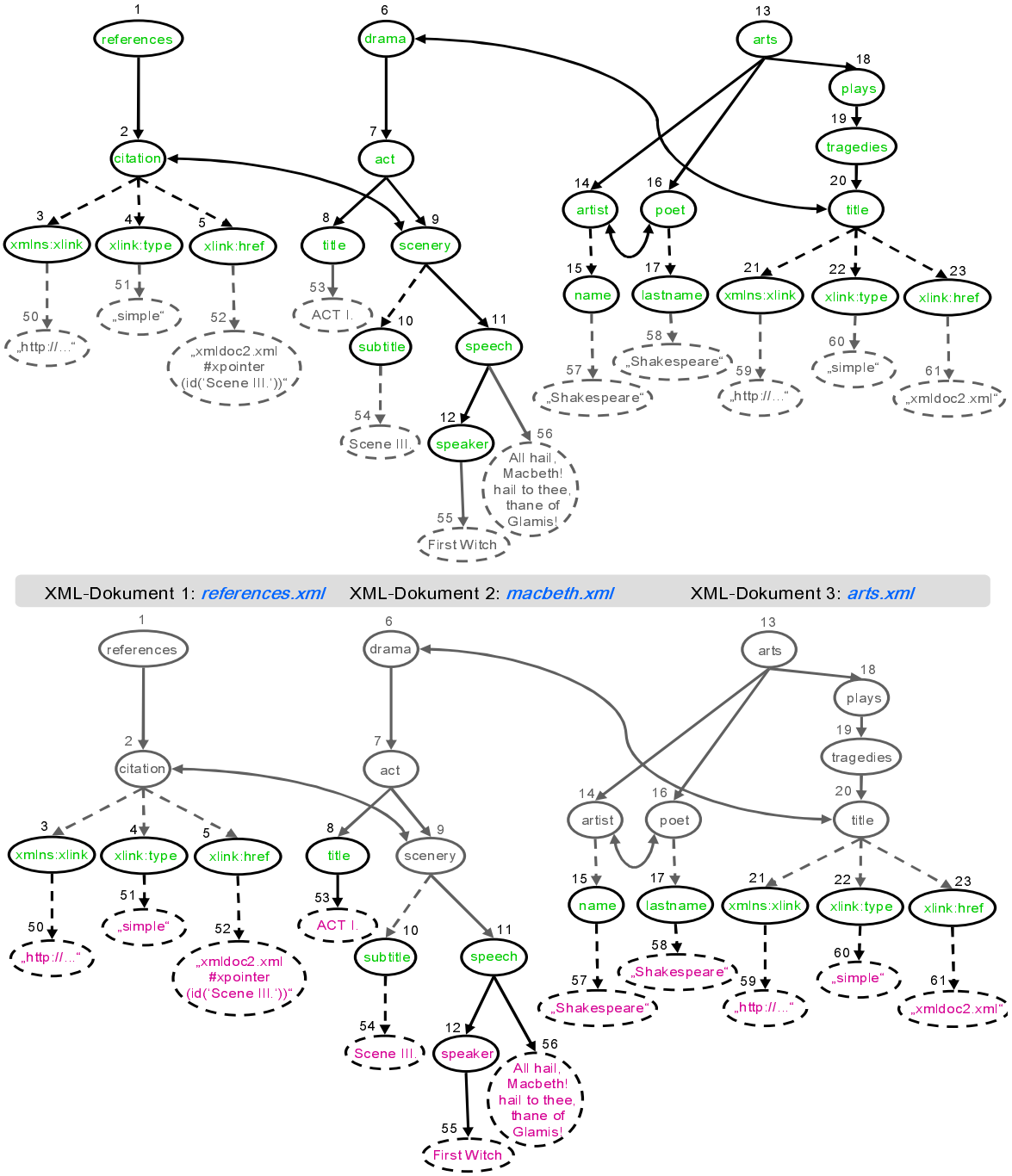


Abbildung 6.2: XML–Strukturgraph (oben) und XML–Datengraph (unten) zu gegebenem XML–Graphen

Im Rahmen dieser Arbeit verfolgen wir das Ziel, einerseits den XML–Strukturgraphen im Hinblick auf die Funktionen der ersten Gruppe und andererseits den XML–Datengraphen im Hinblick auf die Funktionen der zweiten Gruppe zu indizieren. Die Indizierung des XML–Graphen dient der Beschleunigung der Ausführbarkeit oben genannter Funktionen.

In der Vorbetrachtung beschreiben wir im Abschnitt 6.1 Standardindexstrukturen zur Indizierung von gerichteten Graphen bzw. unstrukturiertem Text. Im Abschnitt 6.2 definieren wir den Elementpfadindex zur Indizierung des XML–Strukturgraphen als Anwendung dieser Standardindexstrukturen und betrachten dabei den anfallenden Speicherplatzverbrauch und das Laufzeitverhalten der Funktionen der ersten Gruppe. Im

Abschnitt 6.3 definieren wir dann den Elementinhaltindex als Anwendung dieser Standardindexstrukturen und untersuchen Speicherplatzbedarf und Laufzeitverhalten der Methoden der zweiten Gruppe.

6.1 Vorbetrachtung

In diesem Abschnitt geben wir Definitionen von bekannten Datenstrukturen an. Wir verwenden diese Strukturen zur Indizierung des XML–Strukturgraphen bzw. des XML–Datengraphen und zeigen die Auswirkungen auf den Speicherplatzbedarf und das Laufzeitverhalten für den Zugriff auf einzelne Knoten.

Ein gerichteter Graph heisst *dünn besetzt*, wenn die Anzahl der Kanten erheblich kleiner als das Quadrat der Anzahl der Knoten ist. Liegen diese Zahlen nah beieinander, dann spricht man von einem *dicht besetzten* Graphen.

6.1.1 Adjazenzliste und Adjazenzmatrix

Es gibt zwei Standarddatenstrukturen zur Repräsentation eines gerichteten Graphen [CLR90], zum einen durch ein Array von Adjazenzlisten und zum anderen durch eine Adjazenzmatrix. Die Repräsentation eines gerichteten Graphen $G = (V, E)$ durch ein Array von Listen oder eine Matrix beruht auf der Idee, alle Kanten des Graphen durch die zugehörigen Knotenpaare darzustellen.

Definition 6.1 (Array von Adjazenzlisten)
Sei G=(V,E) ein Graph mit $|V| = n$. Die Adjazenzliste Adj[u] für einen Knoten $u \in V$ enthält alle Knoten $v \in V$, für die es eine Kante $(u, v) \in E$ gibt. Der Graph G wird durch das Array Adj der Adjazenzlisten für seine n Knoten repräsentiert.

Definition 6.2 (Adjazenzmatrix)
Sei G=(V,E) ein Graph mit $|V| = n$, wobei die Knoten von 1 bis n nummeriert sind. Die Adjazenzmatrix des Graphen G ist eine $(n \times n)$–Matrix $A = (a_{ij})$, so dass

$$a_{ij} = \begin{cases} 1 & \textit{falls } (i,j) \in E \\ 0 & \textit{sonst} \end{cases}$$

Wir verwenden nun diese Datenstrukturen zur Darstellung eines XML–Strukturgraphen. Sei der XML–Strukturgraph $X_S = (V_S, E_S, \Sigma^*)$ aus Abb. 6.2 (oben) mit n Knoten ($|V_S| = n$) und m Kanten ($|E_S| = m$) gegeben. Neben der Beschriftung basierend auf den Namen von Elementen und Attributen ordnen wir jedem Knoten einen OID zu. Adjazenzlisten und Adjazenzmatrix werden auf der Basis der OIDs angelegt.

Beispiel 6.3 In der folgenden Abbildung 6.3 wird der XML–Strukturgraph aus Abb. 6.2 (oben) zum einen durch das Array der Adjazenzlisten (links) und zum anderen durch die Adjazenzmatrix (rechts) dargestellt.

1	2
2	3 4 5 9
3	
4	
5	
6	7 20
7	8 9
8	
9	2 10 11
10	
11	12
12	
13	14 16 18
14	15 16
15	
16	14 17
17	
18	19
19	20
20	6 21 22 23
21	
22	
23	

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
20	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Abbildung 6.3: Adjazenzlisten (links) und Adjazenzmatrix (rechts) zum XML–Datengraphen aus Abb. 6.1

Wir verwenden nun diese Datenstrukturen zur Repräsentation des XML–Strukturgraphen X_S und untersuchen die Auswirkungen auf den Speicherverbrauch und die Laufzeiten der Methoden (1) bis (12). Dazu realisieren wir den XML–Strukturgraphen einmal durch eine Adjazenzmatrix A und zum anderen durch ein Array a von Adjazenzlisten, wobei die Einträge den OIDs der Knoten entsprechen. Darüber hinaus nehmen wir an, dass die Adjazenzlisten nach den enthaltenen OIDs sortiert und doppelt verkettet sind, so dass eine binäre Suche innerhalb einer Liste in logarithmischer Zeit durchführbar ist.

Für einen gegebenen XML–Strukturgraphen mit n Knoten und m Kanten benötigen wir folgenden Speicherplatz. Für die Repräsentation dieses Graphen durch Adjazenzlisten benötigen wir für jeden Knoten eine Adjazenzliste und für insgesamt jede Kante einen Eintrag, also ***O(n+m)***. Stellen wir den Graphen durch eine Adjazenzmatrix A dar, benötigen wir eine quadratische Matrix der Dimension n, also insgesamt ***O(n^2)***.

Wir betrachten nun das Laufzeitverhalten der Funktionen der ersten Gruppe aus Tabelle 6.1. Die Auswertung von Knotenbedingungen benötigt in Abhängigkeit von der Vorgabe folgende Zugriffszeiten.

- 1. *findNode (oid)*
  Kommt ein Knoten im XML–Graphen vor, dann kann in beiden Darstellungen in konstanter Zeit, also in ***O(1)***, auf die zugehörige Adjazenzliste bzw. Matrixspalte zugegriffen werden.

- 2. *findNodeByName (name)*
  Um einen Knoten anhand seiner Beschriftung zu finden, müssen wir in beiden Darstellungsarten im schlecht möglichsten Fall die Beschriftungen aller n Knoten betrachten, also ***O(n)***.
- 3. *findChildren (oid)*
  Sollen von einem gegebenen Knoten aus Kinder ermittelt werden, so kann in konstanter Zeit, also in *O(1)* die richtige Adjazenzliste bzw. die richtige Matrixspalte ausgewählt werden (siehe *1. findNode(oid)*). Um einen Kindknoten zu finden, müssen im schlecht möglichsten Fall alle Knoten der zugehörigen Adjazenzliste bzw. der Matrixspalte durchgegangen werden, und das sind maximal n Einträge. Insgesamt wird also lineare Zeit gemäß der Anzahl n der Knoten, also ***O(n)*** benötigt.
- 4. *findChildrenByName (oid, name)*
  Sollen von einem gegebenen Knoten aus Kinder eines bestimmten Namens ermittelt werden, so kann in konstanter Zeit, also in *O(1)* die richtige Adjazenzliste bzw. die richtige Matrixspalte ausgewählt werden (siehe *1. findNode(oid)*). Um einen konkreten Knoten anhand seiner Beschriftung zu finden, müssen wir in beiden Darstellungsarten im schlecht möglichsten Fall die gesamte Adjazenzliste bzw. Matrixspalte durchsuchen, wobei maximal n Knoten in Frage kommen können. Wir benötigen also insgesamt lineare Zeit ***O(n)***.
- 5. *testEdge (oid1, oid2)*
  Die Existenz einer Kante zwischen zwei gegebenen Knoten kann mit Hilfe der Adjazenzmatrix in konstanter Zeit ***O(1)*** validiert werden. Um auf die richtige Adjazenzliste zuzugreifen, benötigen wir konstante Zeit *O(1)*. Um in dieser Liste einen konkreten Knoten anhand seiner OID zu finden, benötigen wir aufgrund der Sortierung logarithmische Zeit ***O(log n)***.
- 6. *findDescendants (oid)*
  Das Ermitteln von Nachfolgern eines gegebenen Knotens erfolgt nach dem Finden des Ausgangsknotens in konstanter Zeit (siehe *1. findNode(oid)*) Kante für Kante. Die Nachfolger können z.B. mittels Breitensuche [CLR90] in der Zeit ***O(n+m)*** gefunden werden.
- 7. *findDescendantsByName (oid, name)*
  Das Ermitteln von Nachfolgern eines bestimmten Namens zu einem gegebenen Knoten erfolgt nach dem Finden des Ausgangsknotens in konstanter Zeit (siehe *1 findNode(oid)*) Kante für Kante. Um die Nachfolger mit bestimmten Namen zu finden, muss die o.g. Breitensuche trotzdem vollständig durchgeführt werden. Es wird insgesamt also die Zeit ***O(n+m)*** benötigt.
- 8. *testPath (oid1, oid2)*
  Die Validierung einer Verbindung zwischen zwei gegebenen Knoten hängt von der Länge des Pfades ab. Eine Möglichkeit besteht in der Breitensuche beginnend bei einem der Knoten, in der Hoffnung, den anderen zu Finden. Dafür wird die Zeit ***O(n+m)*** benötigt.
- 12. *findParents (oid)* und 13. *findParentsByName (oid, name)*
  Sowohl die angegebene Adjazenzmatrix als auch das Array von Adjazenzlisten unterstützt die Verfolgung von Kanten entlang ihrer gegebenen Richtung. Um zu einem gegebenen Knoten einen Elternknoten ausfindig zu machen, muss dieser Knoten in den Adjazenzlisten des Arrays oder als 1–Eintrag in einer Zeile der Matrix gesucht werden. Für beide Vorgänge ist im schlechtesten Fall in diesen Darstellungen Zeit ***O(n+m)*** erforderlich.

- 14. *findAncestors (oid)* und 15. *findAncestorsByName (oid, name)*
  Die beiden Darstellungsarten unterstützen keine Rückwärtsverfolgung von Kanten. Die Suche nach Vorgängern mit den hier gegebenen Mitteln wird sehr ineffizient, vermutlich sogar exponentiell.

Es ist offensichtlich, dass für die Bottom–Up–Auswertungsstrategie zusätzlich die Kanten entgegen ihrer Richtung indiziert werden sollten. Darüber hinaus wird deutlich, dass das Verfolgen von Pfaden mit zunehmender Länge viel Zeit kostet, so dass die Methoden 6 – 8 nicht wirklich effizient realisierbar sind.

6.1.2 B*–Baum

Um Informationen effizient für bestimmte Anfragen aufzubereiten und kompakt darzustellen, werden Suchbäume verwendet.

Definition 6.3 (Baum)
Ein Baum T=(V,E) ist ein gerichteter, azyklischer Graph mit einer endlichen Knotenmenge V und einer endlichen Kantenmenge E, bei dem es zu zwei beliebigen Knoten genau einen gerichteten Pfad gibt. Jeder Knoten außer der Wurzel hat demnach Eingangsgrad 1, die Wurzel hat Eingangsgrad 0. □

In einem *geordneten Baum* sind die Kinder jedes Knotens geordnet. Ein *binärer Baum* ist ein geordneter Baum, der rekursiv über die endliche Menge seiner Knoten definiert wird. Diese Menge enthält entweder keine Knoten oder besteht aus drei disjunkten Knotenmengen: der Wurzel, ein binärer Baum genannt linker Teilbaum und ein binärer Baum genannt rechter Teilbaum.

In einem Suchbaum werden die Informationen, nach denen gesucht werden soll, als Suchschlüssel verwendet. Ein *Mehrwegesuchbaum der Ordnung m* ist die Verallgemeinerung des binären Baums. Hierbei hat jeder Knoten höchstens m Kinder; und wenn ein Knoten $k \leq m$ Kinder hat, dann hat er genau $k-1$ Schlüssel.

Definition 6.4 (B–Baum)
Ein B–Baum T=(V,E) der Ordnung m ($m \geq 1$) *ist ein Mehrwegesuchbaum mit folgenden Eigenschaften:*

1. *Jeder Knoten außer der Wurzel enthält mindestens m Schlüssel und höchstens 2m Schlüssel.*
2. *Ein Nichtblattknoten mit k Schlüsseln* $x_1, \ldots, x_k$ *hat genau (k+1) Söhne* $t_1, \ldots, t_{k+1}$*, so dass*
   - *für alle Schlüssel s im Teilbaum* $t_i, (2 \leq i \leq k)$ *gilt:* $x_{i-1} < s < x_i$
   - *für alle Schlüssel s im Teilbaum* t_1 *gilt:* $s < x_1$ *und*
   - *für alle Schlüssel s im Teilbaum* t_{k+1} *gilt:* $s > x_k$*.*
3. *Alle Blätter haben dasselbe Niveau.*

Ein B*–Baum (engl.: B^+–Tree) ist eine Variante des B–Baums, bei dem die Daten in den Blättern stehen und die inneren Knoten nur noch Schlüssel enthalten.

Definition 6.5 (B*–Baum)
Ein B–Baum T=(V,E) der Ordnung (m,m*) ist ein Mehrwegesuchbaum* $(m, m* \geq 1)$ *mit folgenden Eigenschaften:*

1. *Jeder Knoten außer der Wurzel enthält mindestens m Schlüssel und höchstens 2m Schlüssel.*
2. *Ein Nichtblattknoten mit k Schlüsseln $x_1, \ldots, x_k$ hat genau (k+1) Söhne $t_1, \ldots, t_{k+1}$, so dass*
   - *für alle Schlüssel s im Teilbaum $t_i, (2 \leq i \leq k)$ gilt: $x_{i-1} < s \leq x_i$*
   - *für alle Schlüssel s im Teilbaum t_1 gilt: $s \leq x_1$ und*
   - *für alle Schlüssel s im Teilbaum t_{k+1} gilt: $s > x_k$.*
3. *Alle Blätter haben dasselbe Niveau.*
4. *Jedes Blatt enthält mindestens m* und höchstens 2m* Schlüssel.*

Wir verwenden den B*–Baum nun zum Indizieren des XML–Strukturgraphen und untersuchen dafür jeweils die Auswirkungen auf den Speicherplatzverbrauch und die Laufzeiten der Methoden der ersten Gruppe ersten Gruppe aus Tabelle 6.1.

Sei der XML–Strukturgraph $X_S = (V_S, E_S, \Sigma^*)$ aus Abb. 6.2 (oben) mit n Knoten ($|V_S| = n$), m Kanten ($|E_S| = m$) und n' verschiedenen Namen gegeben. Wir legen einen B*–Baum T_{OID} auf der Basis der OIDs und einen B*–Baum T_{Name} über die vorkommenden Element– und Attributnamen an (siehe dazu die folgenden zwei Abbildungen). In den Knoten stehen die doppelt verketteten und sortierten Adjazenzlisten auf der Basis der OIDs. Im schlimmsten Fall hat jeder Knoten einen anderen Namen, so dass wir für beide B*–Bäume n als schlecht möglichsten Fall zugrunde legen.

Beispiel 6.4 Sei der XML–Strukturgraph $X_S = (V_S, E_S, \Sigma^*)$ aus der Abb. 6.2 (oben) gegeben. In der folgenden Abbildung 6.4 wird ein B*–Baum $T_{OID} = (V', E')$ der Ordnung (2,2) dargestellt, der über die OIDs des Graphen X_S angelegt ist.

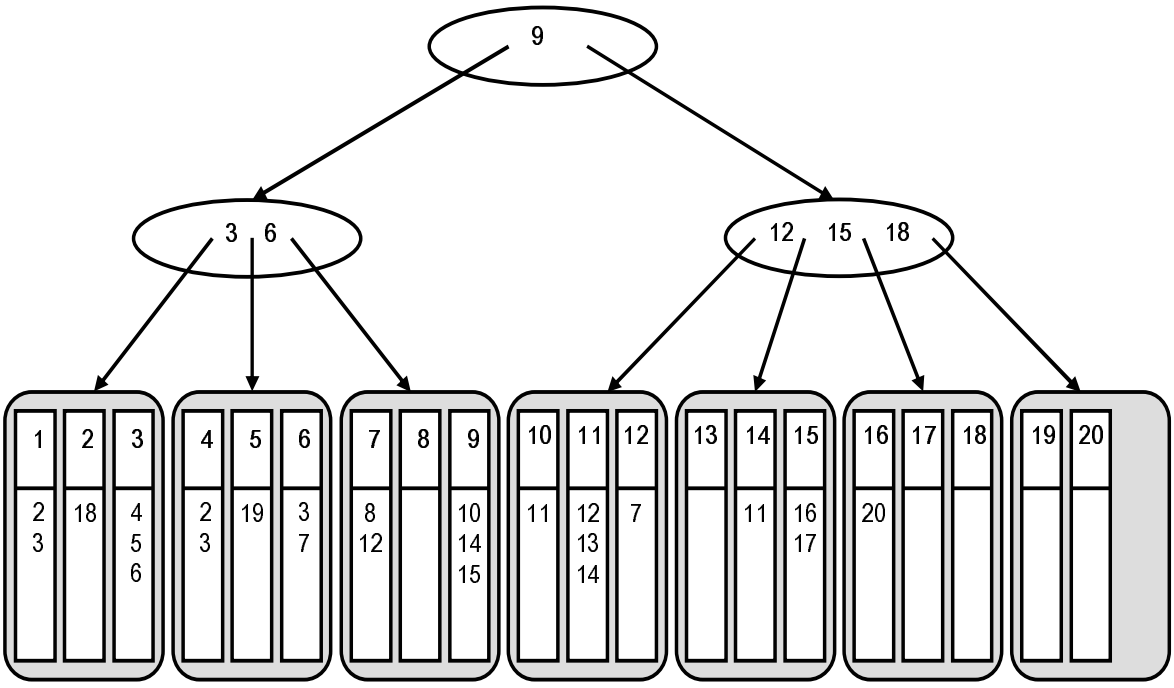


Abbildung 6.4: (2,2)–B*–Baum über die Knoten und deren ausgehende Kanten des Graphen G aus Abb. 6.2

In der folgenden Abbildung 6.5 wird ein B*–Baum $T_{Name} = (V', E')$ der Ordnung (2,2) dargestellt, der über die Namen und Inhalte des Graphen X_S angelegt ist. Die Namen und Inhalte der Knoten des Graphen werden gleichzeitig als Schlüssel der inneren Knoten des B*–Baums verwendet.

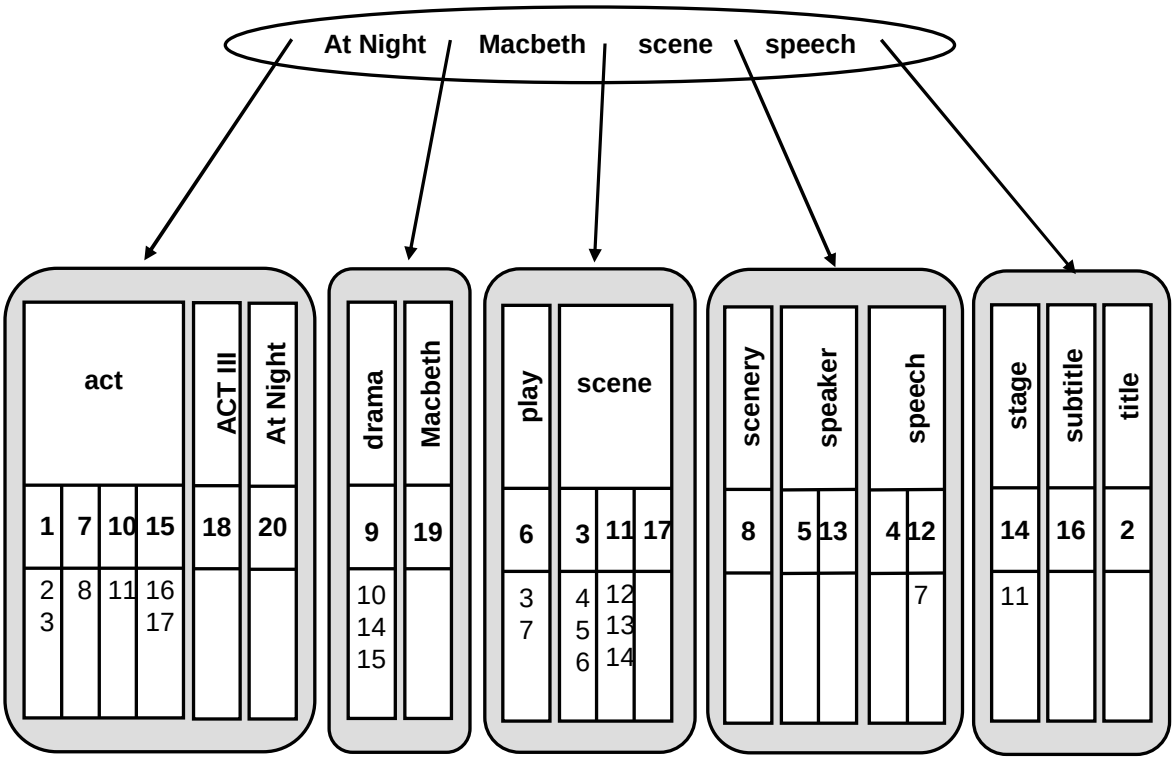


Abbildung 6.5: (2,2)–B*–Baum über die Namen und deren ausgehende Kanten des Graphen X aus Abb. 6.2

Für einen gegebenen XML–Strukturgraphen mit n Knoten und m Kanten benötigen wir folgenden Speicherplatz. Da im schlecht möglichsten Fall jeder Knoten einen anderen Namen besitzt, benötigen wir für beide B*–Bäume jeweils einen Eintrag pro Knoten in den Blättern und logarithmisch viele Suchschlüssel zum Aufbau des B*–Baums, also insgesamt ***O(log n + n)*** Platz.

Wir betrachten nun das Laufzeitverhalten der Funktionen 1 bis 12. Die Auswertung von Knotenbedingungen in Abhängigkeit von der Vorgabe benötigt folgende Zugriffszeiten.

- 1. *findNode (oid)*
  Um einen Knoten anhand seiner OID zu finden, benötigen wir im B*–Baum T_{OID} logarithmisch viele Zugriffe, also ***O(log n)*** und im B*–Baum T_{Name} im schlecht möglichsten Fall linear viele Zugriffe, also ***O(n)***.
- 2. *findNodeByName (name)*
  Um einen Knoten anhand seines Namens zu finden, benötigen wir im B*–Baum T_{OID} im schlecht möglichsten Fall linear viele Zugriffe, also ***O(n)*** und im B*–Baum T_{Name} im schlecht möglichsten Fall logarithmisch viele Zugriffe, also ***O(log n)***.
- 3. *findChildren (oid)*
  Sollen von einem gegebenen Knoten aus Kinder ermittelt werden, so wird der

gegebene Knoten in logarithmischer Zeit *O(log n)* in T_{OID} bzw. in linearer Zeit *O(n)* in T_{Name} erreicht (siehe *1. findNode(oid)*). Um Kindknoten zu finden, muss im schlecht möglichsten Fall die gesamte Adjazenzliste durchgegangen werden. In ihr gibt es maximal n Einträge. Die Gesamtzeit beträgt also ***O(log n+n)*** in T_{OID} bzw. ***O(n+n)*** in T_{Name}.

- 4. *findChildrenByName (oid, name)*
  Sollen von einem gegebenen Knoten aus Kinder eines bestimmten Namens ermittelt werden, so wird der gegebene Knoten in logarithmischer Zeit *O(log n)* in T_{OID} bzw. in linearer Zeit *O(n)* in T_{Name} erreicht (siehe *1. findNode(oid)*). Um Kindknoten zu finden, muss im schlecht möglichsten Fall die gesamte Adjazenzliste durchgegangen werden. In ihr gibt es maximal n Einträge. Die Gesamtzeit beträgt also ***O(log n+n)*** in T_{OID} bzw. ***O(n+n)*** in T_{Name}.

- 5. *testEdge (oid1, oid2)*
  Die Existenz einer Kante zwischen zwei gegebenen Knoten kann in beiden Darstellungsarten nach dem Finden des ersten Knotens in linearer Zeit durch Vergleich mit im schlecht möglichsten Fall allen Knoten durchgeführt werden, also insgesamt ***O(log n + n)*** für T_{OID} bzw. ***O(n + n)*** für T_{Name}.

- 6. *findDescendants (oid)*
  Das Ermitteln von Nachfolgern eines gegebenen Knotens erfolgt nach dem Finden des Ausgangsknotens in logarithmischer bzw. linearer Zeit (siehe *1. findNode(oid)*) Kante für Kante. Die Nachfolger können zum Beispiel durch Breitensuche gefunden werden, was die Zeit ***O(n+m)*** kostet.

- 7. *findDescendantsByName (oid, name)*
  Das Ermitteln von Nachfolgern eines bestimmten Namens zu einem gegebenen Knoten erfolgt nach dem Finden des Ausgangsknotens in logarithmischer bzw. linearer Zeit Kante für Kante (siehe *1. findNode(oid)*). Auch hier muss die Breitensuche vollständig durchgeführt werden, d.h. es wird insgesamt die Zeit ***O(n+m)*** benötigt.

- 8. *testPath (oid1, oid2)*
  Die Validierung einer Verbindung zwischen zwei gegebenen Knoten hängt von der Länge des Pfades ab. Um den richtigen Pfad zu finden, müssen im schlechtesten Fall alle bestehenden Pfade von einem der Knoten aus betrachten, was im Falle von Breitensuche die Zeit ***O(n+m)*** kostet.

- 12. *findParents (oid)* und 13. *findParentsByName (oid, name)*
  Es sind keine Rückwärtskanten modelliert, so dass die Suche nach Elternknoten darauf basiert den gegebenen Knoten in einer der Adjazenzlisten zu finden und das dauert ***O(n+m)***.

- 14. *findAncestors (oid)* und 15. *findAncestorsByName (oid, name)*
  In den B*–Bäumen sind keine Rückwärtskanten modelliert, so dass Vorgänger nur mit viel (exponentiellem) Aufwand gefunden werden können.

Es ist offensichtlich, dass auch hier für die Bottom–Up–Auswertungsstrategie zusätzlich die Kanten entgegen ihrer Richtung indiziert werden sollten. Darüber hinaus wird deutlich, dass das Verfolgen von Pfaden mit zunehmender Länge ebenfalls viel Zeit kostet, so dass die Methoden 6 – 8 nicht wirklich effizient realisierbar sind.

6.2 Elementpfadindex

Ein Elementpfadindex (kurz: EPI) dient zur effizienten Reproduktion von Strukturinformationen von XML–Dokumenten, z.B. Kanten, Pfaden, Verbindungen, Geschwisterknoten. Die konkrete Datenstruktur hängt dabei entscheidend von der Art der wiederzugebenden Strukturinformation ab.

Neben der dokumentinternen Elementhierarchie zeichnet sich XML durch die Fähigkeit aus, beliebige Elemente beliebiger XML–Dokumente mittels ID–Referenzen, XLink und XPointer in Beziehung zu setzen. Wir haben also einen gerichteten Graphen als Repräsentant einer Menge von XML–Dokumenten und keine Bäume für die einzelnen XML–Dokumente.

Die XML–Anfragesprache unterstützt dazu neben der expliziten Formulierung von Pfadausdrücken die Verwendung von Wildcardzeichen als Platzhalter für Pfade beliebiger Länge. Aus diesem Grund sind wir an einem Elementpfadindex interessiert, der zum einen effizient Eltern–Kind–Beziehungen und zum anderen Verbindungen zwischen zwei gegebenen Knoten verifizieren kann.

Sei der XML–Strukturgraph $X_S = (V_S, E_S, \Sigma^*)$ gegeben. Der EPI besteht aus einem B*–Baum und einer Menge von Adjazenzlisten. Die Daten des XML–Strukturgraphen werden dabei wie folgt angeordnet.

1. Es wird ein B*–Baum T_{Name} über die vorkommenden Element– und Attributnamen und ein B*–Baum T_{OID} über die vorkommenden OIDs angelegt.
2. In den Blättern der B*–Bäume werden zu jedem Element– und Attributnamen die zugehörigen Knoten des XML–Strukturgraphen gespeichert.
3. Zu jedem Knoten werden zwei sortierte Adjazenzlisten angelegt, wobei die eine die Väter und die andere die Kinder sortiert nach ihrem eindeutigen Bezeichner beinhaltet.

Auf diese Weise können wir einen Knoten anhand seines Namens oder seiner OID in logarithmischer Zeit finden. Um für einen gefundenen Knoten die Existenz einer Kante zu einem zweiten Eltern– bzw. Kindknoten zu prüfen wird aufgrund der Adjazenzlisten ebenfalls logarithmische Zugriffszeit benötigt.

An dieser Stelle bleibt die Frage offen, wie wir effizient Vorgänger bzw. Nachfolger finden oder eine Verbindung zwischen zwei gegebenen Knoten verifizieren können. Hierzu gibt es verschiedene Ansätze. Wir werden zwei vorstellen. Beide Verfahren beruhen auf einer Ergänzung der Informationen zu den einzelnen Knoten in den Blättern der gegebenen B*–Bäume. Im Abschnitt 6.2.1 stellen wir eine Methode vor, mit deren Hilfe die Verbindung von Knoten innerhalb einer dokumentinternen Elementhierarchie effizient in konstanter Zeit verifiziert werden kann. Im darauf folgenden Abschnitt 6.2.2 geben wir eine Methode an, mit deren Hilfe wir effizient die Verbindung von beliebigen Knoten im XML–Graphen prüfen können.

6.2.1 PrePostOrder–Überdeckung

Eine PrePostOrder–Überdeckung für einen Baum ist die kompakte Repräsentation von Verbindungen in diesem Baum.

Jedes XML–Dokument kann als Baum von Elementen aufgefasst werden, wenn interne und externe Links vernachlässigt werden. Die PrePostOrder–Kodierung weist jedem

Knoten $v \in V$ ein Paar $(pre(v), post(v))$ zu [Gru02], so dass durch den Vergleich der Markierungen zweier gegebener Knoten direkt die Erreichbarkeit getestet werden kann.

Definition 6.6 (PrePostOrder–Marke)
Sei $X' = (V', E', \Sigma^*)$ ein geordneter XML–Baum eines XML–Dokuments aus dem XML–Datengraphen X. Jedem Knoten $v \in V'$ wird eine PrePostOrder–Marke M=(pre(v),post(v)) zugewiesen, wobei pre(v),post(v) $\in \mathbb{N}_0$ sind und für zwei beliebige Knoten $x, y \in V'$ gilt:

$$\exists\ Pfad\ \langle x \ldots y \rangle\ in\ X' \iff pre(x) \leq pre(y) \wedge post(x) \geq post(y)$$

□

Die Zuordnung der PrePostOrder–Marken erfolgt während der Traversierung des Baums X'. Der Baum wird nach dem depth–first–Prinzip traversiert. Dabei kann z.B. ein Stack nach dem LIFO–Prinzip (last in–first out) von links nach rechts mit den Knoten des Baums beginnend bei der Wurzel gefüllt und wieder geleert werden. Zum Zeitpunkt des Einfügens eines Knotens in den Stack wird ihm die aktuelle PreOrder–Marke zugewiesen, also bevor dessen Kinder von links nach rechts traversiert wurden. Beim Entfernen eines Knotens aus dem Stack wird ihm die aktuelle PostOrder–Marke zugewiesen, also nachdem alle seine Kinder traversiert wurden.

PrePostOrder(G)

```
(1) foreach (v ∈ V_N) do
(2)    visit(v) = false
(3)    pre(v)=0
(4)    post(v)=0
(5) od
(6) current_pre = 1
(7) current_post = 1
(8) DepthFirst(v)                    // v ∈ V_N root node
```

DepthFirst(v)

```
(1) visit(v) = true
(2) pre(v)=current_pre
(3) current_pre++
(4) foreach (u ∈ Adj(v)) do
(5)    if (visit(u)=false) then
(6)       DepthFirst(u)
(7)    fi
(8) od
(9) post(v)=current_post
(10) current_post++
```

Für einen Baum X' mit n Knoten benötigt der Algorithmus die Zeit $O(n+m)$ zur Traversierung des gesamten Baums und zur Bezeichnung aller Knoten mit den entsprechenden PreOrder– und PostOrder–Markierungen.

Für zwei Knoten, die aus demselben Dokument stammen, kann in konstanter Zeit $O(1)$ der Erreichbarkeitstest durch Vergleich der beiden PrePostOrder–Marken durchgeführt werden. Da die internen und externen Links nicht berücksichtigt werden, ist eine negative Aussage keine Garantie für die Unerreichbarkeit.

Beispiel 6.5 Wir betrachten den XML–Strukturgraphen X_S aus Abb. 6.2, wobei wir die Element– und Attributnamen momentan unbeachtet lassen können. Jedem Knoten v ist das Paar $pre(v)/post(v)$ zugewiesen, dass sich aus der Traversierung des zugehörigen XML–Baums ergibt. Die grau gezeichneten Links wurden dabei nicht berücksichtigt.

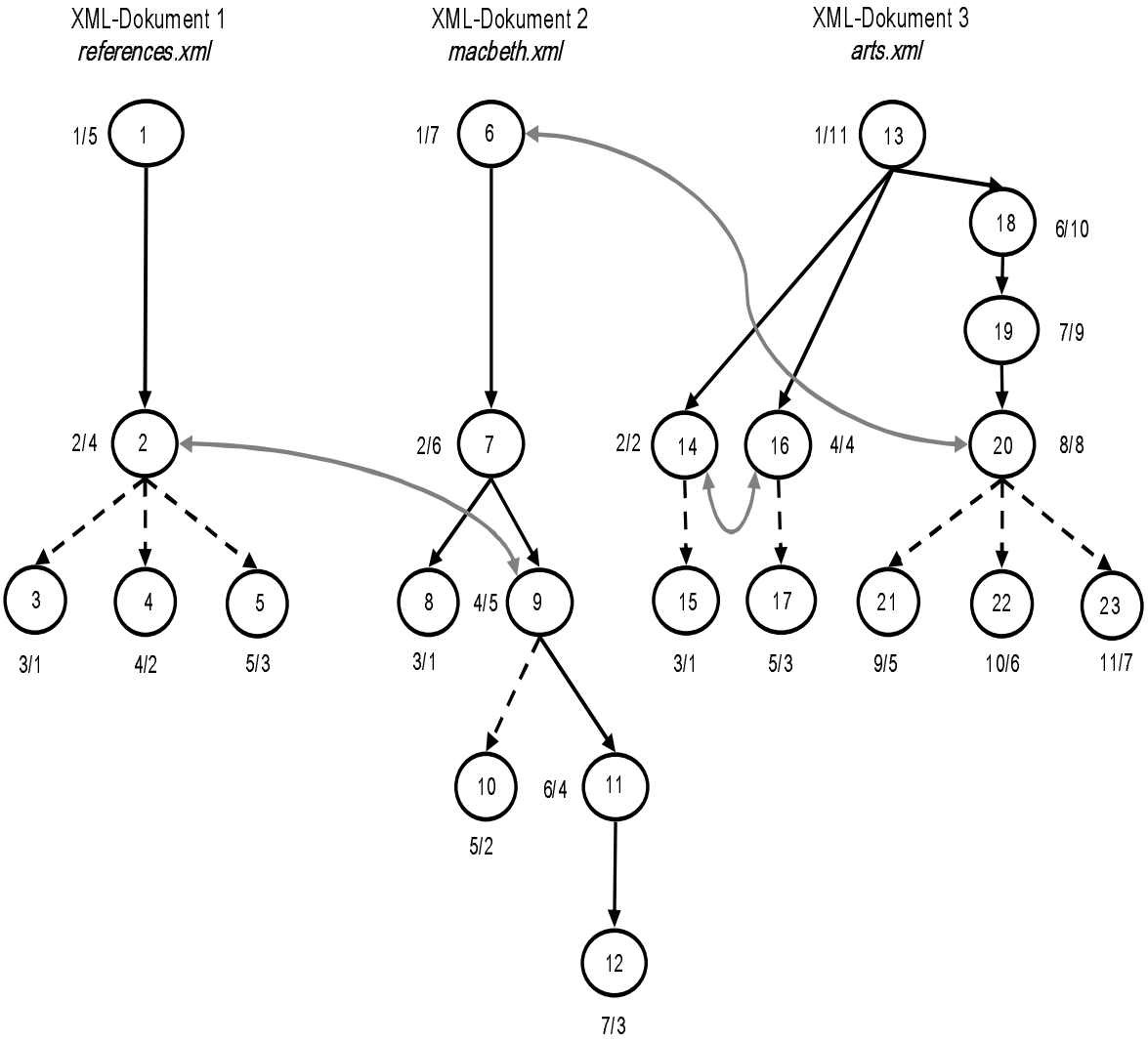


Abbildung 6.6: PrePostOrder–Markierung der Knoten innerhalb der XML–Bäume

Gemäß der Definition 6.6 wird ermittelt, dass es zwischen den Knoten 13 und 15 bzw. 13 und 17 jeweils einen Pfad gibt, zwischen den Knoten 14 und 17 keinen Pfad gibt.

Der entscheidende Nachteil dieser Zusatzinformationen besteht darin, dass nur Kanten aber keine Links berücksichtigt werden. Dieses Verfahren ist nur für die Kodierung von Verbidnungsinformationen in Bäumen geeignet, nicht aber für den Einsatz in einem gerichteten XML–Graphen.

6.2.2 2Hop–Überdeckung

Als 2Hop–Überdeckung eines gerichteten Graphen wird die kompakte Repräsentation von Verbindungen in diesem Graphen bezeichnet. Dieser Ansatz wurde von Cohen et al. [CHKZ02] entwickelt.

Sei der gerichtete Graph $G = (V, E)$ gegeben, Dann ist $G^* = (V, E^*)$ mit $E^* = \{(u, v)|$es gibt einen Pfad von u nach v in G$\}$ die transitive Hülle von G, wobei E^* alle in G bestehenden Verbindungen umfasst.

Wir illustrieren zunächst die Idee einer 2Hop–Überdeckung für einen gerichteten Graphen G.

Beispiel 6.6 In der Abb. 6.7 ist ein XML–Strukturgraph dargestellt, wobei an den Knoten die Informationen der 2Hop–Überdeckung notiert sind.

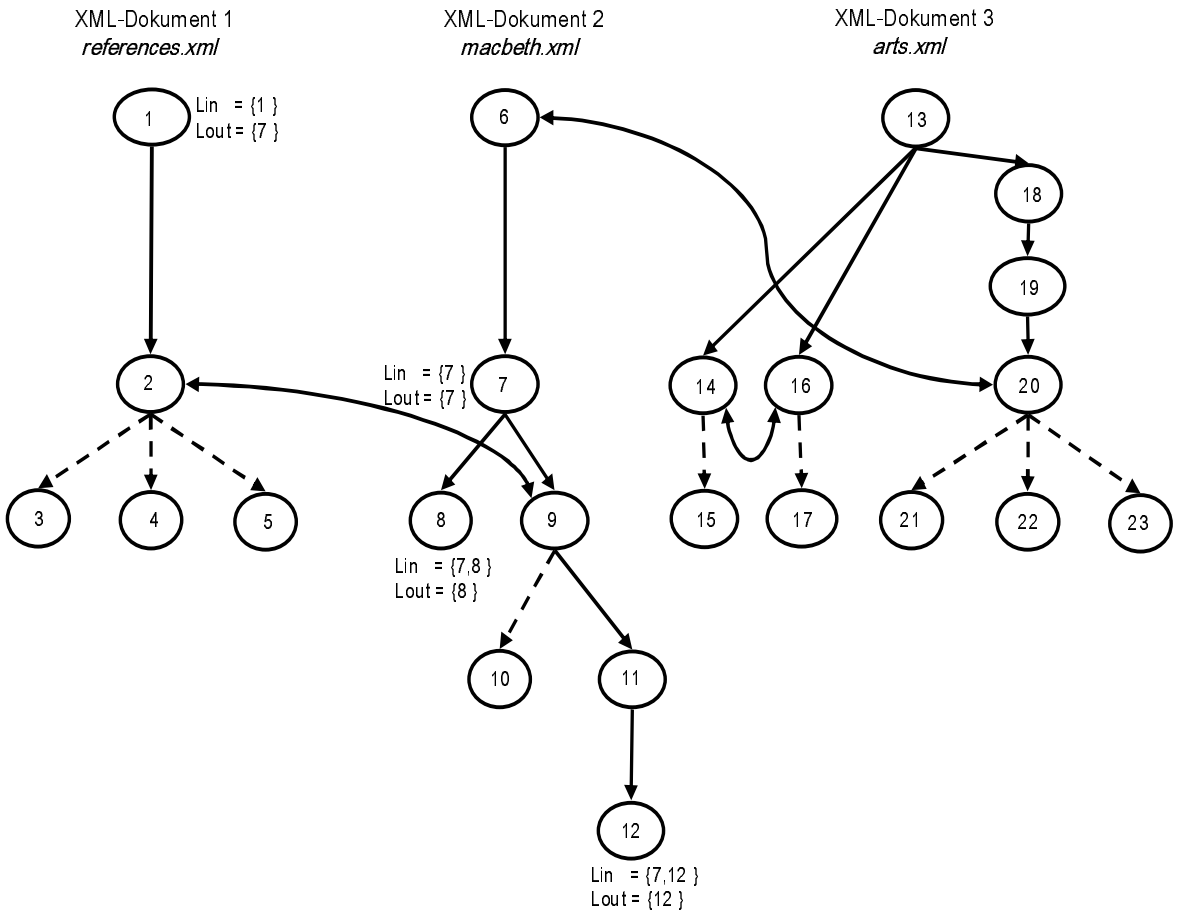


Abbildung 6.7: 2Hop–Markierung der Knoten des XML–Datengraphen

In diesem Beispiel gibt es offensichtlich eine Verbindung vom Knoten $u = 1$ zum Knoten $v = 12$, jedoch keine vom Knoten $w = 8$ zum Knoten $v = 12$.

Sei $T = \{(u, v)|$es gibt einen Pfad von u nach v in G$\}$ die Menge der Verbindungen in G. Für jede Verbindung (u, v) in G (d.h. $(u, v) \in T$) wird ein Knoten $w \in V$ auf einem Pfad von u nach v als *Centerknoten* gewählt. Dieser Knoten wird zu einer Menge $L_{out}(u)$ von Nachfahren von u und zu einer Menge $L_{in}(v)$ von Vorfahren von v hinzugefügt. Auf diese Weise kann effizient verifiziert werden, ob zwei Knoten u und v über einen Pfad verbunden sind, indem getestet wird, ob $L_{out}(u) \cap L_{in}(v) \neq \emptyset$. Es gibt eine Verbindung von u nach v genau dann, wenn $L_{out}(u) \cap L_{in}(v) \neq \emptyset$. Diese Verbindung von u nach v ist durch einen ersten Hop (Sprung) von u zu einem Knoten $w \in L_{out}(u) \cap L_{in}(v)$ und durch einen zweiten Hop von diesem w zu v gegeben. Daher auch der Name dieses Verfahrens (siehe Abb. 6.7).

Im Folgenden geben wir eine formale Definition einer 2Hop–Überdeckung für einen gerichteten Graphen an. Wir übertragen den allgemeinen Ansatz aus [CHKZ02] auf das Problem, einen gerichteten XML–Strukturgraphen entsprechend zu annotieren. Dabei weichen wir in Notation und Terminologie von der Originalarbeit ab und beschränken uns auf den Teil des Ansatzes, der für uns relevant ist.

Ein 2Hop–Label eines Knotens beinhaltet eine Menge von Vorfahren und eine Menge von Nachfahren. Diese Mengen müssen nicht zwingend vollständig sein.

Definition 6.7 (2Hop–Label)
Sei G=(V,E) ein gerichteter Graph. Jedem Knoten $v \in V$ wird eine 2Hop–Label $L(v) = (L_{in}(v), L_{out}(v))$ zugewiesen, wobei $L_{in}(v), L_{out}(v) \subseteq V$, so dass es für jeden Knoten $x \in L_{in}(v)$ einen Pfad $\langle x \ldots v \rangle$ in G bzw. für jeden Knoten $y \in L_{out}(v)$ einen Pfad $\langle v \ldots y \rangle$ in G gibt. □

Die Idee, Erreichbarkeitsinformationen eines gerichteten Graphen mit Hilfe von 2Hop–Labels zu kodieren, basiert auf folgender Eigenschaft dieser Labels.

Satz 6.1 *Sei $G = (V, E)$ ein gerichteter Graph und seien $u, v \in V$ zwei Knoten mit den 2Hop–Labels $L(u)$ and $L(v)$. Falls es einen Knoten $w \in V$ gibt, so dass $w \in L_{out}(u) \cap L_{in}(v)$, dann gibt es eine Verbindung von u nach v in G, also mindestens einen Pfad von u nach v in G.* □

Beweis:
Dieser Satz ist eine offensichtliche Folge der Definition 6.7. □

Ein 2Hop–Labeling eines gerichteten Graphen G weist jedem Knoten von G ein 2Hop–Label gemäß Definition 6.7 zu. Eine 2Hop–Überdeckung eines gerichteten Graphen ist ein 2Hop–Labeling, das alle Pfade (d.h. alle bestehenden Verbindungen) von G überdeckt.

Definition 6.8 (2Hop–Überdeckung)
Sei G=(V,E) ein gerichteter Graph. Eine 2Hop–Überdeckung ist ein 2Hop–Labeling des Graphen G, so dass gilt: wenn es eine Verbindung von u nach v in G gibt, dann ist $L_{out}(u) \cap L_{in}(v) \neq \emptyset$. □

Die Größe der 2Hop–Überdeckung ergibt sich aus der Summe aller 2Hop–Labelgrößen:

$$\sum_{v \in V} (|L_{in}(v)| + |L_{out}(v)|)$$

Zur Darstellung der transitiven Hülle eines gerichteten Graphen mit Hilfe einer 2Hop–Überdeckung sind wir an einer 2Hop–Überdeckung minimaler Größe interessiert. Das

Minimum–Set–Cover–Problem lässt sich auf das Problem, eine minimale 2Hop–Überdeckung zu finden, reduzieren [CHKZ02, CLR90]. Damit ist unser Problem NP–hart. Folglich wird ein Algorithmus benötigt, der für gerichtete Graphen eine annähernd minimale 2Hop–Überdeckung berechnet. In [CHKZ02] wird ein polynomieller Algorithmus vorgestellt, der eine 2Hop–Überdeckung berechnet, dessen Größe bewiesener Maßen höchstens um einen Faktor $O(log|V|)$ von der Größe einer minimalen 2Hop–Überdeckung abweicht.

Dieser Algorithmus funktioniert wie folgt. Sei $G = (V, E)$ ein gerichteter Graph und $G^* = (V, T)$ seine transitive Hülle. Für einen Knoten $w \in V$ ist $C_{in}(w) = \{v \in V | (v, w) \in T\}$ die Menge der Knoten $v \in V$, für die es eine Verbindung (also mindestens einen Pfad) von v nach w in G gibt. Analog ist $C_{out}(w) = \{v \in V | w, v \in T\}$ die Menge der Knoten $v \in V$, für die es eine Verbindung (also mindestens einen Pfad) von w nach v in G gibt. Somit ist $C_{in}(w)$ die Menge der Vorfahren und $C_{out}(w)$ die Menge der Nachfahren von w in G.

Für ein gewähltes $C'_{in}(w) \subseteq C_{in}(w)$ und ein gewähltes $C'_{out}(w) \subseteq C_{out}(w)$ und dem Knoten w ist die Menge $S(C'_{in}(w), w, C'_{out}(w)) \subseteq \{(u, v) \in T | u \in C'_{in}(w) \wedge v \in C'_{out}(w)\} = \{(u, v) \in T | (u, w) \in T \wedge (w, v) \in T\}$ die Menge der Verbindungen in G, auf denen der Knoten w vorkommt. w wird dabei als *Centerknoten* der Menge $S(C'_{in}(w), w, C'_{out}(w))$ bezeichnet.

Für ein gegebenes 2Hop–Labeling, das noch keine 2Hop–Überdeckung ist, sei $T' \subseteq T$ die Menge der noch nicht überdeckten Verbindungen. Damit enthält $S(C'_{in}(w), w, C'_{out}(w)) \cap T'$ die Menge der noch nicht überdeckten Verbindungen, auf denen w vorkommt. Der Quotient

$$r(w) = \frac{|S(C'_{in}(w), w, C'_{out}(w)) \cap T'|}{|C'_{in}(w)| + |C'_{out}(w)|}$$

beschreibt das Verhältnis zwischen der Anzahl der noch nicht überdeckten Verbindungen, auf denen w vorkommt, und der Anzahl der Knoten, die auf diesen Verbindungen vorkommen.

Der Algorithmus zur Berechnung einer annähernd minimalen 2Hop–Überdeckung startet mit der Menge $T' = T$ und leeren 2Hop–Labels für alle Knoten von G. In jedem Schritt enthält T' die noch nicht überdeckten Verbindungen in G. In Greedy–Manier wählt der Algorithmus den "besten" Knoten $w \in V$, der mit einer möglichst kleinen Anzahl von Knoten möglichst viele noch nicht überdeckte Verbindungen abdeckt. Durch die Wahl von w auf der Basis des größten Quotienten $r(w)$ erhalten wir eine kleine Menge von Knoten, die viele noch nicht überdeckte Verbindungen abdeckt und dabei das bestehende 2Hop–Labeling so gering wie möglich vergrößert. Nachdem also $w, C'_{in}(w)$ und $C'_{out}(w)$ gewählt wurden, werden diese Knoten zur Aktualisierung des 2Hop–Labelings verwendet:

$$\begin{aligned} &\forall v \in C'_{in}(w): \; L_{out}(v) := L_{out}(v) \cup \{w\} \\ &\forall v \in C'_{out}(w): L_{in}(v) \; := L_{in}(v) \cup \{w\} \end{aligned}$$

und die Menge T' um die Menge $S(C'_{in}(w), w, C'_{out}(w))$ der nun überdeckten Verbindungen reduziert. Der Algorithmus terminiert, wenn die Menge T' leer ist, d.h. also wenn alle Verbindungen von T durch die 2Hop–Überdeckung für G abgedeckt sind.

Für einen Knoten w stellen die Mengen $C'_{in}(w)$ und $C'_{out}(w)$ jeweils eine Teilmenge der Vorgänger $C_{in}(w)$ bzw. Nachfolger $C_{out}(w)$ von w aus der kompletten transitiven Hülle dar. Dafür gibt es in einem Berechnungsschritt exponentiell viele Teilmengen $C'_{in}(w)$

und $C'_{out}(w)$. Folglich benötigt obiger Algorithmus eine exponentielle Laufzeit zur Berechnung einer annähernd minimalen 2Hop–Überdeckung für eine gegebene Menge T. Darüber hinaus wird während der gesamten Berechnung die Menge T beibehalten, was insbesondere für große Graphen zu erheblichen Speicherengpässen führen kann.

Zur Verbesserung des Algorithmus hinsichtlich seines Laufzeitverhaltens und seines Speicherhaushalts nutzen wir folgenden Zusammenhang aus. Das Problem, die Mengen $C'_{in}(w), C'_{out}(w) \subseteq V$ für einen gegebenen Knoten $w \in V$ zu bestimmen, so dass der Quotient $r(w)$ minimal ist, entspricht genau dem Problem, den *dichtesten Teilgraphen* (engl.: densest subgraph) des *Centergraphen* von w zu finden.

Wir konstruieren für einen gegebenen Knoten w einen ungerichteten, bipartiten Centergraphen, der die noch nicht überdeckten Verbindungen repräsentiert, auf denen w vorkommt.

Beispiel 6.7 Sei der gerichtete Graph $G = X_S$ aus Abb. 6.2 gegeben, wobei wir hier nur die Knotennummern betrachten. Die Namen der Knoten spielen bei dieser Berechnung keine Rolle. In der folgenden Abbildung wird der Centergraph für den Knoten $w = 14$ dargestellt, wobei wir annehmen, dass $T' = T$ ist, also noch keine Verbindungen überdeckt sind.

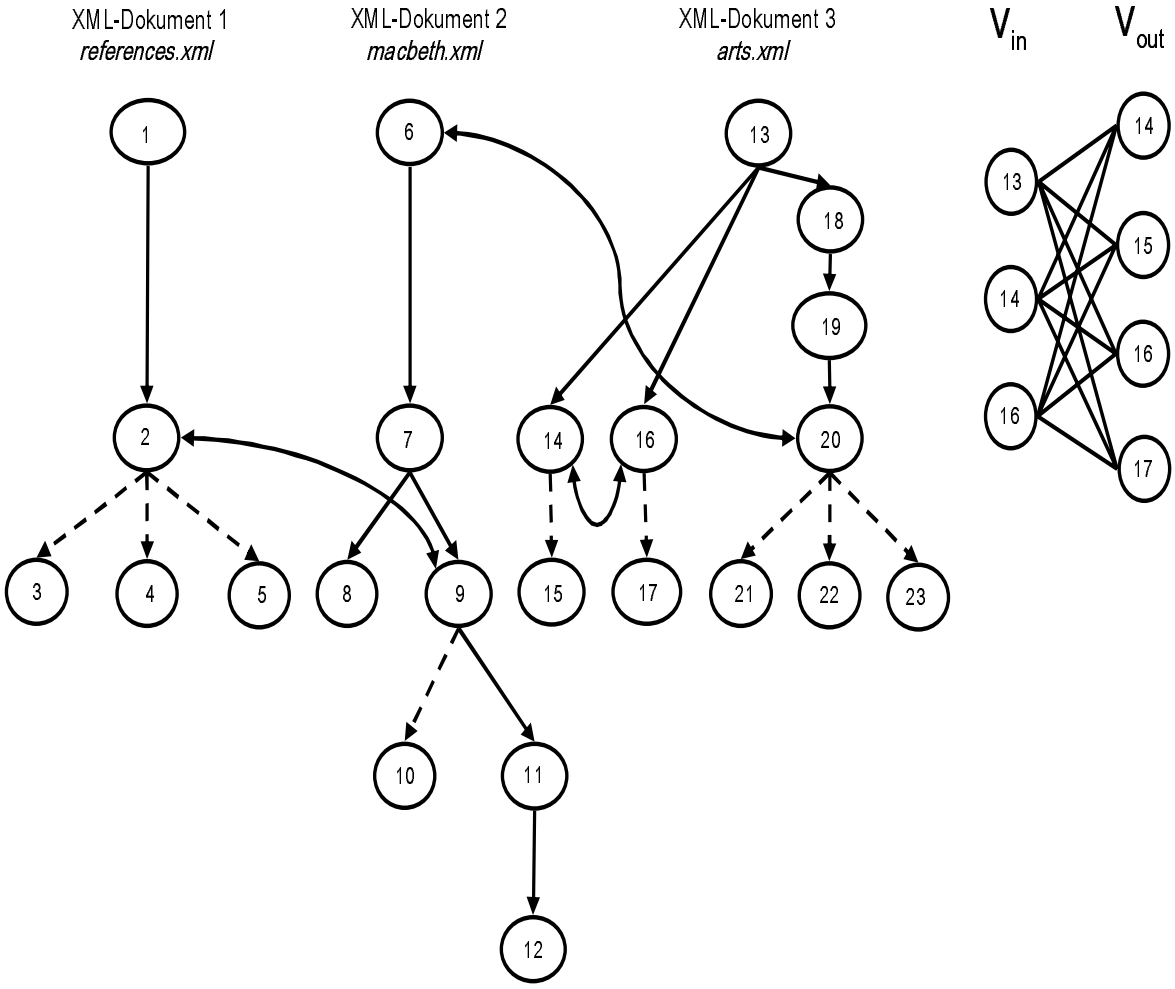


Abbildung 6.8: Gerichteter Graph G (links) und Centergraph G_w für $w = 15$ (rechts)

Die Knotenmenge V_w enthält für jeden Knoten $v \in V$ des Originalgraphen zwei Knoten v_{in} und v_{out}. Es gibt eine ungerichtete Kante $(u_{out}, v_{in}) \in E_w$ genau dann, wenn die Verbindung $(u, v) \in T'$ noch nicht überdeckt ist und $u \in C'_{in}(w)$ und $v \in C'_{out}(w)$ ist. Abschließend werden alle Knoten ohne ein– und ausgehende Kanten vom Centergraphen entfernt (siehe Abb. 6.8).

Definition 6.9 (Centergraph)
Sei G=(V,E) ein gerichteter Graph. Für ein gegebenes 2Hop–Labeling sei $T' \subseteq T$ die Menge der noch nicht überdeckten Verbindungen in G und sei $w \in V$ ein Knoten in G. Der Centergraph $CG_w = (V_w, E_w)$ für w ist ein ungerichteter bipartiter Graph mit der Knotenmenge V_w und der Kantenmenge E_w.

Die Menge der Knoten ergibt sich aus $V_w = V_{in} \cup V_{out}$, wobei $V_{in} = \{u_{in} | u \in V : \exists v \in V : (u, v) \in T'$ und $u \in C_{in}(w)$ und $v \in C_{out}(w)\}$ und $V_{out} = \{v_{in} | v \in V : \exists u \in V : (u, v) \in T'$ und $u \in C_{in}(w)$ und $v \in C_{out}(w)\}$.

Es gibt eine ungerichtete Kante $(u_{out}, v_{in}) \in E_w$ genau dann, wenn $(u, v) \in T'$ und $u \in C_{in}(w)$ und $v \in C_{out}(w)$. □

Die Dichte eines beliebigen Teilgraphen ergibt sich aus dem durchschnittlichen Grad seiner Knoten, also der Anzahl der eingehenden und ausgehenden Kanten. Der dichteste Teilgraph eines gegebenen Centergraphen kann mit Hilfe eines Approximationsalgorihtmus in linearer Zeit berechnet werden [CLR90]. Iterativ wird der Knoten mit dem kleinsten Grad aus dem Graphen entfernt. Auf diese Weise entsteht eine Folge von Teilgraphen mit entsprechenden Dichten. Der Algorithmus kann am Ende aus dieser Folge von Teilgraphen den mit der größten Dichte als dichtesten Teilgraphen des Centergraphen zurückgeben.

Der dichteste Teilgraph eines gegebenen ungerichteten Graphen ist der Teilgraph, bei dem möglichst viele Kanten mit möglichst wenigen Knoten dargestellt werden (Def. 6.10 und Abb. 6.9).

Definition 6.10 (dichtester Teilgraph)
Sei CG=(V,E) ein ungerichteter Graph. Das "dichtester Teilgraph"–Problem (densest subgraph problem) bedeutet: finde eine Teilmenge $V' \subseteq V$, für die der durchschnittliche Grad eines Knotens, also die Anzahl seiner ausgehenden Kanten, im Teilgraphen CG'=(V', E') maximal ist, d.h. $d = |E'|/|V'|$ ist maximal. Dabei ist E' die Menge der Kanten aus E, die zwei Knoten in V' verbindet. □

Beispiel 6.8 Sei der Centergraph eines Knotens gegeben (links). Für diesen Centergraphen wird nun schrittweise jeder Teilgraph auf seine Dichte hin überprüft und der dichteste Teilgraph des Centergraphen (dritter von links) ermittelt. Es wird in jedem Schritt der Knoten mit minimalem Grad (gestrichelt gezeichnet) entfernt.

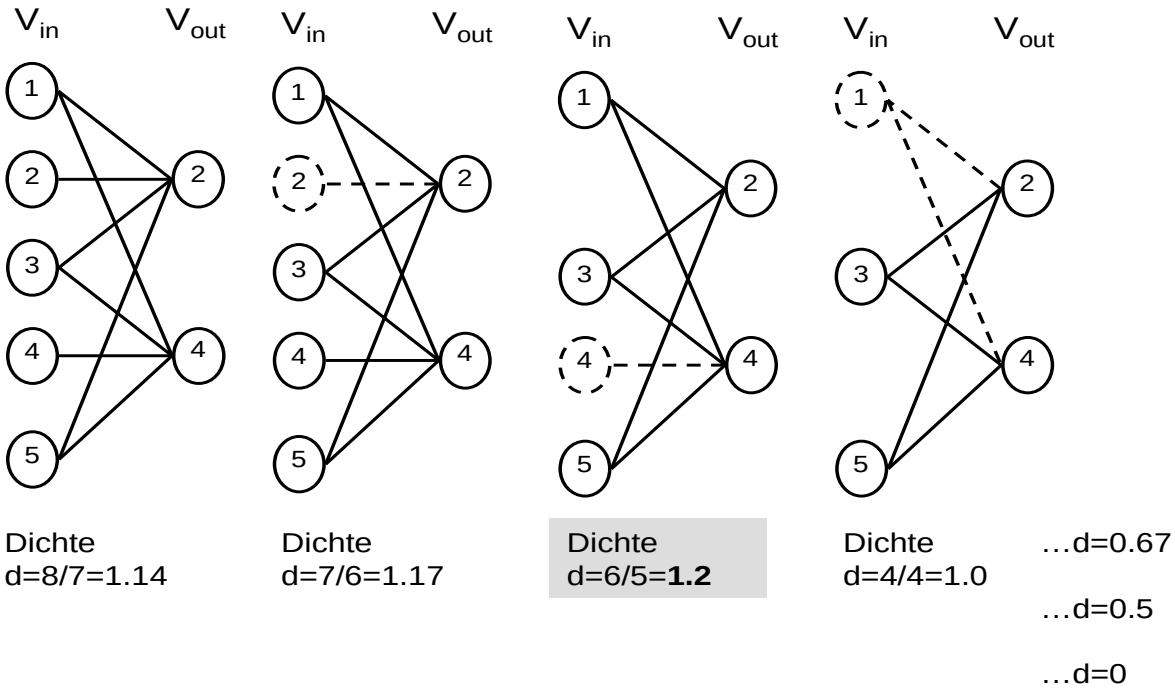


Abbildung 6.9: Berechnung des dichtesten Teilgraphen eines gegebenen Centergraphen

In diesem Beispiel hat der Centergraph die Dichte $d = 1.14$ und sein dichtester Teilgraph die Dichte $d = 1.2$.

Der verbesserte Algorithmus zur Berechnung einer annähernd minimalen 2Hop–Überdeckung basiert auf der Auswahl des "besten" Knotens w aus der Menge der verbleibenden Knoten $v \in V$, die absteigend nach der Dichte der dichtesten Teilgraphen $CG'_v = (V'_v = V'_{in} \cup V'_{out}, E'_v)$ ihrer Centergraphen $CG_v = (V_v, E_v)$. Auf diese Weise lassen sich effizient die Mengen $C_{in}(w) = V'_{in}$ und $C_{out}(w) = V'_{out}$ für einen gegebenen Knoten w mit maximalem Quotienten $r(w)$ berechnen. Dieser Algorithmus berechnet in polynomieller Zeit eine annähernd minimale 2Hop–Überdeckung für T, wobei durch die Sortierung der verbleibenden Knoten nach den aktuellen Dichten der dichtesten Teilgraphen ihrer Centergraphen die Menge T zur Initialisierung von T' benötigt wird.

Im Rahmen dieser Arbeit nutzen wir eine Eigenschaft von Centergraphen aus, um die Anzahl der zur berechnenden dichtesten Teilgraphen im durchschnittlichen Fall zu reduzieren. Die worst–case Laufzeit verändert sich nicht.

In einem Berechnungsschritt wird eine bestimmte Anzahl von Verbindungen entfernt, wobei sich nur ein Teil der Centergraphen verändert. Es ist zum einen also nicht erforderlich, dass in jedem Schritt die Centergraphen komplett neu berechnet werden. Darüber hinaus lässt sich zeigen, dass die Dichte des dichtesten Teilgraphen eines Centergraphen durch das Entfernen von Verbindungen nicht erhöht werden kann.

Aus diesem Grund wird in einer weiteren Modifikation des obigen Algorithmus für jeden Knoten $w \in V$ des Graphen G die Dichte d_w des dichtesten Teilgraphen des zugehörigen Centergraphen berechnet und in einer Prioritätenwarteschlange mit der Priorität "Dichte" absteigend nach der Dichte d_w abgelegt. In jedem Schritt des Algorithmus wird der Knoten m mit der maximalen Dichte d_m aus der Warteschlange entnommen und hinsichtlich seiner aktuellen Dichte überprüft. Stimmt die ursprüngliche Dichte mit der aktuellen Dichte nicht überein, dann wird der Knoten m mit der neuen Dichte d'_m in die Warteschlange einsortiert und der nächste "beste" Knoten betrachtet, bis

einer mit unveränderter Dichte gefunden wird. Der Algorithmus terminiert, wenn die Prioritätenwarteschlange leer ist.

Darüber hinaus kann die Anzahl der Berechnung von Teilgraphen und deren Dichten zu einem gegebenen Centergraphen durch Ausnutzung einer weiteren Eigenschaft dramatisch reduziert werden.

Wir nennen einen Centergraphen vollständig, falls es zwischen allen Knoten $u \in V_{in}$ und $v \in V_{out}$ Kanten gibt. Wir können folgende Eigenschaft zeigen.

Lemma 6.1 *Sei $G = (V, E)$ ein gerichteter Graph und T' die Menge der noch nicht überdeckten Verbindungen. Ein vollständiger Teilgraph CG'_w des Centergraphen CG_w für den Knoten w ist immer sein eigener dichteste Teilgraph.* □

Beweis: Für einen vollständigen Teilgraphen $CG'_w = (V'_{in} \cup V'_{out}, E'_w)$ gilt: $|V'_{in}| \cdot |V'_{out}| = |E'_w|$. Durch einfache Umformungen kann man zeigen, dass die Dichte $d = (|V'_{in}| \cdot |V'_{out}|)/(|V'_{in}| + |V'_{out}|)$ maximal ist. □

Mit Hilfe dieses Lemmas können wir zeigen, dass die Centergraph zu Beginn des Algorithmus immer ihre dichtesten Teilgraphen selbst sind.

Lemma 6.2 *Sei $G = (V, E)$ ein gerichteter Graph und $T' = T$ die Menge der noch nicht überdeckten Verbindungen. Der Centergraph CG_w eines Knotens $w \in V$ ist selbst sein dichtester Teilgraph.* □

Beweis: Wir zeigen, dass die anfänglichen Centergraphen immer vollständig sind und damit obiges Lemma angewendet werden kann.

Sei T die Menge aller Verbindungen in G. Wir nehmen an, dass es einen Knoten w gibt, so dass der zugehörige Centergraph nicht vollständig ist. Folglich gilt eine der drei Annahmen:

1. Es gibt zwei Knoten $u \in V_{in}, v \in V_{out}$, so dass $(u, v) \notin E_w$.
2. Es gibt mindestens einen Knoten $x \in V_{out}$, so dass $(u, x) \in E_w$.
3. Es gibt mindestens einen Knoten $y \in V_{in}$, so dass $(y, v) \in E_w$.

Aus der Definition 6.9 geht hervor, dass aus den Bedingungen 2 und 3 folgt, dass $(u, w), (w, x) \in T$ und $(y, w), (w, v) \in T$. Aber wenn $(u, w) \in T$ und $(w, v) \in T$ sind, dann ist insbesondere $(u, v) \in E_w$. Das ist aber ein Widerspruch zur ersten Annahme. Daraus folgt, dass der anfängliche Centergraph immer selbst sein dichtester Teilgraph ist. □

Zum Abschluss geben wir die wichtigsten Algorithmen der Berechnung einer 2Hop–Überdeckung in Pseudocode an und betrachten die Komplexität dieser Berechnung. Im Gegensatz zu der Veröffentlichung [CHKZ02] iterieren wir nicht über die Menge der noch zu überdeckenden Pfade, sondern über die Prioritätenwarteschlange.

CENTERGRAPH(w)

```
(1)  C_in = In(w)
(2)  C_out = Out(w)
(3)  forall (u ∈ C_in) do
(4)     forall (v ∈ C_out) do
(5)        if (v ∈ Out(u)) then
(6)           E_w = E_w ∪ {(u, v)}
(7)        fi
(8)     od
(9)  od
(10) forall (u ∈ C_in) do
(11)    if (DEGREE(u) = 0) then
(12)       C_in = C_in \ {u}
(13)    fi
(14) od
(15) forall (v ∈ C_out) do
(16)    if (DEGREE(v) = 0) then
(17)       C_out = C_out \ {v}
(18)    fi
(19) od
(20) RETURN(C_in, C_out, E_w)
```

In der Prozedur DENSEST_SUBGRAPH() brechen wir die Berechnung ab, wenn wir die Dichte für einen Teilgraphen berechnet haben, für den gilt: $\forall v_{out} \in V_w, \forall v_{in} \in V_w : (v_{out}, v_{in}) \in E_w$.

DENSESTSUBGRAPH(C_{in}, C_{out}, E_w)

```
(1)  if (|C_in| · |C_out| = |E_w|) then
(2)     fullCG = true
(3)  else
(4)     fullCG = false
(5)  fi
(6)  dmax = |E_w|/(|C_in| + |C_out|)
(7)  while (|E_w| > 0 AND fullCG = false) do
(8)     (C_in, C_out, E_w) =REMOVEMIN(C_in, C_out, E_w))
(9)     d'' = |E_w|/(|C_in| + |C_out|)
(10)    if (d'' > dmax) then
(11)       dmax = d''
(12)    fi
(13)    if (|C_in| · |C_out| = |E_w|) then
(14)       fullCG = true
(15)    fi
(16) od
(17) RETURN(dmax, C_in, C_out)
```

Schließlich geben wir nun die Berechnung einer 2Hop–Überdeckung an.

2HopLabeling(G)

```
(1)  forall (w ∈ V) do
(2)     Out(w) = {v ∈ V|P_wv ≠ ∅}
(3)     In(w) = {v ∈ V|P_wv ≠ ∅}
(4)     d = |In(w)| · |Out(w)|
(5)     Insert(Q, (w, d))
(6)  od
(7)  Out = {Out(v)|v ∈ V}
(8)  In = {In(v)|v ∈ V}
(9)  (w, d) =ExtractMax(Q)
(10) (C_in, C_out, E_w) =Centergraph(w)
(11) d', (C_in, C_out) =DensestSubgraph(C_in, C_out, E_w)
(12) if (d' < d) then
(13)    Insert(Q, (w, d'))
(14) else
(15)    Update(w, C_in, C_out)
(16) fi
(17) od
```

Die Komplexitätsbetrachtungen führen zu folgenden Ergebnissen. Eine annähernd minimale 2Hop–Überdeckung kann in Zeit $O(|V|^3)$ berechnet werden, da die Berechnung der transitiven Hülle eines Graphen mit Hilfe des Floyd–Warshall–Algorithmus diese Zeitkomplexität hat [CLR90] und die Berechnung der 2Hop–Überdeckung für die transitive Hülle ebenfalls diese Zeitkomplexität besitzt. Im ersten Schritt werden für $|V|$ Knoten die dichtesten Teilgraphen der zugehörigen Centergraphen berechnet, im zweiten Schritt wird diese Berechnung für $|V| - 1$ Knoten durchgeführt, im dritten Schritt für $|V| - 2$ Knoten, usw. Das führt zu einer $O(|V|^2)$ Gesamtlaufzeit, wobei jeder Schritt die Zeit $O(|V|)$ benötigt. Die Größe der 2Hop–Überdeckung beträgt im schlecht möglichsten Fall $O(|V|^2)$, wobei der Test, ob es eine Verbindung zwischen zwei gegebenen Knoten gibt nun in linearer Zeit $O(L)$ in Abhängigkeit von der Labelgröße L durchgeführt werden kann.

Im Rahmen dieser Arbeit berechnen wir die 2Hop–Überdeckung für den XML–Graphen und speichern die 2Hop–Labels in den Blättern des EPI als Zusatzinformation zu jedem Knoten. Auf diese Weise werden die Laufzeiten der Funktionen (7) bis (12) verbessert, mit denen nach Vorgängern, Nachfolgern gesucht bzw. die Existenz von Kanten/Pfaden validiert wird.

6.3 Elementinhaltindex

Der Elementinhaltindex (kurz: EII) dient zur effizienten Auswertung von inhaltsorientierten Suchbedingungen. Diese Suchbedingungen bestehen typischer Weise aus einer Menge von logisch verknüpften Stichwörtern. Im Rahmen dieser Arbeit orientieren wir uns an dem Abschnitt 6.1 und legen die Element– und Attributwerte des XML–Datengraphen X_D wie folgt ab.

Beispiel 6.9 In der folgenden Abbildung 6.10 ist ein invertiertes File mit den Keywords, den statistischen Informationen wie Elementfrequenz (EF) und Termfrequenz (TF) sowie Verweisen auf die Vorkommen in den entsprechenden Element– und Attributwerten durch die Angabe der zugehörigen OID angegeben.

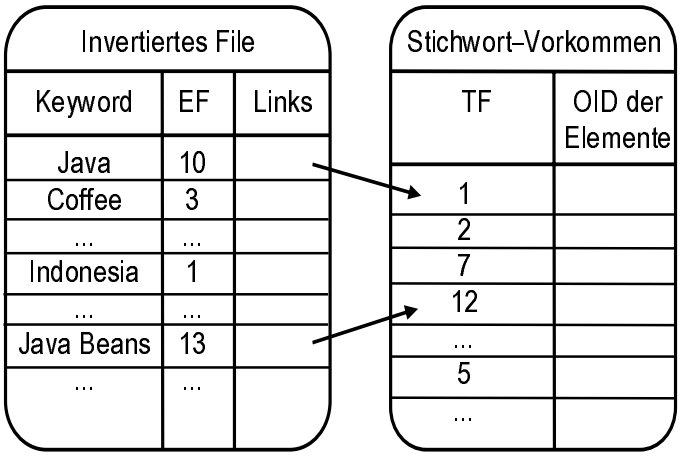


Abbildung 6.10: Invertiertes File zur Ablage von Stichwörtern (Keywords)

Der EII setzt sich aus folgenden B*–Bäumen zusammen.

- Es wird ein B*–Baum T_{Name} über die vorkommenden Element– und Attributnamen und ein B*–Baum T_{OID} über die OIDs der n–Knoten angelegt.
- Es wird ein invertiertes File über die sinntragenden Stichwörter angelegt, die in den Element– und Attributwerten vorkommen. Darin sind die statistischen Informationen, wie Termfrequenz, Elementfrequenz und inverse Elementfrequenz verankert. Das invertierte File wird als B*–Baum über die Stichwörter realisiert.

Mit Hilfe des EII können zur Auswertung von Inhaltsbedingungen gemäß der Semantikdefinition im Abschnitt 4.4.4 die Blätter des zugehörigen Operatorbaums ausgewertet und mit dem entsprechenden *score* bewertet werden. Die Funktionen *findNode(operator, condition)* (13) und *findNodeByName(name, operator, condition)* müssen zur Auswertung der gegebenen Inhaltsbedingung diese in den Operatorbaum zerlegen, die Blätter mit Hilfe des EII auswerten und die blattbasierten, lokalen Relevanzwerte zu einem Relevanzwert zusammenführen. Dies ist durch die Bereitstellung der oben genannten drei B*–Bäume effizient möglich (siehe Abschnitt 6.1) .

Kapitel 7

Implementierung

In diesem Kapitel beschreiben wir die Umsetzung der beschriebenen Konzepte in einer prototypischen Implementierung. In diesem Prototypen werden Technologien aus dem Datenbankbereich und dem Information Retrieval kombiniert. Dazu werden wir im Abschnitt 7.1 die Architektur der XXL–Suchmaschine erläutern. Im Abschnitt 7.2 geben wir einen Überblick über das gesamte Implementierungsprojekt. Im letzten Abschnitt 7.3 gehen wir auf Besonderheiten der Implementierung ein.

7.1 Architektur der XXL–Suchmaschine

Die XXL–Suchmaschine (engl.: XXL search engine) ist eine Java–basierte Client–Server–Anwendung mit entsprechenden grafischen Benutzeroberflächen (engl.: graphical user interfaces, kurz. GUI). Die Architektur der XXL–Suchmaschine (kurz: XXL–SE) ist in der Abb. 7.1 dargestellt.

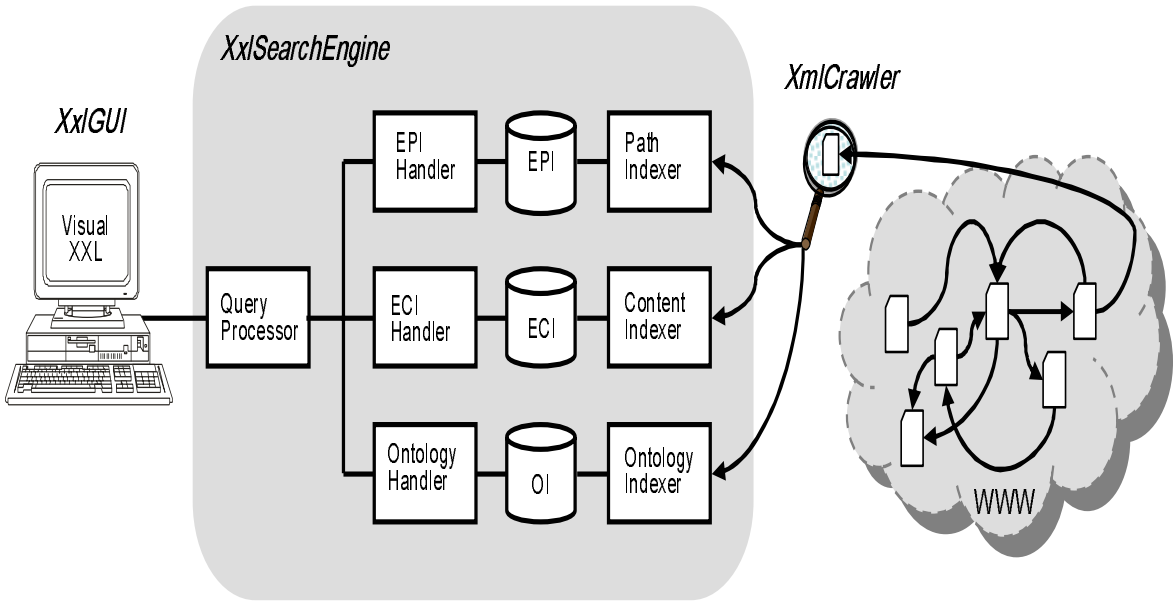


Abbildung 7.1: Architektur der XXL–Suchmaschine

Der Server stellt dabei Speicherplatz für die Indexstrukturen (EPI, ECI und OI) und Dienste zur Verarbeitung von XML–Dokumenten, zum Umgang mit den Indexstrukturen, zum Aufbau der Ontologie sowie zur Auswertung von XXL–Anfragen zur Verfügung. Daten und statistische Informationen werden in einer Oracle–Datenbank

abgelegt. Der Prototyp verwendet eine Konfigurationsdatei, in der die wichtigen Daten zum Datenbank–Account und den verwendeten Tabellennamen hinterlegt sind.

Die XXL–SE hat drei wesentliche, voneinander unabhängige Aufgaben:

1. Crawlen und Indizieren von XML–Dokumenten aus dem Web,
2. Aufbau und Pflege einer Ontologie sowie
3. Auswertung von XXL–Anfragen und Rückgabe einer Rangliste von Treffern im XML–Format an den Client.

7.2 Das Prototypsystem

In diesem Abschnitt geben wir einen Überblick über das gesamte Implementierungsprojekt.

Der Prototyp wurde in Java in der Version Java 2 Standard Edition 1.4.2 [Java] in einer Windows(XP)–Umgebung [Microsoft] implementiert. Die Datenbankfunktionalität wurde mit Hilfe von Oracle 9i Release 2 (9.2.0.1) [Oracle] realisiert. Die Java–Klassen der Prototyp–Implementierung sind wie folgt organisiert.

```
src/
    - de.unisb.xxl.Dictionary (268)
        - Dict.java
        - DictFactory.java
    - de.unisb.xxl.Ontology (2.694)
        - OntologyCrawler.java
        - OntologyDatabase.java
        - OntologyGUI.java
    - de.unisb.xxl.OracleDatabase (411)
        - Database.java
        - DatabaseFactory.java
    - de.unisb.xxl.SearchEngine (2.053)
        - SearchEngine.java
        - SerachEngineGUI.java
        - XXLSearchEngine.java
    - de.unisb.xxl.Util (173)
        - Util.java
    - de.unisb.xxl.XMLCrawler (3.640)
        - FileData.java
        - Statistics.java
        - XMLCrawler.java
        - XMLCrawlerDatabase.java
        - XMLCrawlerGUI.java
        - XMLDocHandler.java
    - de.unisb.xxl.XMLIndexes (227)
        - ContentIndex.java
        - OntologyIndex.java
        - PathIndex.java
    - de.unisb.xxl.XMLIndexes.ECI (1.807)
        - SCOREDatabase.java
        - RSVDatabase.java
    - de.unisb.xxl.XMLIndexes.EPI (3.459)
        - PPODatabase.java
        - HOPIDatabase.java
    - de.unisb.xxl.XMLIndexes.OI (156)
        - OIDatabase.java
    - de.unisb.xxl.XMLResult (1.179)
        - VarBinding.java
        - XMLResultnode.java
        - XMLResultpath.java
        - XMLResultset.java
    - de.unisb.xxl.XXLQuery (1.964)
        - XXLContentCondition.java
        - XXLQueryGraph.java
        - XXLQueryHandler.java
        - XXLQueryNode.java
        - XXLQueryRep.java
```

Der in Klammern stehende Wert entspricht in etwa der Anzahl der Sourcecode–Zeilen (engl.: lines of code) des Pakets.

Die Pakete *Dictionary, OracelDatabase* und *Util* enthalten im wesentlichen Methoden zum Umgang mit bestimmten Datenobjekten, z.B. Datenbankobjekten, Name2Integer–Abbildungen und Integer2Name–Abbildungen für die URLs und die Element–/Attributnamen.

Das Paket *Ontology* dient dem Aufbau der Ontologie auf der Basis von WordNet 1.7.1 [WordNet]. Das *XMLCrawler*–Paket enthält alle notwendigen Methoden zum Einlesen von XML–Dokumenten aus dem Web oder Dokumentarchiven. Das Paket

XXLQuery dient zum Parsen und Zerlegen einer XXL–Anfrage in Textform in seine Bestandteile, wobei außerdem die zugehörigen Pfadbedingungsgraphen erzeugt werden. Das Paket *SearchEngine* dient der Verarbeitung von XXL–Anfragen und der Rückgabe gewünschter Ergebnisse im XML–Format. Dazu wird das Paket *XMLResult* benötigt, das die Methoden zum Umgang mit Zwischenergebnissen und Ergebnissen der XXL–Anfrageverarbeitung bereitstellt. Schließlich definiert das Paket *XMLIndexes* die Schnittstellen (engl.: Interfaces) zu den Indexstrukturen, wobei die Teilpakete entsprechende Implementierungen enthalten.

Darüber hinaus gibt es eine Konfigurationsdatei, die die wesentlichen Angaben zum Datenbank–Account und zu den verwendeten Tabellennamen enthält. Auf diese Weise ist es sehr einfach, die bestehende Implementierung auf einem Rechner mit Java und Oracle zum Einsatz zu bringen.

```
// Oracle Database: USER ACCOUNT ================================= //
sUser   = "xml"
sPasswd = "xml"
sHost   = "mpino5303"
sServiceName = "(SERVER=DEDICATED)(SERVICE_NAME=db4xml.mpino5303)"

// Oracle Database: TABLE NAMES ================================== //
UrlLength = "200"
NameLength = "200"
UrlTable  = "URLS"
NameTable = "NAMES"
NodeTable = "NODES"
EdgeTable = "EDGES"
LinkTable = "LINKS"
ContentTable = "CONTENTS"
NestedcontentTable = "NESTEDCONTENTS"

ConceptTable = "CONCEPTS"
TermTable = "TERMS"
RelationshipTable = "RELATIONSHIPS"

// DEBUG MODE ==================================================== //
Verbose = "false"

// XXL-SE Mode: 0 - standard, 1 - demo, 2 - benchmark
mode = "1"
```

In dieser Implementierung füllen wir die Ontologie mit den Konzepten, Wörtern und Beschreibungen aus WordNet. Wir verwenden die Java WordNet Library [JWNL], um auf die Daten von WordNet zuzugreifen.

7.3 Spezielle Aspekte der Implementierung

In diesem Abschnitt wollen wir auf Besonderheiten der Implementierung eingehen. Die XXL–Suchmaschine unterstützt die folgenden drei voneinander unabhängigen Aufgaben:

1. *Crawlen & Indizieren:* Zerlegen von XML–Dokumenten in die Namen und Werte von Elementen und Attributen, Speichern der Namen und Werte von Elemen-

ten und Attributen in geeigneten Datenbanktabellen, Speichern der Kanten und Links in geeigneten Datenbanktabellen, Erzeugen geeigneter Indexstrukturen,

2. *Ontologieaufbau:* Aufbauen und Pflegen einer Ontologie bestehend aus Konzepten, Wörtern und semantischen, gewichteten Beziehungen, die in geeigneten Datenbanktabellen abgelegt sind, und
3. *XXL–Anfrageverarbeitung:* Auswerten von XXL–Anfragen unter Verwendung der Daten in den Datenbanktabellen und der verfügbaren Indexstrukturen und Zurückgeben einer Rangliste von Treffern im XML–Format gemäß der gegebenen Select–Klausel.

Typisch für diese drei Aufgaben der XXL–Suchmaschine ist die Bereitstellung einer GUI und die Verwendung von Datenbanktabellen. In den folgenden drei Abschnitten werden wir uns diesen drei Aufgaben zuwenden und auf die Besonderheiten im Umgang mit der Datenbank eingehen.

7.4 Crawlen & Indizieren

1. Die Daten

Der XML–Crawler parst und zerlegt XML–Dokumente aus dem Web oder aus lokalen Dokumentarchiven in ihre Bestandteile. Dabei werden die Namen und Werte von Elementen und Attributen in geeigneten Datenbanktabellen gespeichert. In dieser Implementierung gibt es die folgenden sechs Datenbanktabellen und zwei Indextabellen.

URLS	(urlid, url, lastmodified)
NAMES	(nid, name)
NODES	(oid, urlid, nid, pre, post, num, type, depth)
EDGES	(oid1, urlid1, nid1, oid2, urlid2, nid2)
LINKS	(oid1, urlid1, nid1, oid2, urlid2, nid2)
CONTENTS	(oid, urlid, nid, content, type)
LIN	(oid1, oid2)
LOUT	(oid1, urlid1, nid1, oid2, urlid2, nid2)

Die unterstrichenen Attribute sind die Primärschlüssel der jeweiligen Tabelle. Jedes Element/Attribut wird als Knoten des XML–Graphen mit einem eindeutigen Objektbezeichner (oid) versehen. Kanten und Links werden durch Verwendung der OIDs gespeichert. Jede URL und jeder vorkommende Element–/Attributnamen wird auf ein Integer (urlid bzw. nid) abgebildet und in der Tabelle URLS bzw. NAMES abgelegt.

Für jeden n–Knoten im XML–Graphen wird in der Tabelle NODES ein Eintrag mit einer eindeutigen OID angelegt, wobei für die zugehörige Url und den zugehörigen Namen die entsprechenden Integerwerte verwendet werden. Die Attribute 'pre' und 'post' enthalten die PrePostOrder–Kodierung, das Attribut 'num' nummeriert die Elemente gleichen Namens und gleicher Tiefe (depth) und 'type' speichert ggf. den zugehörigen Attributtyp.

In den Tabellen EDGES und LINKS werden die Kanten und die Links (ID–Referenzen, XLinks, XPointer) zwischen den einzelnen Knoten abgespeichert.

Die Tabelle CONTENTS enthält die Element– und Attributwerte. Dabei wird die OID des zugehörigen n–Knotens aus der Relation NODES verwendet.

Die Tabellen LIN und LOUT stellen einen Teil des Elementpfadindizes (EPI) zur Verfügung. Sie enthalten die 2Hop–Labels der 2Hop–Überdeckung, die durch ein externes Tool berechnet wird.

Zusätzlich werden die Tabellen analysiert und eine Reihe von Oracle–Indizes angelegt, um den Zugriff auf die Daten zu beschleunigen. Neben den Indizes auf den Primärschlüsseln erzeugen wir für folgende Tabellen und Attribute weitere Indizes:

NAMES	(name)
NODES	(nid)
NODES	(urlid, pre)
NODES	(urlid, post)
EDGES	(oid2, oid1)
CONTENTS	(nid)

Darüber hinaus wird für das Attribut 'content' der Tabelle CONTENTS ein Oracle Text Index in Form eines invertierten Files angelegt. Hierbei weisen wir auf zwei Besonderheiten hin. Ersten, wir speichern die vollständigen Werte der Elemente, die sich aus den eigenen Elementwerten und den Elementwerten der Kinder zusammensetzen. Zweitens, wir benötigen dazu einen ausreichend großen Datentypen, z.B. CLOB (character large object). Die Verwendung dieses Textindizes wird im Abschnitt 7.6 ausführlich erläutert.

2. Der Prozess

Der XMLCrawler lässt sich über eine eigens dafür implementierte GUI steuern (siehe Abb. 7.2). Damit ist der Benutzer in der Lage, ein oder mehrere XML–Dokumente als Ausgangspunkt des XMLCrawlers anzugeben und den Prozess zu starten. Dabei entscheidet er, ob er die bestehenden Daten beibehält und aktualisiert oder alle Tabellen löscht und neu beginnt.

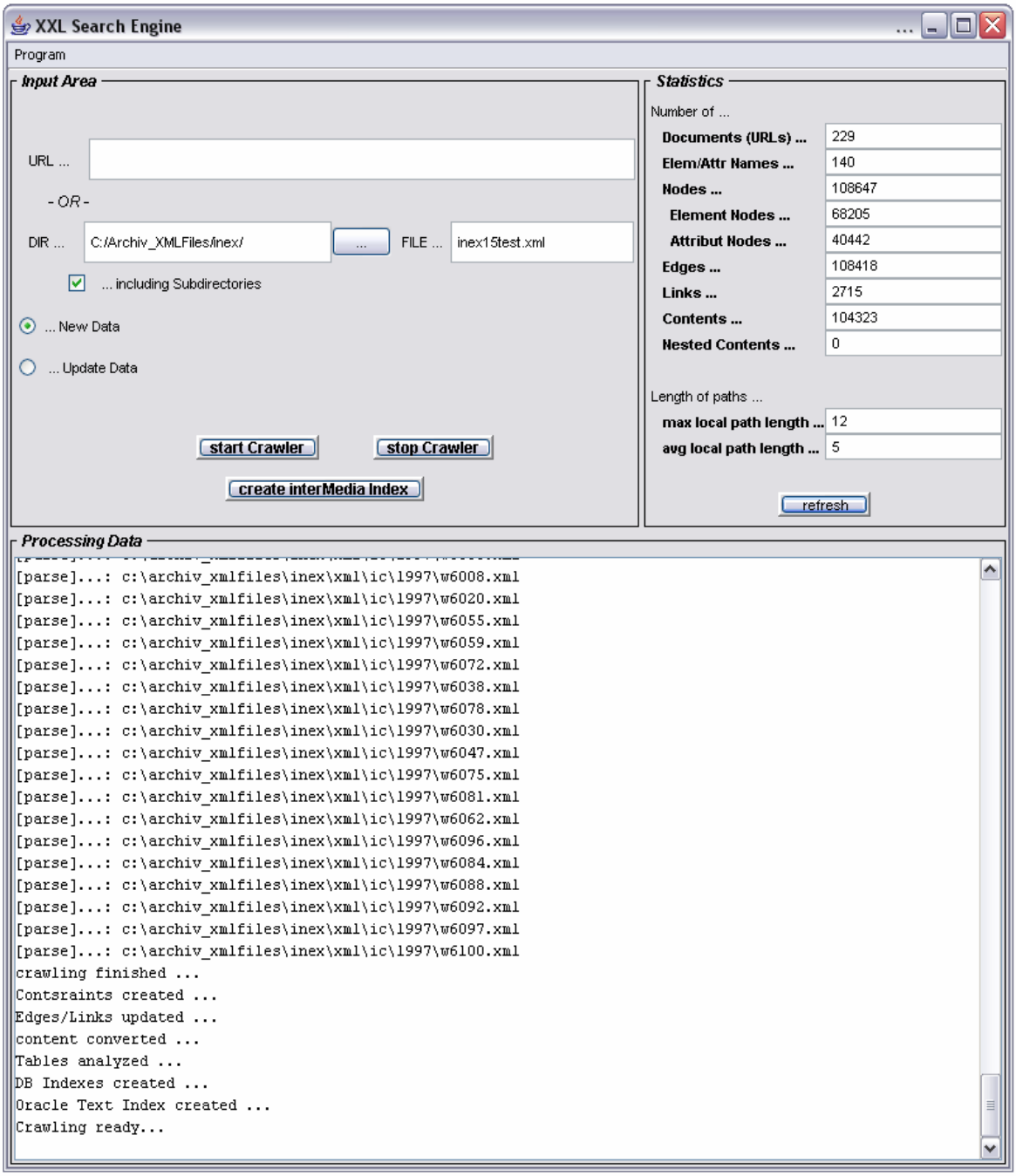


Abbildung 7.2: XmlCrawlerGUI

Der XML–Crawler arbeitet im Multithread–Verfahren, wobei für jedes XML–Dokument, das verarbeitet werden soll, ein eigener Thread gestartet wird. Die maximale Anzahl paralleler Threads ist limitiert. Werden in einem XML–Dokument Links gefunden, wo wird die Information des Startelements zwischengespeichert und die URL des Zieldokuments in die Liste der noch zu verarbeitenden XML-Dokumente aufgenommen.

Der Prozess kann auf zwei Wegen zu einem Ende kommen. Wird der Prozess durch den Benutzer gestoppt, dann beendet der XML–Crawler alle in Arbeit befindlichen Threads und hält an. Auf diese Weise wird sichergestellt, dass die Datenbank konsistente Informationen enthält. Ist die Liste der zu verarbeitenden Dokumente leer und sind alle Threads ohne Aufgabe, dann wird der Prozess ebenfalls beendet.

7.5 Ontologieaufbau

1. Die Daten

Die Konzepte und semantischen Beziehungen zwischen ihnen werden im Rahmen dieser Arbeit in den folgenden drei Tabellen abgelegt. Die unterstrichenen Attribute entsprechen den Primärschlüsseln der einzelnen Tabellen.

CONCEPTS	(cid, description, freq, query)
TERMS	(cid, term)
RELATIONSHIPS	(cid1, cid2, type, freq, weight, formula)

2. Der Prozess

Die Ontologie wird über eine eigens implementierte GUI verwaltet (siehe Abb. 7.3). Dabei besteht zum einen die Möglichkeit, die Ontologie mit Konzepten aus WordNet zu füllen, und zum anderen die Möglichkeit, eine eigene Ontologie aufzubauen.

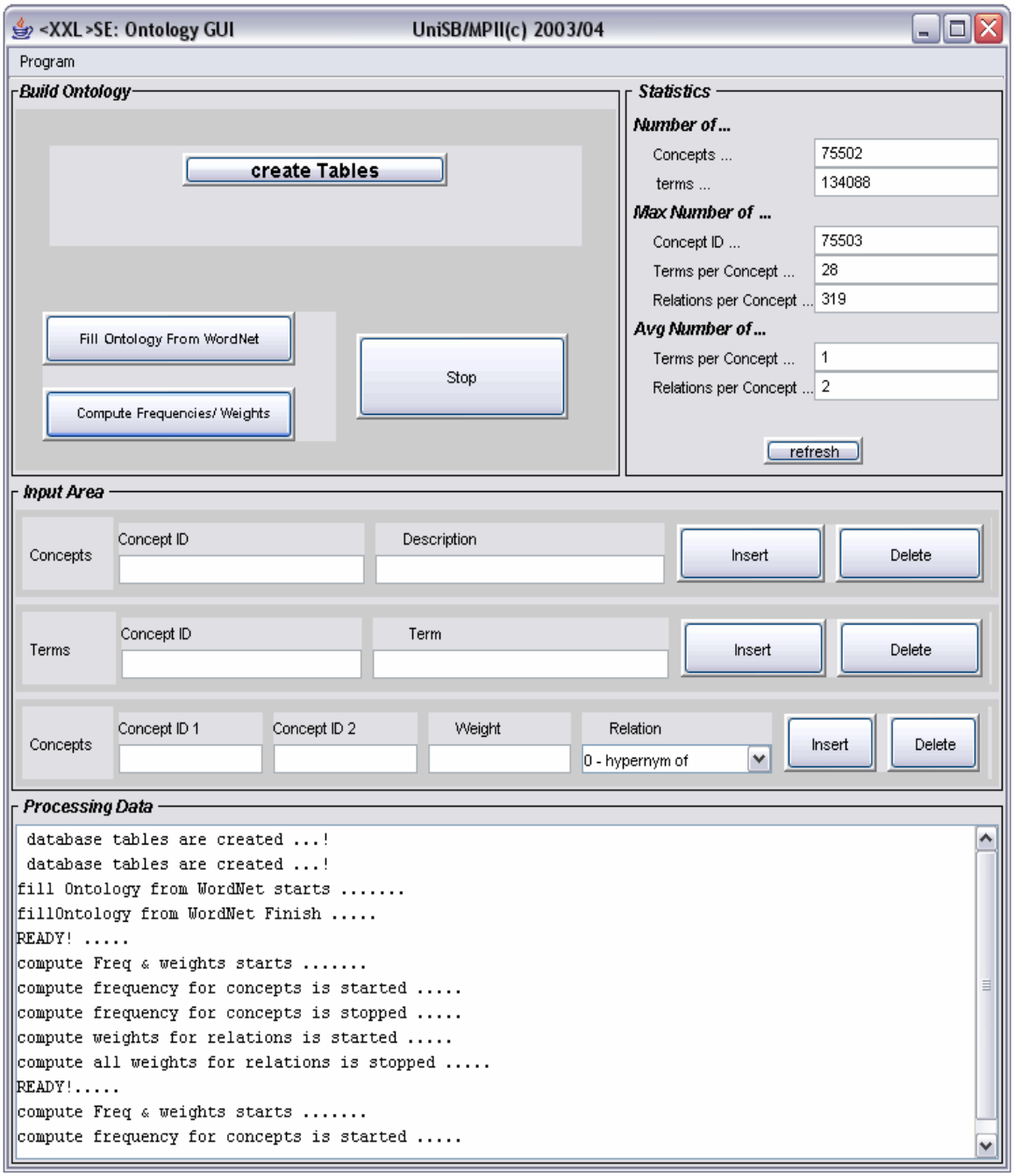


Abbildung 7.3: XmlOntologyGUI

Die Berechnung der Häufigkeiten (freq) von Konzepten und Relationships basiert auf Anfragen an die Websuchmaschine Google nach der Anzahl der Dokumente, die die Stichwörter des zugehörigen Konzeptes umfassen. Die Berechnung der Gewichte für die Relationships basiert auf der Korrelation der Konzepte, wobei hier der Dice–Koeffizient verwendet wird. Hierbei gibt es eine technische Besonderheit, die sich aus der Limitierung der Websuchmaschinen ergibt. Standardwebsuchmaschinen, wie z.B. Google, verarbeiten höchstens 10 Stichwörter in einer Anfrage. Dementsprechend haben wir die Kontexte der Konzepte auf 5 Wörter begrenzt.

Da die Ontologie sehr groß sein kann, kann die Berechnung der Frequenzen und Gewichte lange dauern. Aus diesem Grund besteht die Möglichkeit, die Berechnung zu unterbrechen und später fortzusetzen.

7.6 XXL–Anfrageverarbeitung

Die XXL–Anfragen werden von Clients durch Verwendung einer speziellen grafischen Benutzeroberfläche an den Server der XXL–Suchmaschine gerichtet (siehe Abb. 7.4).

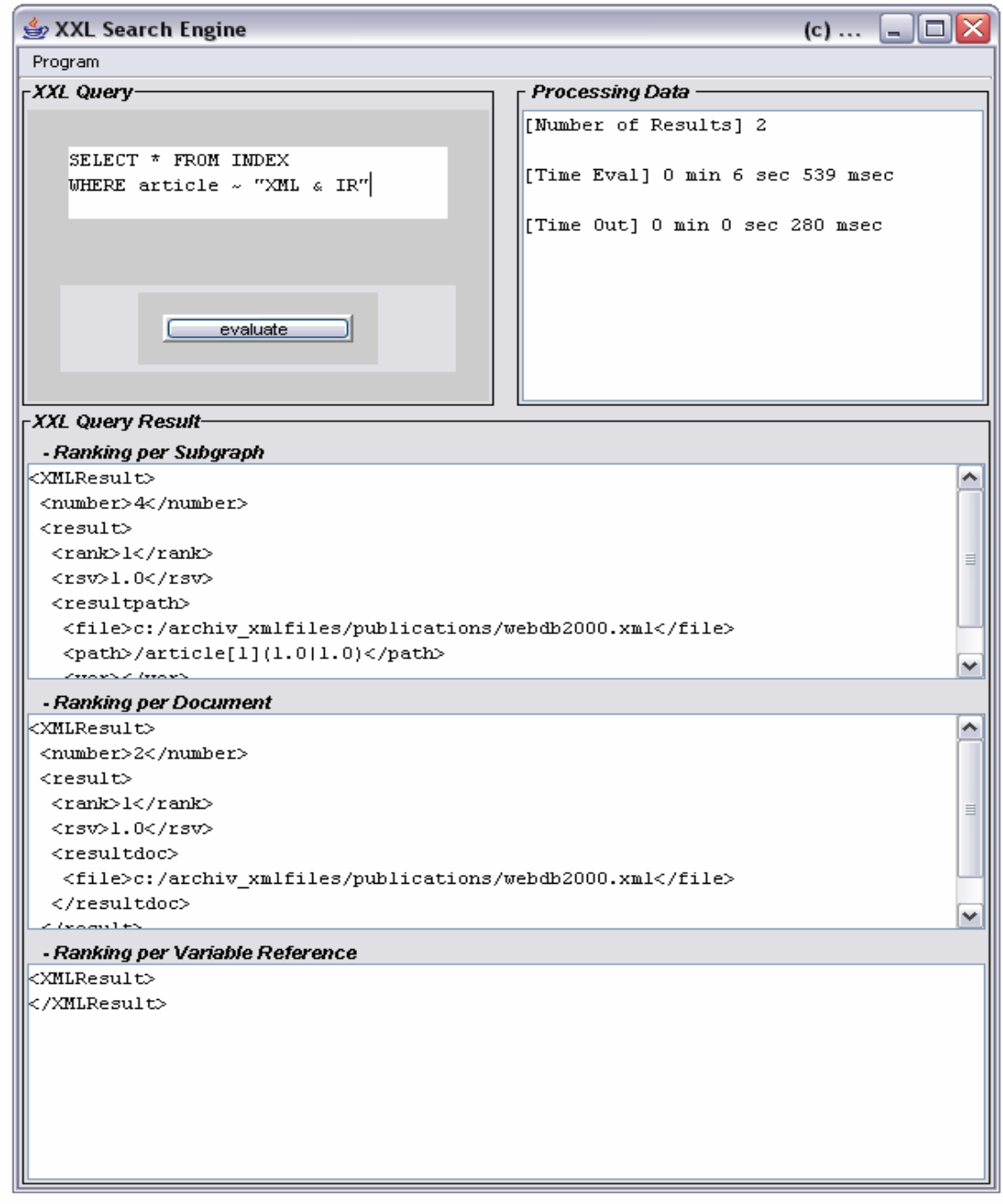


Abbildung 7.4: XxlQueryGUI

Dieser wertet sie aus und liefert das Ergebnis im XML–Format an den Client zurück. Das Ergebnis wird dann in der Benutzeroberfläche angezeigt.

In der aktuellen Version der Implementierung werden die im Kapitel 5 eingeführten Methoden entsprechend der im Kapitel 6 definierten Indexstrukturen durch geeignete Anfragen an die Oracle–Tabellen realisiert. Wir gehen hier auf die wichtigsten Methoden ein und erläutern die verwendeten SQL–Anfragen.

1. Der Elementpfadindex (EPI)

Der Elementpfadindex (EPI) stellt sowohl die PrePostOrder–Kodierung für jedes XML–Dokument in der Tabelle NODES als auch die 2Hop–Überdeckung für den gesamten XML–Graphen in den Tabellen LIN, LOUT zur Verfügung. Dem entsprechend unterstützen wir beide Methoden, um die Verbindungen zwischen Elementen effizient auszunutzen. In Abhängigkeit der vorliegenden XML–Dokumente kann wahlweise eine der Implementierungen verwendet werden.

In `PPODatabase.java` werden Nachfolger und Vorgänger sowie der Pfadtest nur innerhalb eines XML–Dokumentes auf der Basis der dokumentinternen PrePostOrder–Kodierung realisiert. Wir geben die SQL–Anfragen an, die in den Methoden 6, 7, 8, 14 und 15 (siehe Abschnitt 5.1) in dieser Implementierung verwendet werden.

6. *findDescendants(oid)* (links) und 7. *findDescendantsByName(oid, nid)* (rechts)

```
SELECT n2.oid, n2.urlid, n2.nid
FROM NODES n1, NODES n2
WHERE n1.oid=<oid>

  AND n1.urlid=n2.urlid
  AND n1.pre<=n2.pre
  AND n1.post=>n2.post;
```

```
SELECT n2.oid, n2.urlid, n2.nid
FROM NODES n1, NODES n2
WHERE n1.oid=<oid>
  AND n2.nid=<nid>
  AND n1.urlid=n2.urlid
  AND n1.pre<=n2.pre
  AND n1.post=>n2.post;
```

14. *findAncestors(oid)* (links) und 15. *findAncestorsByName(oid, nid)* (rechts)

```
SELECT n1.oid, n1.urlid, n1.nid
FROM NODES n1, NODES n2
WHERE n2.oid=<oid>

  AND n1.urlid=n2.urlid
  AND n1.pre<=n2.pre
  AND n1.post=>n2.post;
```

```
SELECT n1.oid, n1.urlid, n1.nid
FROM NODES n1, NODES n2
WHERE n2.oid=<oid>
  AND n1.nid=<nid>
  AND n1.urlid=n2.urlid
  AND n1.pre<=n2.pre
  AND n1.post=>n2.post;
```

8. *testPath(oid1,oid2)*

```
SELECT count(*)
FROM NODES n1, NODES n2
WHERE n1.oid=<oid1>
  AND n2.oid=<oid2>
  AND n1.urlid=n2.urlid
  AND n1.pre<=n2.pre AND n1.post=>n2.post;
```

In `HOPIDatabase.java` werden Nachfolger und Vorgänger sowie der Pfadtest auf der Basis der 2Hop–Überdeckung für den gesamten XML–Graphen realisiert. Wir geben die SQL–Anfragen an, die in den Methoden 6, 7, 8, 14 und 15 (siehe Abschnitt 5.1) in dieser Implementierung verwendet werden.

6. *findDescendants(oid)* (links) und 7. *findDescendantsByName(oid, nid)* (rechts)

```
SELECT in.oid2
FROM LIN in, LOUT out
WHERE out.oid1=<oid>
  AND out.oid2=in.oid1
```

```
SELECT in.oid2
FROM LIN in, LOUT out, NODES n
WHERE out.oid1=<oid>
  AND out.oid2=in.oid1
  AND in.oid2=n.oid
  AND n.nid=<nid>
```

14. *findAncestors(oid)* (links) und 15. *findAncestorsByName(oid, nid)* (rechts)

```
SELECT out.oid1
FROM LIN in, LOUT out
WHERE in.oid2=<oid>
  AND out.oid2=in.oid1
```

```
SELECT out.oid1
FROM LIN in, LOUT out, NAME
WHERE in.oid2=<oid>
  AND out.oid2=in.oid1
  AND out.oid1=n.oid
  AND n.nid=<nid>
```

8. *testPath(oid1,oid2)*

```
SELECT count(*)
FROM LIN in, LOUT out
WHERE out.oid1=<oid1>
  AND in.oid2=<oid2>
  AND out.oid2=in.oid1
```

2. Der Elementinhaltindex (EII)

Der Elementinhaltindex (EII) verwendet den Oracle Text Index zur Auswertung von Inhaltsbedingungen gemäß der Semantikdefinition in Abschnitt 4.4.4. Für jeden Term t jeden Blattes des Operatorbaums wird eine Anfrage der Form

```
SELECT score(1), oid, urlid, nid, value
FROM CONTENTS
WHERE CONTAINS(content, 't', 1)>0
ORDER BY score(1) desc;
```

gestellt. Es werden hierbei alle Element– und Attributwerte berücksichtigt, in denen der gegebene Term exakt mindestens einmal vorkommt. Die SCORE–Funktion gibt einen *TF/IDF*–basierten Relevanzwert zwischen 0 und 100 zurück. Da wir in der Tabelle CONTENTS nur Elemente und Attribute haben, aber keine vollständigen XML–Dokumente, entspricht dieser Wert dividiert durch 100 dem *tf/ief*–Wert, wie wir ihn im Abschnitt 4.4.4.1 definiert haben.

In der Klasse `SCOREDatabase.java` verwenden wir den von Oracle zurückgegebenen Relevanzwert zur Kalkulation des Gewichtes des Terms t. In der Klasse `RSVDatabase.java` ermitteln wir die statistischen Werte wie Elementfrequenz, Termfrequenz zur Laufzeit selbst und verwenden die obige Anfrage zur Auswahl der relevanten Element– und Attributwerte. Der Relevanzwert basiert auf den eigenen statistischen Daten.

Kapitel 8

Evaluation

In den Kapiteln 2 bis 6 haben wir ein Konzept zur ontologiebasierten Ähnlichkeitssuche in umfangreichen XML–Daten entwickelt. Im Kapitel 7 haben wir die Prototyp–Implementierung zur praktischen Anwendung dieses Konzepts vorgestellt. In diesem Kapitel steht die Evaluation unseres Konzepts hinsichtlich der Effektivität und der Effizienz der Informationssuche in XML–Daten im Vordergrund. Dazu haben wir zwei Testszenarios aufgestellt.

- *Experiment 1: "Proof of Concept"*
  Anhand einer sehr kleinen Auswahl von XML–Dokumenten zeigen wir die Effektivität der ontologiebasierten Ähnlichkeitssuche in XML–Daten mit Hilfe der XXL–Suchmaschine.
- *Experiment 2: "Benchmarktest"*
  Tests im Rahmen der "Initiative for the Evaluation of XML Retrieval" (INEX) demonstrieren die Effizienz der XXL–Suchmaschine im Umgang mit einer großen Menge von XML–Dokumenten und unterstreichen die Effektivität der ontologiebasierten Ähnlichkeitssuche in diesen XML–Daten.

Die Experimente wurden auf einem Notebook durchgeführt, das über einen Intel Mobile Pentium 1600 MHz Prozessor und 1 Gigabyte Hauptspeicher (RAM) verfügt. Im Rahmen einer Windows XP–Umgebung [WinXP] stehen Oracle 9i Release 9.2.0.1 [Oracle] und Java 2 Standard Edition 1.4.2 [Java] zur Verfügung.

8.1 Experiment 1: "Proof of Concept"

Mit diesem Experiment zeigen wir die Effektivität der ontologiebasierten Ähnlichkeitssuche in XML–Dokumenten. Die XML–Anfragesprache XXL verfügt über zwei besondere Fähigkeiten, die anhand von Beispielanfragen demonstriert werden sollen.

1. Mit Hilfe einer XXL–Anfrage können Informationen gefunden werden, die in verschiedenen Elementen/Attributen verschiedener XML–Dokumente enthalten sind.
2. Mit Hilfe einer XXL–Anfrage können semantisch ähnliche Namen und semantisch ähnliche Werte von Elementen/Attributen mit Hilfe einer Ontologie gefunden werden.

8.1.1 Die Daten

Das erste Experiment basiert auf dem Beispiel aus Kapitel 1. Dieses Beispiel dient als Vorlage für vier miteinander durch XLinks, XPointer und ID–Referenzen verknüpfte XML–Dokumente *reference.xml, arts.xml, macbeth.xml* und *hamlet.xml*. Die Shakespeare–Dramen "Macbeth" und "Hamlet" enthalten nur die für uns interessanten Ausschnitte.

In der folgenden Abbildung 8.1 sehen wir einen Ausschnitt aus dem zugehörigen XML–Graphen dargestellt. In jedem Knoten steht die OID in eckigen Klammern und dahinter der Name oder Wert des entsprechenden Elements bzw. Attributs. Außerdem haben wir für die Knoten 6, 7 und 22 die zugehörigen 2Hop–Labels angegeben.

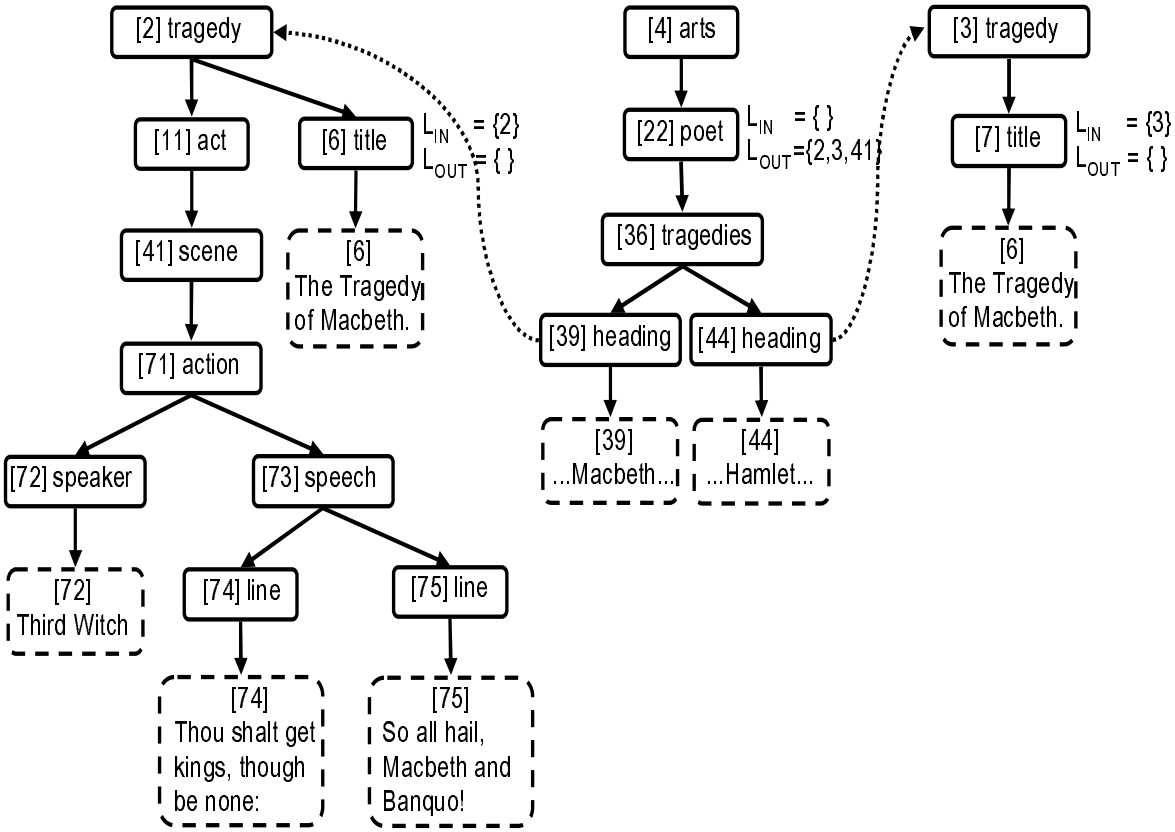


Abbildung 8.1: XML–Graph zum Experiment 1

Zu Beginn des Experiments werden die oben genannten XML–Dokumente mit Hilfe des XML–Crawlers in ihre Bestandteile zerlegt und in den entsprechenden Tabellen gespeichert. Die Anzahl der Einträge in jeder Tabelle sind in der folgenden Übersicht zusammengestellt.

DB–Tabelle	Anzahl	DB–Tabelle	Anzahl	DB–Tabelle	Anzahl	DB–Tabelle	Anzahl
URLS	4	NODES	89	EDGES	85	LIN	102
NAMES	22	CONTENTS	86	LINKS	6	LOUT	51

Die Zerlegung der gegebenen XML–Dokumente, die Speicherung der Elemente und Attribute in den entsprechenden Tabellen der Datenbank sowie die Berechnung der Indizes dauert etwa 4 Sekunden. Für die Berechnung der 2Hop–Überdeckung werden insgesamt 1,6 Sekunden benötigt. Davon entfallen auf die Berechnung der transitiven Hülle für diesen XML–Graphen, die 646 Verbindungen umfasst, 0,49 Sekunden. Die maximale in einem XML–Dokument auftretende Tiefe beträgt 5.

8.1.2 Die Ergebnisrückgabe

Die XXL–Suchmaschine berechnet als Ergebnis einer XXL–Anfrage eine absteigend sortierte Rangliste von Resultatgraphen (siehe Abschnitt 4.4). Dabei genügt jeder Resultatgraph einer gegebenen Variablenbelegung. Solch ein Resultatgraph setzt sich aus je einem relevanten Pfad pro XXL–Suchbedingung der gegebenen XXL–Anfrage zusammen. Zu dem Ergebnis gehören die Anzahl der Treffer, der Rang und der Relevanzwert des Ergebnisses, sowie die Laufzeit zur Anfrageauswertung und die Laufzeit zur Ergebniszusammenstellung. Typischer Weise werden die Werte von Elementen und Attributen aufgrund ihrer Größe nicht mit ausgegeben.

Das Rückgabeergebnis wird aus dieser Rangliste von Resultatgraphen erstellt und hängt dabei maßgeblich von der Select–Klausel der gegebenen XXL–Anfrage ab. Betrachten wir die folgenden drei XXL–Anfragen, die alle drei nach den Titeln von Tragödien fragen aber verschiedene Teile des Gesamtergebnisses zurückgeben:

```
XXL-Anfrage 1a:                XXL-Anfrage 1b:                XXL-Anfrage 1c:
 SELECT *                       SELECT URLS                    SELECT $T
 FROM   INDEX                   FROM   INDEX                   FROM   INDEX
 WHERE  tragedy/title AS $T     WHERE  tragedy/title AS $T     WHERE  tragedy/title AS $T
```

Steht in der Select–Klausel ein '*', dann werden die Resultatgraphen unverändert in einem einfachen XML–Format ausgegeben. Zu einem Ergebnis wird zu jeder XXL–Suchbedingung die Belegung der vorkommenden Variablen, der relevante Pfad und das Ursprungsdokument angegeben. Die Angabe des zugehörigen XML–Dokumentes bezieht sich immer auf den ersten Knoten des relevanten Pfades. Zu jedem relevanten Knoten werden außerdem die beiden lokalen Relevanzwerte π und π' in runden Klammern und die lokale Nummerierung gleicher Elementnamen in eckigen Klammern angegeben. Für die XXL–Anfrage 1a ergibt sich dann folgende Ergebnisrückgabe.

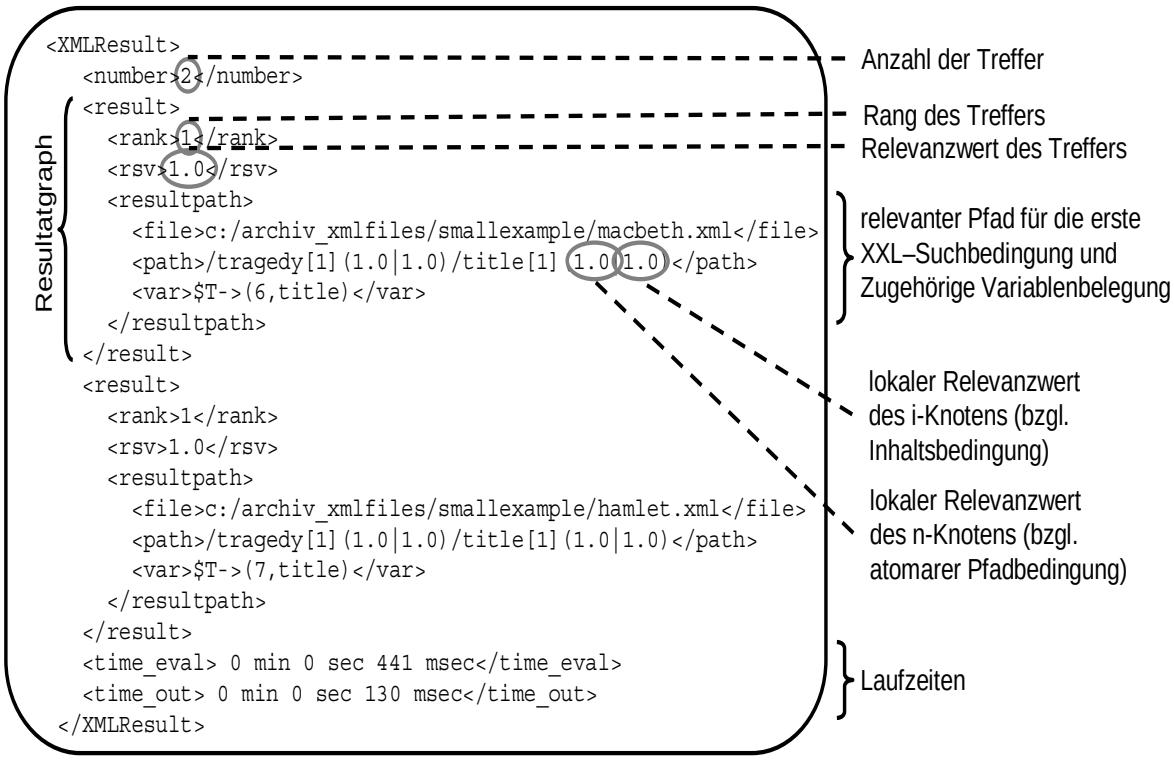


Abbildung 8.2: Vollständige Ergebnisrückgabe gemäß XXL–Anfrage 1a

Steht in der Select–Klausel das reservierte Wort 'URLS', dann werden aus den Resultatgraphen die zugehörigen Urls extrahiert und ausgegeben. Die Rangfolge bleibt unverändert, allerdings wird jedes Dokument nur einmal durch die Url mit dem höchsten Relevanzwert repräsentiert. Für die XXL–Anfrage 1b ergibt sich dann folgende Ergebnisrückgabe.

```
<XMLResult>
  <number>2</number>
  <result>                                   Resultatdokument
    <rank>1</rank>
    <rsv>1.0</rsv>
    <resultdoc>
      <file>c:/archiv_xmlfiles/smallexample/macbeth.xml</file>
    </resultdoc>
  </result>
  <result>
    <rank>1</rank>
    <rsv>1.0</rsv>
    <resultdoc>
      <file>c:/archiv_xmlfiles/smallexample/hamlet.xml</file>
    </resultdoc>
  </result>
  <time_eval> 0 min 0 sec 441 msec</time_eval>
  <time_out> 0 min 0 sec 90 msec</time_out>
</XMLResult>
```

Abbildung 8.3: Rückgabe der Ergebnisdokumente gemäß XXL–Anfrage 1b

Steht in der Select–Klausel ein Variablenname (es können auch mehrere sein), dann wird aus der Rangliste der Resultatgraphen die Belegung der Variablen extrahiert und mit den entsprechenden n–Knoten ausgegeben. Die Rangfolge und die Anzahl der Ergebnisse bleibt dabei unverändert. Für die XXL–Anfrage 1c ergibt sich dann folgende Ergebnisrückgabe.

```
<XMLResult>
  <number>2</number>
  <result>                                   Resultatvariable
    <rank>1</rank>
    <rsv>1.0</rsv>
    <resultvar>
      <file>c:/archiv_xmlfiles/smallexample/macbeth.xml</file>
      <var>$T->(6,title)</var>
    </resultvar>
  </result>
  <result>
    <rank>1</rank>
    <rsv>1.0</rsv>
    <resultvar>
      <file>c:/archiv_xmlfiles/smallexample/hamlet.xml</file>
      <var>$T->(7,title)</var>
    </resultvar>
  </result>
  <time_eval> 0 min 0 sec 441 msec</time_eval>
  <time_out> 0 min 0 sec 89 msec</time_out>
</XMLResult>
```

Abbildung 8.4: Rückgabe der Variablenbelegungen gemäß XXL–Anfrage 1c

8.1.3 Finden von verteilten Informationen

Mit Hilfe von XXL–Anfragen ist es möglich, Informationen zu suchen, die in verschiedenen Elementen eines oder auch mehrerer XML–Dokumente enthalten sind. Die folgende XXL–Anfrage sucht nach Werken (Titeln) von Theaterstücken von Shakespeare:

```
SELECT *
FROM   INDEX
WHERE  arts/poet AS $P
  AND  $P/lastname = "Shakespeare"
  AND  $P/#/title AS $T
```

In Anlehnung an den XML–Graphen in Abb. 8.1 wird deutlich, dass die Informationen zum Dichter und die Informationen über die Werke in verschiedenen XML–Dokumenten zu finden sind. Mit Hilfe der 2Hop–Überdeckung ist es möglich, effizient die Verbindung zwischen dem Element `poet` im XML–Dokument *arts.xml* und den Elementen `title` in den XML–Dokumenten *macbeth.xml* und *hamlet.xml* zu finden und auf diese Weise folgendes Ergebnis zu produzieren:

```
<XMLResult>
  <number>2</number>
  <result>
    <rank>1</rank>
    <rsv>1.0</rsv>
    <resultpath>
      <file>c:/archiv_xmlfiles/smallexample/arts.xml</file>
      <path>/arts[1](1.0|1.0)/poet[1](1.0|1.0)</path>
      <var>$P->(22,poet)</var>
    </resultpath>
    <resultpath>
      <file>c:/archiv_xmlfiles/smallexample/arts.xml</file>
      <path>/poet[1](1.0|1.0)/lastname[1](1.0|1.0)</path>
      <var>$P->(22,poet)</var>
    </resultpath>
    <resultpath>
      <file>c:/archiv_xmlfiles/smallexample/arts.xml</file>
      <path>/poet[1](1.0|1.0)/#/title[1](1.0|1.0)</path>
      <var>$P->(22,poet)</var>
      <var>$T->(6,title)</var>
    </resultpath>
  </result>
  <result>
    <rank>1</rank>
    <rsv>1.0</rsv>
    <resultpath>
      <file>c:/archiv_xmlfiles/smallexample/arts.xml</file>
      <path>/arts[1](1.0|1.0)/poet[1](1.0|1.0)</path>
      <var>$P->(22,poet)</var>
    </resultpath>
    <resultpath>
      <file>c:/archiv_xmlfiles/smallexample/arts.xml</file>
      <path>/poet[1](1.0|1.0)/lastname[1](1.0|1.0)</path>
      <var>$P->(22,poet)</var>
```

```
    </resultpath>
    <resultpath>
      <file>c:/archiv_xmlfiles/smallexample/arts.xml</file>
      <path>/poet[1](1.0|1.0)/#/title[1](1.0|1.0)</path>
      <var>$P->(22,poet)</var>
      <var>$T->(7,title)</var>
    </resultpath>
  </result>
  <time_eval> 0 min 0 sec 566 msec</time_eval>
  <time_out> 0 min 0 sec 155 msec</time_out>
</XMLResult>
```

Mit Hilfe der XXL–Suchmaschine können also Informationen gefunden werden, die insbesondere in Elementen verschiedener XML–Dokumente enthalten sind.

8.1.4 Finden von semantisch ähnlichen Informationen

Mit Hilfe von XXL–Anfragen ist es möglich, nach semantisch ähnlichen Informationen mit Hilfe einer Ontologie zu fragen. Die folgende XXL–Anfrage sucht nach Szenen in Theaterstücken von Shakespeare, in denen eine Frau über Macbeth als eine Führungsperson spricht:

```
SELECT *
FROM   INDEX
WHERE  ~play/#/scene AS $S
  AND  $S/action AS $A
  AND  $A/speaker ~ "woman"
  AND  $A/speech ~ "Macbeth & leader"
```

In dieser XXL–Anfrage sind Ähnlichkeitsbedingungen an die Strukturdaten durch '∼`play`' und an die Inhaltsdaten durch '∼ "`woman`"' bzw. '∼ "`Macbeth & leader`"' formuliert worden.

Wir nehmen an, dass wir obige XXL–Anfrage mit Hilfe der Ontologie wie folgt expandieren können:

```
play   -> play | tragedy | history | comedy
woman  -> woman | women | lady | ladies | witch | witches
leader -> leader | thane | baron | king | kings
```

Ohne ontologiebasierte Anfrageerweiterung würden wir in den gegebenen Daten keine relevanten Informationen finden. Mit Hilfe der Ontologie finden wir folgenden Treffer:

```
<XMLResult>
  <number>1</number>
  <result>
    <rank>1</rank>
    <rsv>0.12</rsv>
    <resultpath>
      <file>c:/archiv_xmlfiles/smallexample/macbeth.xml</file>
      <path>/tragedy[1](0.9|1.0)/#/scene[1](1.0|1.0)</path>
      <var>$S->(41,scene)</var>
```

```
      </resultpath>
      <resultpath>
        <file>c:/archiv_xmlfiles/smallexample/macbeth.xml</file>
        <path>/scene[1](1.0|1.0)/action[1](1.0|1.0)</path>
        <var>$S->(41,scene)</var>
        <var>$A->(71,action)</var>
      </resultpath>
      <resultpath>
        <file>c:/archiv_xmlfiles/smallexample/macbeth.xml</file>
        <path>/action[1](1.0|1.0)/speaker[1](1.0|0.34)</path>
        <var>$A->(71,action)</var>
      </resultpath>
      <resultpath>
        <file>c:/archiv_xmlfiles/smallexample/macbeth.xml</file>
        <path>/action[1](1.0|1.0)/speech[1](1.0|0.4)</path>
        <var>$A->(71,action)</var>
      </resultpath>
    </result>
    <time_eval> 0 min 1 sec 345 msec</time_eval>
    <time_out> 0 min 0 sec 187 msec</time_out>
  </XMLResult>
```

Es ist bereits hier offensichtlich, dass mit wenig zusätzlichem Wissen die Ausbeute an relevanten Treffern erheblich gesteigert werden kann. Um genaue Aussagen über Präzision (engl.: Precision) und Ausbeute (engl.: Recall) treffen zu können, reichen "kleine" Experimente mit "kleinen" Datenmengen im Allgemeinen nicht aus.

8.2 Experiment 2: "Benchmarktest"

Mit diesem Experiment zeigen wir die Effektivität und Effizienz der XXL–Suchmaschine im Umgang mit einer großen Menge von XML–Dokumenten und demonstrieren die Effektivität der ontologiebasierten Informationssuche in solch zahlreichen XML–Dokumenten.

Das Hauptanliegen der "Initiative for Evaluation of XML Retrieval" besteht darin, inhaltsorientierte und strukturbasierte Informationssuche in einer großen Menge von XML–Dokumenten zu organisieren, ein einheitliches Bewertungsverfahren bereitzustellen und auf diese Weise Vergleichsmöglichkeiten für verschiedene Ansätze des XML–Retrievals verschiedener Forschungsgruppen zu bieten [INEX, INEX03].

8.2.1 Die INEX–Daten

Die INEX–Dokumentkollektion besteht aus 18 Zeitschriften der IEEE Computer Society (engl.: journals) mit allen Ausgaben (engl.: volumes) seit 1995. Für jedes Journal sind die Ausgaben in Verzeichnissen pro Erscheinungsjahr geordnet. Zu jeder Ausgabe gibt es ein XML–Dokument *volume.xml*, das die zugehörigen Artikel (engl.: articles) mit Hilfe von Entities einbindet. Um jeden Artikel als eigenständiges XML–Dokument zu erkennen, haben wir die Entities durch XLinks ersetzt. Die INEX–Dokumentkollektion benötigt etwa 500 MB Speicherplatz und umfasst 125 Ausgaben mit 12.117 Artikeln.

In der folgenden Abbildung 8.5 sehen wir einen Ausschnitt aus dem zugehörigen XML–Graphen. In den gegebenen XML–Dokumenten gibt es maximal Pfade der Länge 19.

Die schattierten Knoten stehen für ein Beispiel von gemischtem Inhalt, d.h. ein Elementwert (ein Satz im Dokument) ist über mehrere i–Knoten verteilt.

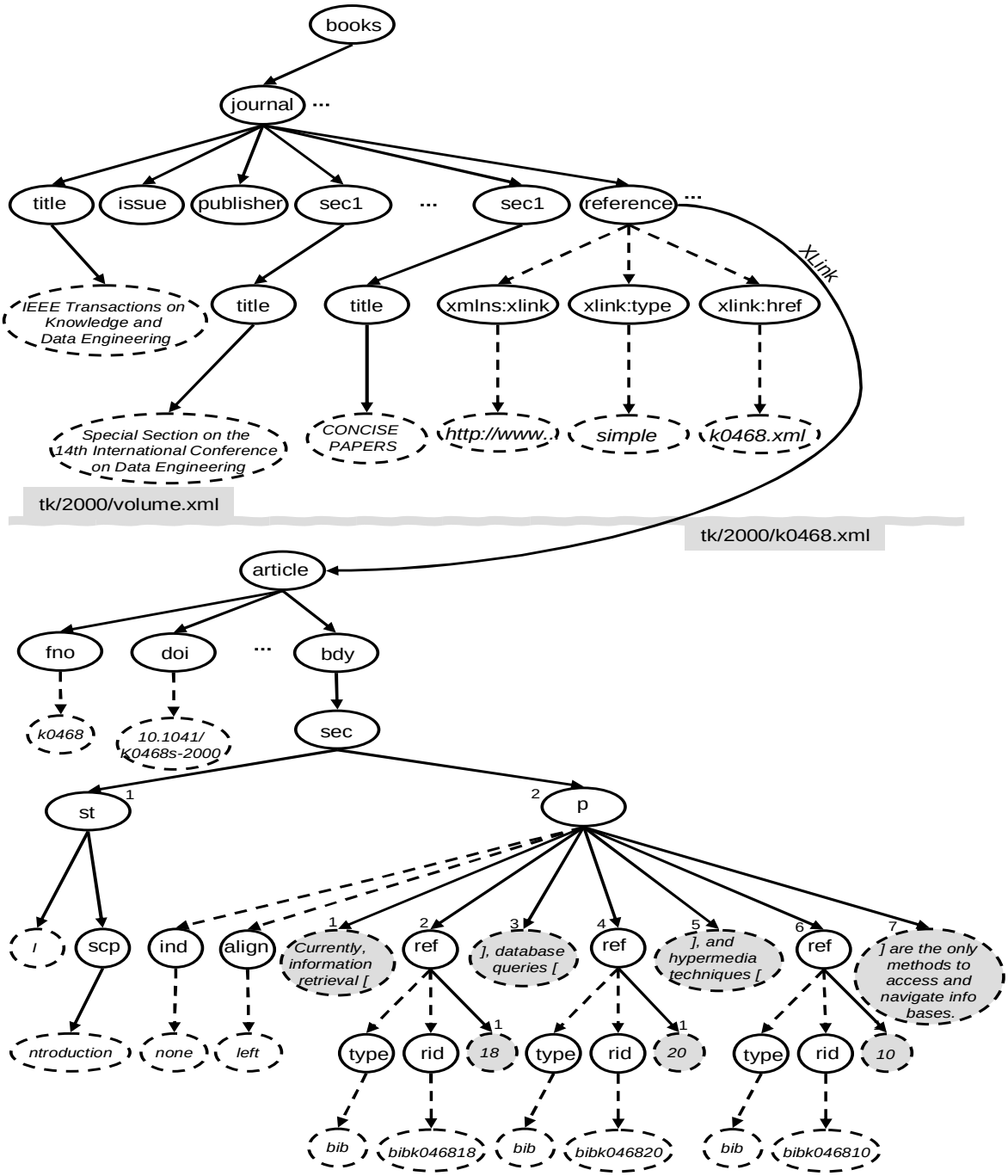


Abbildung 8.5: XML–Graph zum Experiment 2

Ein erster Blick in die Daten (siehe Abb. 8.5) weist auf eine Vielzahl von Besonderheiten der INEX–Dokumentkollektion hin.

- Eine Vielzahl der Element– und Attributnamen ist nicht aus der natürlichen Sprache ableitbar, z.B. steht 'sbt' für 'subtitle'.
- Es gibt diverse Element– und Attributnamen, die formatierenden Charakter haben, z.B. steht 'scp' für 'small caps'.
- Jeder Artikel hat zwar eine reichhaltige Struktur, aber alle Artikel genügen derselben DTD.

- Die INEX–Anfragen enthalten häufig Elementnamen und Stichwörter, die in WordNet nicht zu finden sind.

Aufgrund der relativ einheitlichen und generischen Struktur der vorliegenden XML–Dokumente wird deutlich, dass die wirkliche Stärke von XXL, nämlich die Kombination von semantischen Ähnlichkeitsbedingungen an die Struktur– und die Inhaltsdaten hier nur teilweise ausgenutzt werden kann. Trotzdem können wir die Effizienz und Effektivität der XXL–Suchmaschine anhand dieser XML–Daten demonstrieren.

Zu Beginn des Experiments werden die gegebenen XML–Dokumente (Ausgaben und Artikel) mit Hilfe des XML–Crawlers in ihre Bestandteile zerlegt und in den entsprechenden Tabellen gespeichert. Die Anzahl der Einträge in jeder Tabelle werden in der folgenden Übersicht zusammengefasst:

DB–Tabelle	Anzahl	DB–Tabelle	Anzahl	DB–Tabelle	Anzahl	DB–Tabelle	Anzahl
URLS	12.232	NODES	12.061.348	EDGES	12.049.114	LIN	28.776.664
NAMES	219	CONTENTS	11.779.732	LINKS	408.085	LOUT	4.924.420

Das Importieren und Indexieren der über 12.000 XML–Dokumente benötigt mehr als 9 Stunden, wobei die einzelnen Aufgaben folgende Zeiten in Anspruch nehmen:

```
- Parsen und Speichern der XML-Dokumente       278 min 20 sec
- Konvertierung der Werte von LONG in CLOB      95 min 20 sec
- Analyse der Tabellen                          10 min 31 sec
- Erzeugen der Datenbank-Indizes                22 min 01 sec
- Erzeugen des Oracle Text-Indexes              28 min 01 sec
```

Die Berechnung der 2Hop–Überdeckung auf der Basis eines Algorithmus [STW04], der den gesamten XML–Graphen geeignet partitioniert, dauert mehrere Stunden.

8.2.2 Die INEX–Topics

Der INEX–Benchmark umfasst 36 Anfragen zur inhaltsorientierten Informationssuche (engl.: content–only topic), kurz: CO–Topic, und 30 Anfragen zur strukturbasierten und inhaltsorientierten Informationssuche (engl.: content–and–structure topic), kurz: CAS–Topic. Jeder Topic liegt im XML–Format vor und besteht aus einem XPath–Ausdruck, einer kurzen und einer ausführlichen Beschreibung und einer Menge von Stichwörtern. Im Anhang A sind alle CO– und CAS–Topics vollständig aufgelistet.

Ein CO–Topic enthält ausschließlich Bedingungen an die Werte von Elementen und Attributen.

Beispiel 8.1 In diesem Beispiel geben wir den CO–Topic 112 im XML–Format an.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="112" query_type="CO" ct_no="71">
   <title>+"Cascading Style Sheets" -"Content Scrambling System"</title>
   <description>
      References to World Wide Web Consortium(W3C)'s Cascading Style Sheets (CSS)
      to meet design and style requirements of information.
   </description>
```

```
   <narrative>
      In order to be relevant, the result must simply contain any reference to the
      World Wide Web Consortium(W3C) endorsed Cascading Style Sheet specification.
      An explanation of the concept of separating style and content with CSS is
      especially relevant. We are not interested in Content-Scrambling Systems
      (CSS), which are encryption systems used on some DVDs.
   </narrative>
   <keywords>
      css, style, design, standards, w3c, web, internet, www, html, xml, xhtml
   </keywords>
</inex_topic>
```

Um ein besseres Gefühl für die CO–Topics zu vermitteln, geben wir in der folgenden Tabelle die Stichwörter aus dem `title`–Element für die ersten 10 CO–Topics an. Die CO–Topics 91 bis 126 sind im Anhang A vollständig aufgelistet.

Topic	Stichwörter aus dem `title`–Element
91	Internet traffic
92	”query tightening” ”narrow the search” ”incremental query answering”
93	”Charles Babbage” -institute -inst.
94	”hyperlink analysis” +”topic distillation”
95	+”face recognition” approach
96	+”software cost estimation”
97	Converting Fortran source code
98	”Information Exchange”, +”XML”, ”Information Integration”
99	perl features
100	+association +mining +rule +medical

Ein CAS–Topic enthält Bedingungen an die Struktur– und Inhatlsdaten der gegebenen XML–Dokumente.

Beispiel 8.2 In diesem Beispiel geben wir den CAS–Topic 63 im XML–Format an.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="63" query_type="CAS" ct_no="14">
   <title>
     //article[about(.,'"digital library"') AND
               about(.//p,'+authorization +"access control" +security')]
   </title>
   <description>
      Find articles about digital libraries that also have one or more
      paragraphs dealing with security, authorization or access control
      matters.
   </description>
   <narrative>
      Relevant documents are about digital libraries and include one or more
      paragraphs discussing security, authorization or access control in digital
      libraries.
   </narrative>
   <keywords>digital library, authorization, access control,security</keywords>
</inex_topic>
```

Um die Vielfalt der Strukturbedingungen zu skizzieren, geben wir in der folgenden Tabelle die XPath–Ausdrücke aus dem `title`–Element der ersten 10 CAS–Topics an. Die CAS–Topics 61 bis 90 sind im Anhang A vollständig aufgelistet.

Topic	XPath–Ausdruck aus dem `title`–Element
61	//article[about(.,'clustering +distributed') and about(.//sec,'java')]
62	//article[about(.,'security +biometrics') AND about(.//sec,' "facial recognition" ')]
63	//article[about(.,' "digital library" ') AND about(.//p,'+authorization +"access control" +security')]
64	//article[about(., 'hollerith')]//sec[about(., 'DEHOMAG')]
65	//article[./fm//yr > '1998' AND about(., ' "image retrieval" ')]
66	/article[./fm//yr < '2000']//sec[about(.,' "search engines" ')]
67	//article//fm[(about(.//tig, '+software +architecture') or about(.//abs, '+software +architecture')) and about(., '-distributed -Web')]
68	//article[about(., '+Smalltalk') or about(., '+Lisp') or about(.,'+Erlang') or about(., '+Java')]//bdy//sec[about(.,'+"garbage collection" +algorithm')]
69	/article/bdy/sec[about(.//st,' "information retrieval" ')]
70	/article[about(./fm/abs,' "information retrieval" "digital libraries" ')]

Der zweite Schritt im Rahmen des INEX–Benchmarks besteht für jeden Teilnehmer darin, aus den vorgegebenen CO– und CAS–Topics automatisch Anfragen in der eigenen XML–Anfragesprache zu generieren und diese in der eigenen Systemumgebung auszuwerten. Aufgrund diverser Schwierigkeiten, die sich zum Beispiel aus Differenzen zwischen XPath–Ausdruck und Beschreibung ergeben, erzeugen wir die XXL–Anfragen ausschließlich auf der Basis des `title`–Elements der Topics. Wir müssen zusätzlich beachten, dass im Rahmen des INEX–Benchmarks ausschließlich nach Elementen und Attributen von Artikeln (article) gesucht werden soll, nicht nach Elementen und Attributen der Ausgaben (volume) selbst.

Das `title`–Element eines CO–Topics enthält eine Aufzählung von Stichwörtern. Wir interpretieren die Aufzählung als Konjunktion und das Minuszeichen als Negation und vernachlässigen das Pluszeichen.

Beispiel 8.3 Aus einem gegebenen CO–Topic, z.B. CO–Topic 112, wird dann eine XXL–Anfrage der Form:

```
SELECT *
FROM   INDEX
WHERE  article/# ~ "(Cascading Style Sheets) & !(Content Scrambling)"
```

Der XPath–Ausdruck eines CAS–Topics lässt sich direkt in eine XXL–Pfadbedingung übertragen. Hierbei wird '//' durch '#' ersetzt und der Ausdruck in einer about–Funktion als Inhaltsbedingung mit dem Ähnlichkeitsoperator übernommen.

Beispiel 8.4 Aus einem gegebenen CAS–Topic, z.B. CAS–Topic 63, werden die XPath–Ausdrücke in Pfadausdrücke umgewandelt, die mit den entsprechenden Variablenbindungen und Inhaltsbedingungen ergänzt werden. Die folgende XXL–Anfrage entspricht dem CAS–Topic 63:

```
SELECT $A
FROM   INDEX
WHERE  article AS $A
  AND  $A ~ "digital library"
  AND  $A/#/p ~ "authorization & (access control) & security"
```

8.2.3 Der INEX–Benchmarktest

Das Ziel des INEX–Benchmarks besteht darin, die Effektivität (Ausbeute und Präzision) der Informationssuche in den gegebenen XML–Dokumenten zu untersuchen. Um der Vielfalt der inhalts– und strukturorientierten Suchmöglichkeiten in XML–Daten gerecht zu werden, wurden 36 CO–Topics und 30 CAS–Topics erstellt. Jeder Teilnehmer generiert aus den vorgegebenen Topics Anfragen in seiner eigenen XML–Anfragesprache und wertet diese in der eigenen Systemumgebung aus. Das Ergebnis eines Topics besteht aus einer relevanzbasierten Rangliste von Elementen und Attributen der 12.117 Artikel und wird im XML–Format zurückgegeben. Diese Rangliste muss der folgenden DTD genügen und darf maximal 1.500 Treffer enthalten, um als Ergebnis akzeptiert zu werden.

```
<!ELEMENT inex-submission (description, topic+)>
<!ATTLIST inex-submission
          participant-id CDATA #REQUIRED
          run-id CDATA #REQUIRED
          task (CO | SCAS | VCAS) #REQUIRED
          query (automatic | msnusl) #REQUIRED
          topic-part (T|D|K|TD|TK|DK|TDK) #REQUIRED>
<!ELEMENT description (#PCDATA)>
<!ELEMENT topic (result*)>
<!ATTLIST topic
          topic-id CDATA #REQUIRED>
<!ELEMENT result (file, path, rank?, rsv?)>
<!ELEMENT file (#PCDATA)>
<!ELEMENT path (#PCDATA)>
<!ELEMENT rank (#PCDATA)>
<!ELEMENT rsv (#PCDATA)>
```

Die Relevanzbewertung erfolgt in einem sehr aufwendigen Verfahren. Zunächst werden von allen Teilnehmern die CO– und CAS–Topics ausgewertet. Zu jedem Topic werden die Ergebnisse aller Teilnehmer in einem Pool zusammengefasst. Auf diese Weise hofft man, möglichst viele der tatsächlich relevanten Elemente und Attribute zu erfassen.

Als nächstes werden die Ergebnisse hinsichtlich ihrer Relevanz manuell bewertet. Dazu wird ein zweidimensionales Qualitätsmaß verwendet. Hierbei wird jedem eingereichten Element/Attribut ein Wert für die *Spezifität* und ein Wert für die *Vollständigkeit* zugeordnet. Die Spezifizität (engl.: specificity) beschreibt den Umfang, mit dem der Elementwert sich auf den gegebenen Topic konzentriert. Die Vollständigkeit (engl.: exhaustiveness) beschreibt den Umfang, mit dem der Elementwert sich mit dem gesamten Gebiet des Topics befasst. Für jedes Qualitätsmaß können vier Werte angegeben werden:

- 0 – nicht spezifisch/vollständig,

- 1 – marginal spezifisch/vollständig,
- 2 – ziemlich spezifisch/vollständig bzw.
- 3 – absolut spezifisch/vollständig.

Um Präzision und Ausbeute einzelner Anfrageergebnisse bewerten zu können, müssen die zwei Dimensionen von Spezifizität und Vollständigkeit auf einen einzelnen Relevanzwert quantisiert werden. Im Rahmen des INEX–Benchmarks werden zwei Sichtweisen des Benutzers berücksichtigt und dazu geeignete Quantisierungsfunktionen angegeben.

- Die *exakte Quantisierung* (engl.: strict) berücksichtigt nur die als absolut spezifisch bzw. absolut vollständig bewerteten Ergebnisse und ordnet jedem eingereichten Treffer einen Wert 0 oder 1 zu:

$$f_{strict}(vollst, spez) = \begin{cases} 1 & \textit{falls vollst=3, spez=3 (kurz: 3/3)} \\ 0 & \textit{sonst} \end{cases}$$

- Die *verallgemeinerte Quantisierung* (engl.: generalized) berücksichtigt alle bewerteten Ergebnisse und ordnet jedem eingereichten Treffer einen Wert zwischen 0 und 1 wie folgt zu:

$$f_{generalized}(vollst, spez) = \begin{cases} 1 & \text{falls } 3/3 \\ 0.75 & \text{falls } 2/3, 3/2, 3/1 \\ 0.5 & \text{falls } 1/3, 2/2, 2/1 \\ 0.25 & \text{falls } 1/1, 1/2 \\ 0 & \text{falls } 0/0 \end{cases}$$

Die Rangliste eines Teilnehmers enthält bis zu 1.500 Treffer, die für den vorgegebenen Topic im Sinne seines Retrieval–Ansatzes relevant sind. Der Pool zu diesem Topic enthält eine lange Rangliste von relevanten Treffern nach exakter Quantisierung und nach verallgemeinerter Quantisierung.

Um die Qualität der Ranglisten von verschiedenen Teilnehmern vergleichen zu können, wird hier die Anzahl der nicht–relevanten Treffer abgeschätzt, die ein Benutzer der Ranglisten sich ansehen muss, bevor er eine vorgegebene Anzahl von tatsächlich (im Sinne des Pools) relevanten Treffer gesehen hat.

Die Bewertung der Präzision (engl.: precision) bzw. Ausbeute (engl: recall) einer eingereichten Rangliste von Treffern im INEX–Benchmark beruht auf der Wahrscheinlichkeit $P(rel|retr)$, dass ein Treffer der Rangliste, der von einem Benutzer gesehen und bewertet wurde, relevant ist. Die Wahrscheinlichkeit, dass ein Benutzer nach einer gefundenen Anzahl von relevanten Treffern NR die Durchsicht der Rangliste beendet, ist dann [INEX02]:

$$P(rel|retr)(NR) = \frac{NR}{NR + esl_{NR}} = \frac{NR}{NR + j + s \cdot \frac{i}{r+1}}$$

Die erwartete Suchlänge els_{NR} schätzt die Gesamtanzahl der nicht–relevanten Treffern ab, die in der Rangliste auftreten, bis der NR–te relevante Treffer vom Benutzer gesehen wird. Sei l der Rang des NR–ten relevanten Treffers. Dann steht j für die Anzahl der nicht–relevanten Treffer in der Rangliste vom Rang 1 bis zum Rang (l-1). Der Wert von s beschreibt die Anzahl der relevanten Treffer vom Rang l, die sich der Benutzer angesehen hat. Außerdem bezeichnen r die Anzahl der relevanten Treffer vom Rang l und i die Anzahl der nicht–relevanten Treffer vom Rang l. Dabei wird die Überlappung von Elementen innerhalb einer Rangliste nicht berücksichtigt. Mit

Hilfe dieser Wahrscheinlichkeit kann die durchschnittliche Präzision für eine bestimmte Anzahl von Ausbeute–Punkten NR (z.B. 100) berechnet werden. Die verschiedenen Parameter obiger Gleichung werden durch folgende Abbildung 8.6 zusätzlich näher erläutert.

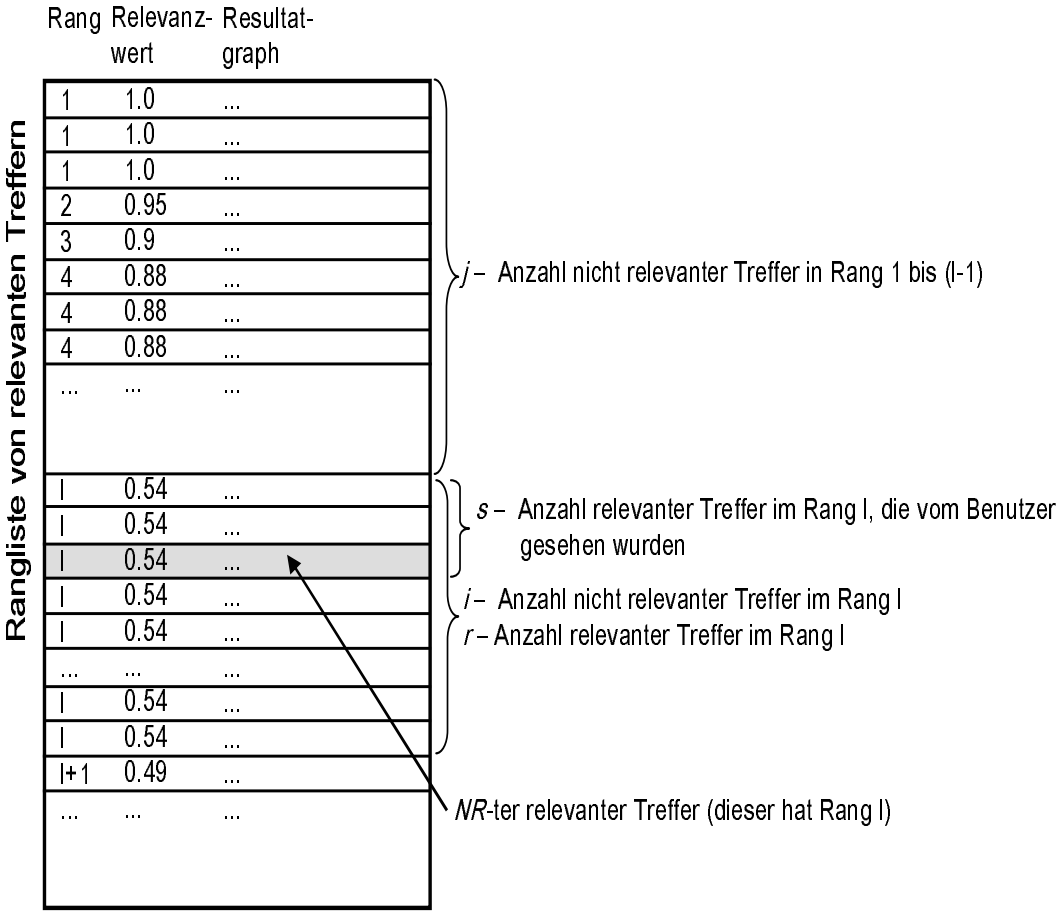


Abbildung 8.6: Exemplarische Rangliste von Treffern

Im Rahmen des INEX–Benchmarks steht ein Assessment–Tool zur Verfügung, das auf der Basis der oben beschriebenen Vorarbeit jede eingereichte Rangliste mit ihren maximal 1.500 Treffern hinsichtlich Präzision und Ausbeute bzgl. der vorliegenden Ergebnisse im zugehörigen Pool bewertet.

Dazu gibt es einmal die Möglichkeit, die Qualität eines einzelnen Anfrageergebnisses bzgl. der beiden Bewertungsansätze *exakte Quantisierung* und *verallgemeinerte Quantisierung* zu untersuchen (siehe Abb. 8.7). Hierbei wird die durchschnittliche Präzision (engl.: average precision) als Mittelwert auf der Basis der eingereichten Treffer ermittelt.

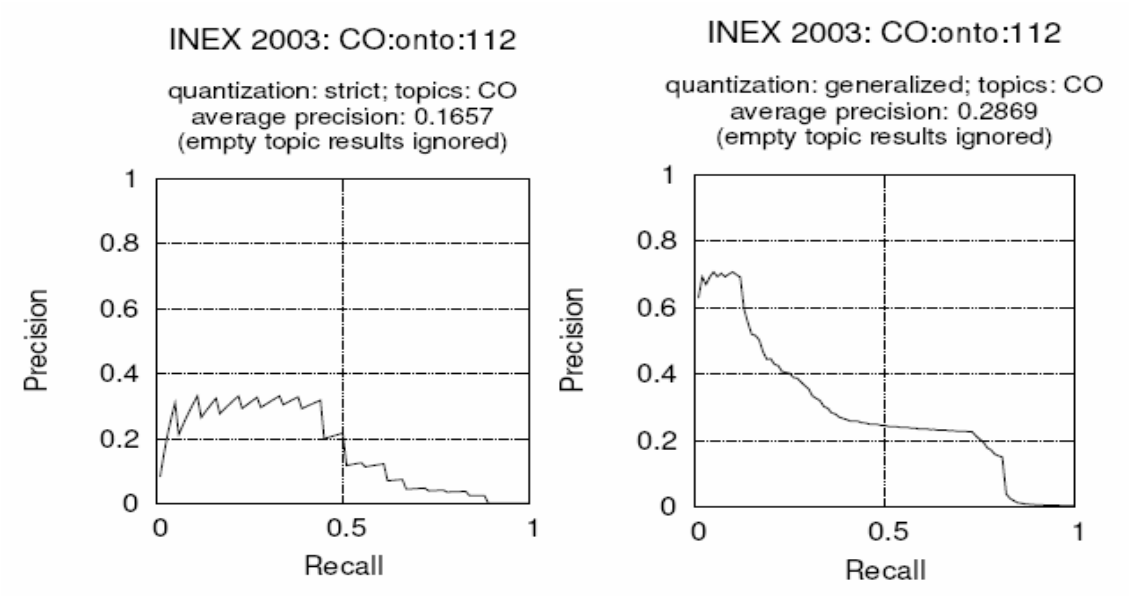


Abbildung 8.7: Bewertung einer einzelnen Rangliste von relevanten Treffern zu einem konkreten Topic

Darüber hinaus besteht die Möglichkeit, einen vollständigen Benchmarktest (also die Auswertung aller 36 CO–Topics bzw. aller 30 CAS–Topics) mit exakter bzw. verallgemeinerter Quantisierung bewerten zu lassen und mit den Ergebnissen der anderen Teilnehmer vergleichen zu lassen (siehe Abb. 8.8). Die hierbei angegebene durchschnittliche Präzision wird über alle Topics (einmal 36 und einmal 30) gemittelt. Insbesondere werden nicht bearbeitete Topics berücksichtigt, sie erhalten die Präzision 0.

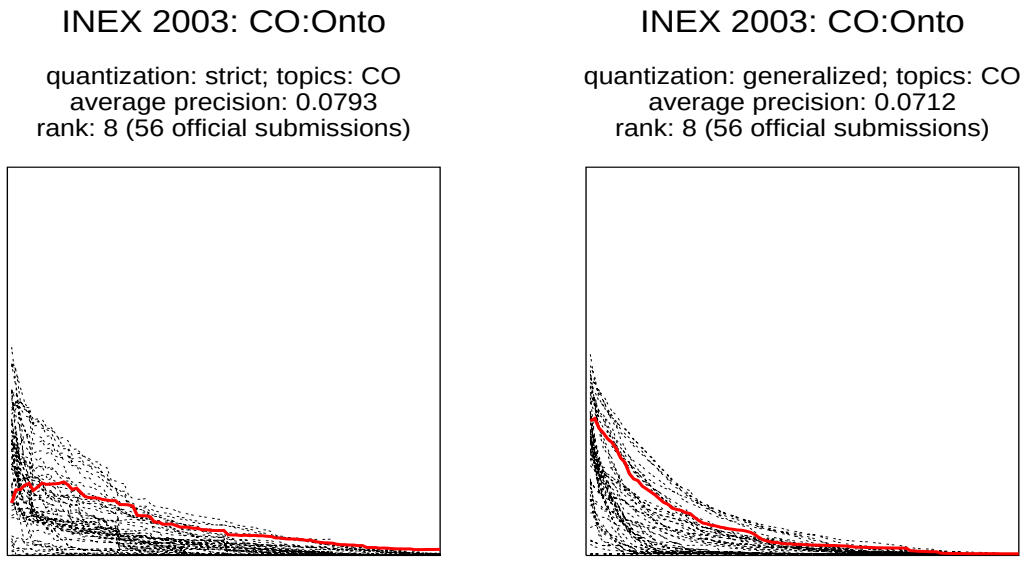


Abbildung 8.8: Bewertung eines Benchmarktests (dunkle Kurve) und Vergleich mit den Testergebnissen der anderen Teilnehmer (gepunktete Kurven)

8.2.4 XXL @ INEX 2003

Wir haben für 4 CO–Topics und für 4 CAS–Topics offizielle Ergebnisse eingereicht. Aussagekräftige Angaben über die Effektivität und Effizienz der XXL–Suchmaschine lassen

sich jedoch nur aus den Ergebnissen umfangreicher Benchmarktests ableiten. Nach dem offiziellen Ende des INEX–Benchmarktests haben wir weitere Testläufe durchgeführt und präsentieren im folgenden die Ergebnisse aus den nachträglichen Benchmarktests.

8.2.4.1 CO–Topics

Wir haben die 36 gegebenen CO–Topics 91 bis 126 (siehe Anhang A) in entsprechende XXL–Anfragen transformiert, die die gegebenen Inhaltsbedingungen ausschließlich für Element– und Attributwerte von Artikeln (article), nicht von Ausgaben (volume) auswerten. Für jeden CO–Topic haben wir einen ersten Testlauf *CO:Init*, bei dem die gegebene XXL–Anfrage ohne Hilfe einer Ontologie ausgewertet wurde, und einen zweiten Testlauf *CO:Onto* durchgeführt, bei dem die gegebene XXL–Anfrage mit Hilfe einer Ontologie ausgewertet wurde.

Wir zeigen zunächst die Ergebnisse für drei CO–Topics 95, 98 und 112 und gehen dabei auf Besonderheiten ein, die sich aus den XML–Daten und den CO–Topics ergeben. Anschließend präsentieren wir die Ergebnisse eines vollständigen CO–Benchmarktests.

CO–Topic 98

Aus den Stichwörtern des `title`–Elements des Topics 98 ergibt sich für den ersten Testlauf folgende XXL–Anfrage *CO:Init:98*:

```
SELECT *
FROM   INDEX
WHERE  article/# ~ "(information exchange)
                   & XML
                   & (information integration)"
```

Mit Hilfe der Ontologie werden zu den einzelnen Stichwörtern semantisch ähnliche Stichwörter disjunktiv hinzugefügt, so dass die folgende XXL–Anfrage *CO:Onto:98* entsteht:

```
SELECT *
FROM   INDEX
WHERE  article/# ~ "((information exchange) | (data exchange) |
                     (heterogeneous data))
                   & (XML | (semistructured data))
                   & ((information integration) | (information sharing))"
```

Für die erste Anfrage erhalten wir 7 und für die zweite Anfrage 28 Treffer. Beide Anfragen werden in weniger als einer Sekunde ausgewertet. Insgesamt werden die beiden Ranglisten mit exakter Quantisierung wie folgt bewertet (siehe Abb. 8.8). Die Präzisionswerte sind für diese beiden Testläufe sehr gering, daher verlaufen die Kurven fast nicht erkennbar nah bei 0.

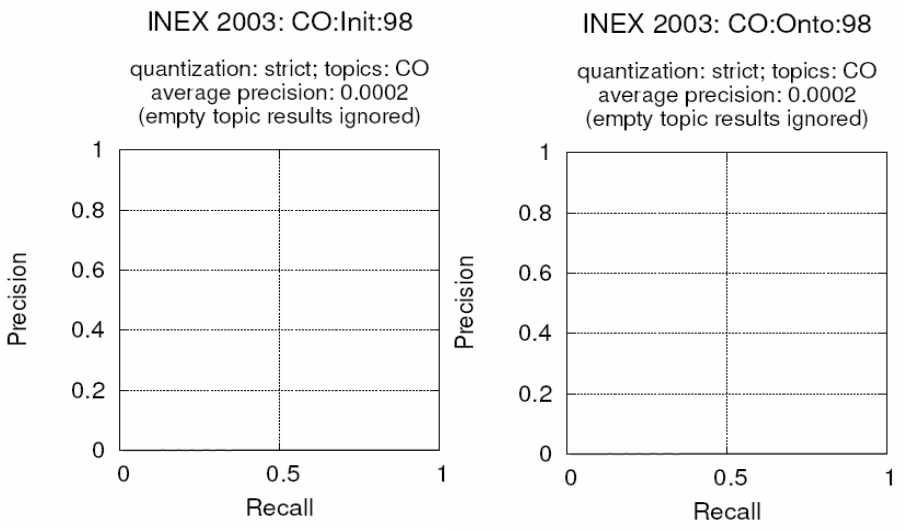


Abbildung 8.8: CO–Topic 98 – Ergebnisse mit exakter Quantisierung

Die Bewertung der beiden Ergebnisse mit verallgemeinerter Quantisierung wird in den Diagrammen der folgenden Abbildung 8.9 dargestellt. Auch in dieser Bewertung schneiden beide Testläufe mit relativ geringen Präzisionswerten ab.

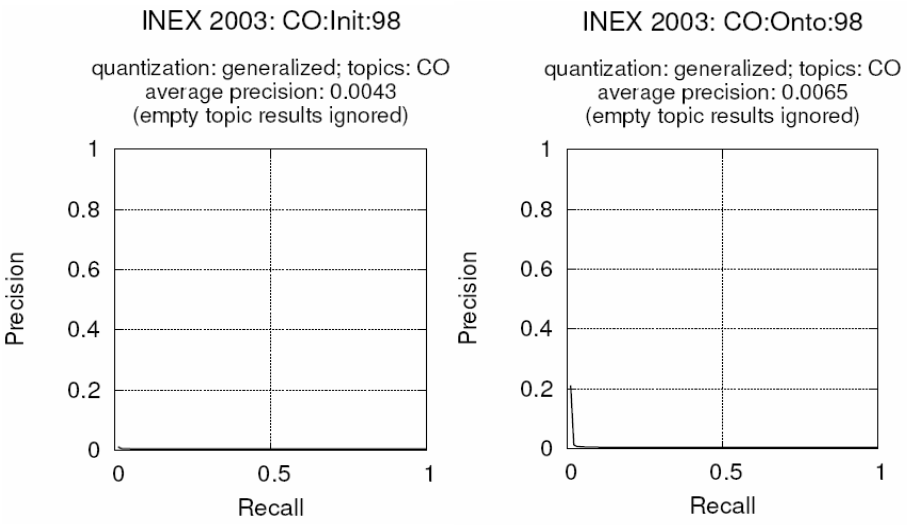


Abbildung 8.9: CO–Topic 98 – Ergebnisse mit verallgemeinerter Quantisierung

Offensichtlich trägt die ontologiebasierte Auswertung des Topics 98 dazu bei, die Qualität des Ergebnisses geringfügig zu verbessern. Ein aufmerksamer Blick auf das vorgegebene CO–Topic 98 lässt die Vermutung zu, dass eine andere logische Verknüpfung der Stichwörter dem eigentlichen Anfrageziel eher gerecht wird. Aus diesem Grund haben wir in diesem Fall einen zweiten Versuch mit den folgenden beiden XXL–Anfragen *CO:Init:98'*:

```
SELECT *
FROM   INDEX
WHERE  article/# ~ "((information exchange) | (information integration))
                   & XML"
```

und *CO:Onto:98'*:

```
SELECT *
FROM   INDEX
WHERE  article/# ~ "((information exchange) | (data exchange) |
                     (heterogeneous data) | (information integration) |
                     (information sharing))
                   & (XML | (semistructured data))"
```

gestartet. Für die erste dieser beiden Anfragen erhalten wir nun 183 und für die zweite 478 relevante Treffer in einer entsprechenden Rangliste. Die Auswertung der ersten Anfrage benötigt etwa eine die der zweiten etwa vier Sekunden. Die folgenden Diagramme zeigen die Ergebnisse mit exakter Quantisierung (siehe Abb. 8.10).

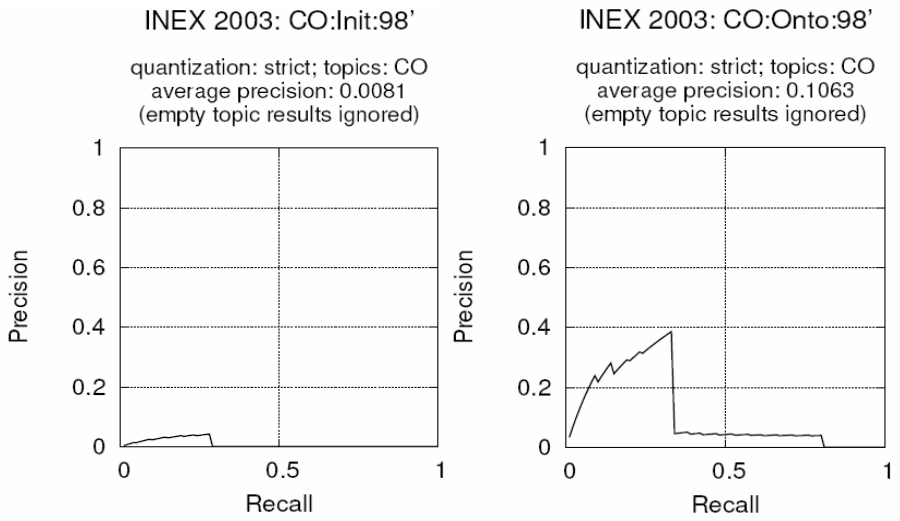


Abbildung 8.10: CO–Topic 98' – Ergebnisse mit exakter Quantisierung

Die zwei Diagramme der nächsten Abbildung 8.11 zeigen die Ergebnisse mit verallgemeinerter Quantisierung (siehe Abb. 8.11).

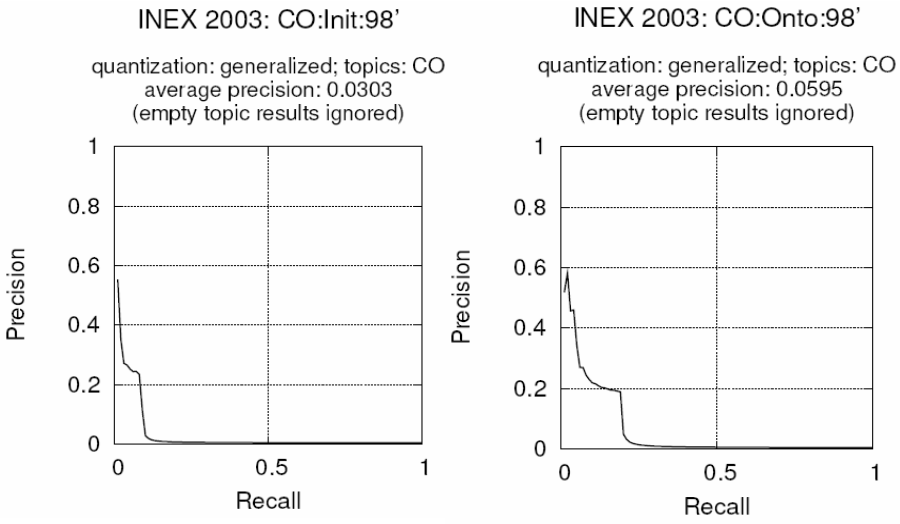


Abbildung 8.11: CO–Topic 98' – Ergebnisse mit verallgemeinerter Quantisierung

Sowohl die Ergebnisse mit exakter als auch die Ergebnisse mit verallgemeinerter Quantisierung zeigen, dass die neue Kombination die Anzahl der Treffer erhöht und

die Qualität der Ergebnismenge deutlich verbessert. Außerdem ist gut zu erkennen, dass das ontologiebasierte Ergebnis mehr relevante Treffer liefert.

CO–Topic 95

Aus den Stichwörtern des `title`–Elements des Topics 95 ergibt sich für den ersten Testlauf folgende XXL–Anfrage *CO:Init:95*:

```
SELECT *
FROM   INDEX
WHERE  article/# ~ "(face recognition) & approach"
```

Die Expansion der Stichwörter mit Hilfe der Ontologie führt zu folgender XXL–Anfrage *CO:Onto:95*:

```
SELECT *
FROM   INDEX
WHERE  article/# ~ "((face recognition) | (face identification) |
                    & (approach | application | expression | algorithm)"
```

Die erste XXL–Anfrage liefert 626 relevante Treffer und die zweite Anfrage liefert 945 relevante Treffer in jeweils etwa 25 Sekunden. Die Diagramme der nächsten beiden Abbildungen 8.12 und 8.13 zeigen die Ergebnisse mit exakter und mit verallgemeinerter Quantisierung.

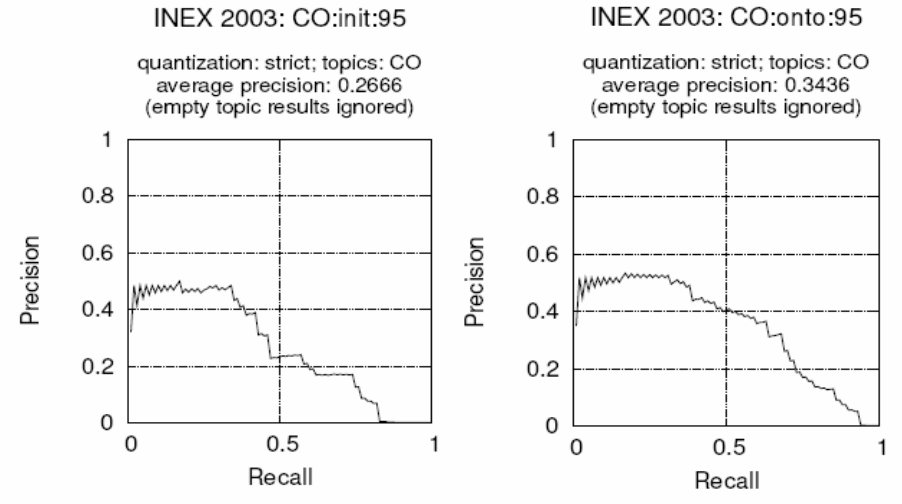


Abbildung 8.12: CO–Topic 95 – Ergebnisse mit exakter Quantisierung

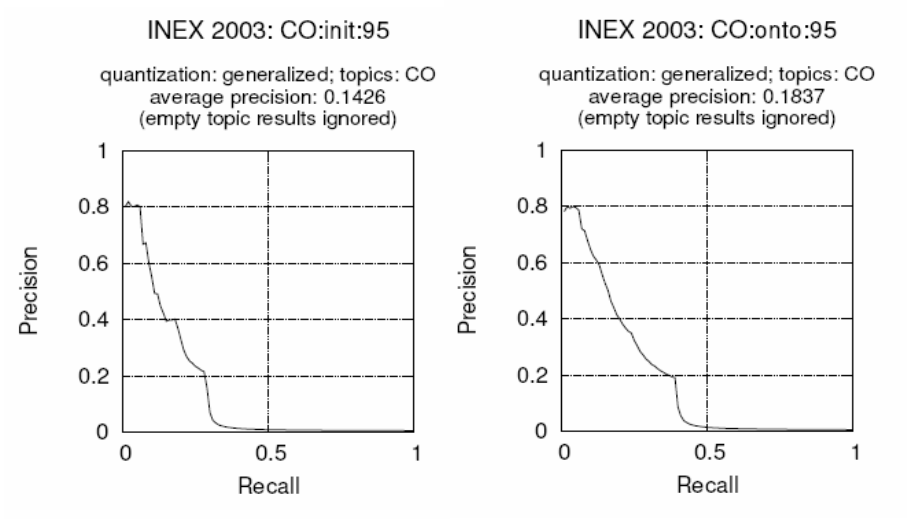


Abbildung 8.13: CO–Topic 95 – Ergebnisse mit verallgemeinerter Quantisierung

Auch an diesem Beispiel wird deutlich, dass sich die Qualität der Ergebnisse durch den Einsatz einer Ontologie bei der Anfrageauswertung deutlich verbessern lässt.

CO–Topic 112

Aus den Stichwörtern des `title`–Elements des Topics 112 ergibt sich für den ersten Testlauf folgende XXL–Anfrage *CO:Init:112*:

```
SELECT *
FROM   INDEX
WHERE  article/# ~ "(Cascading Style Sheets) -(Content Scrambilng)"
```

Die ontologiebasierte Expansion der Stichwörter ergibt folgende XXL–Anfrage *CO:Onto:112*:

```
SELECT *
FROM   INDEX
WHERE  article/# ~ "((Cascading Style Sheets) | (Cascading Style Sheet) |
                    (Style Sheets) | (Style Sheet)) -(Content Scrambilng)"
```

Die erste XXL–Anfrage produziert eine Rangliste mit 345 relevanten Treffern und benötigt dazu etwa 10 Sekunden. Die zweite XXL–Anfrage berechnet 808 relevante Treffer in etwa 20 Sekunden. Die Ergebnisse mit exakter und verallgemeinerter Quantisierung sind in den folgenden beiden Abbildungen 8.14 und 8.15 dargestellt.

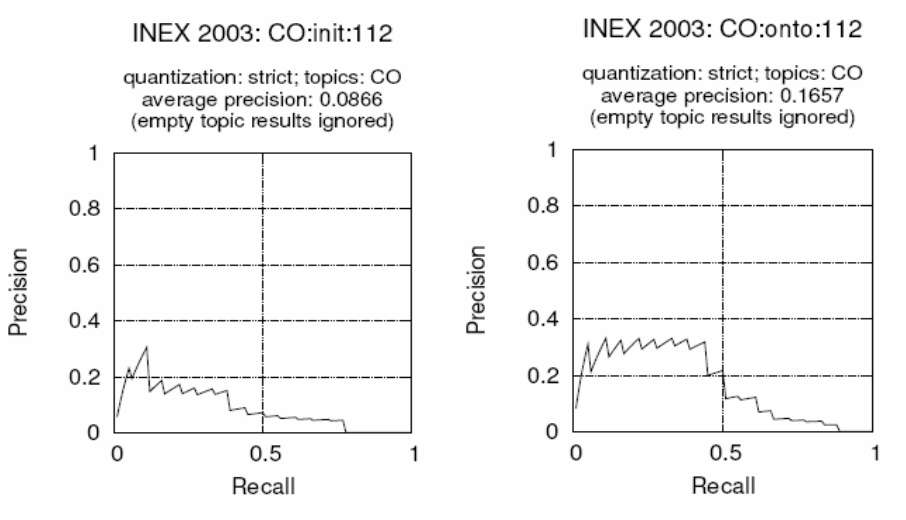


Abbildung 8.14: CO–Topic 112 – Ergebnisse mit exakter Quantisierung

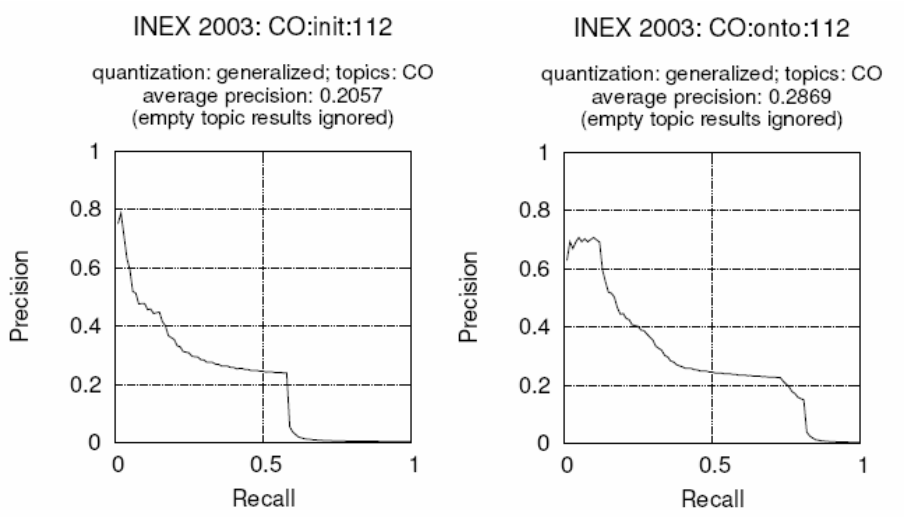


Abbildung 8.15: CO–Topic 112 – Ergebnisse mit verallgemeinerter Quantisierung

Auch das dritte Beispiel zeigt eine deutliche Verbesserung der Effektivität der Informationssuche in den zahlreichen XML–Dokumenten durch den Einsatz einer Ontologie.

CO–Topic komplett

Die Auswertung der 36 CO–Topics im ersten und zweiten Testlauf hängt vor allem von der Anzahl der Treffer ab und benötigt im Allgemeinen wenige Sekunden pro XXL–Anfrage.

Zunächst vergleichen wir mit Hilfe von zwei Balkendiagrammen die durchschnittliche Präzision der Ergebnisse für jeden CO–Topic aus dem ersten und dem zweiten Testlauf.

Für die CO–Topics 105, 106, 114 und 120 sind die Pools bisher nicht bewertet worden, so dass eine Bewertung der Ergebnisse mit exakter bzw. verallgemeinerter Quantisierung nicht möglich ist. Für die CO–Topics 92, 100, 102 und 121 gibt es keine relevanten Treffer, die mit absolut spezifisch und vollständig bewertet wurden. Aus diesem Grund gibt es für diese Topics nur die Möglichkeit der Ergebnisbewertung mit verallgemeinerter Quantisierung.

In der folgenden Abbildung 8.16 sind die Ergebnisse mit exakter Quantisierung für alle

36 CO–Topics des ersten Testlaufs (linker Balken) und des zweiten Testlaufs (rechter dunklerer Balken) dargestellt.

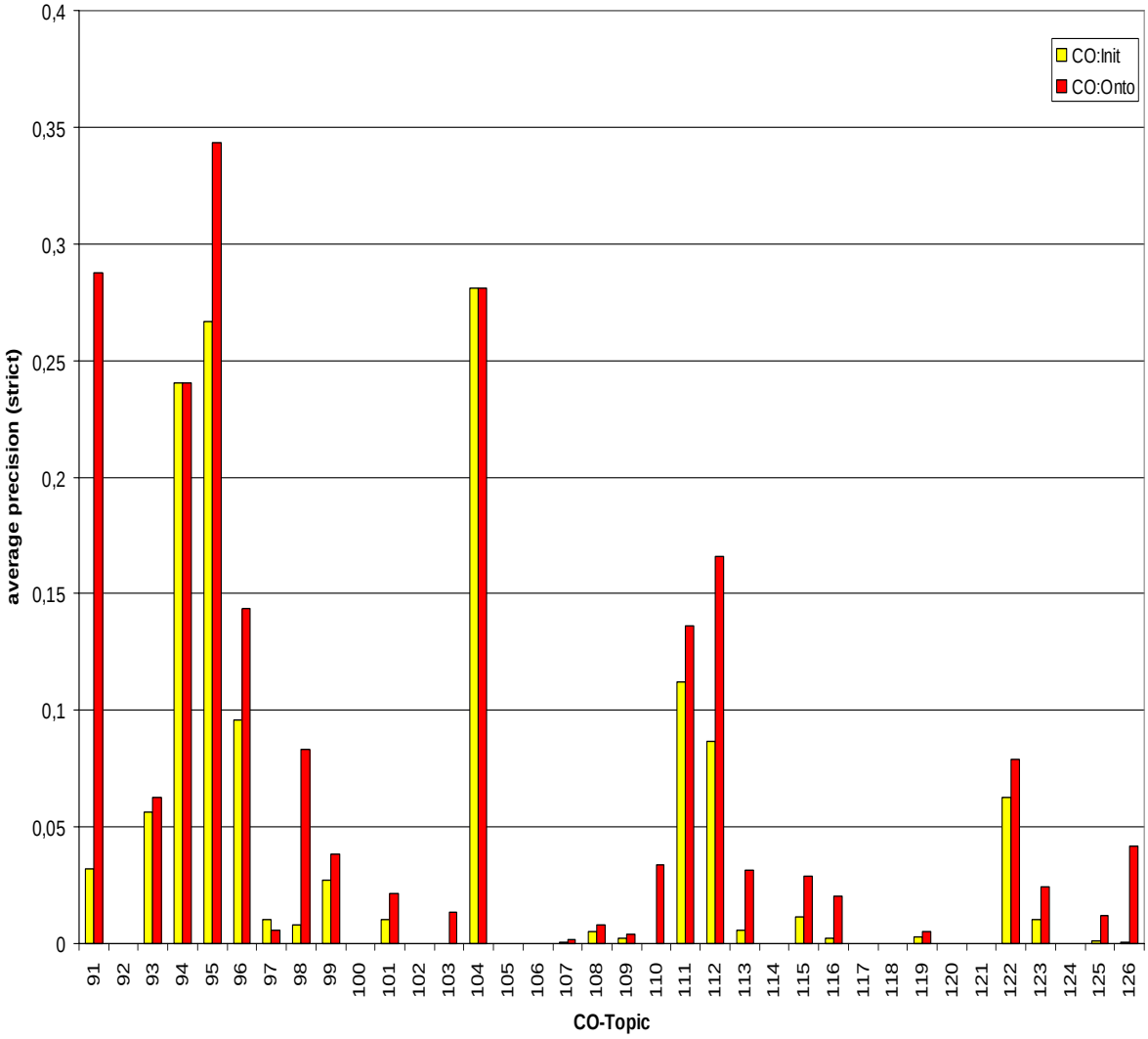


Abbildung 8.16: CO–Topics – Ergebnisse mit exakter Quantisierung

Im Diagramm in der Abbildung 8.17 sind die Ergebnisse mit verallgemeinerter Quantisierung aus dem ersten Testlauf (linker Balken) und dem zweiten Testlauf (rechter, dunklerer Balken) für jeden CO–Topic gegenüber gestellt.

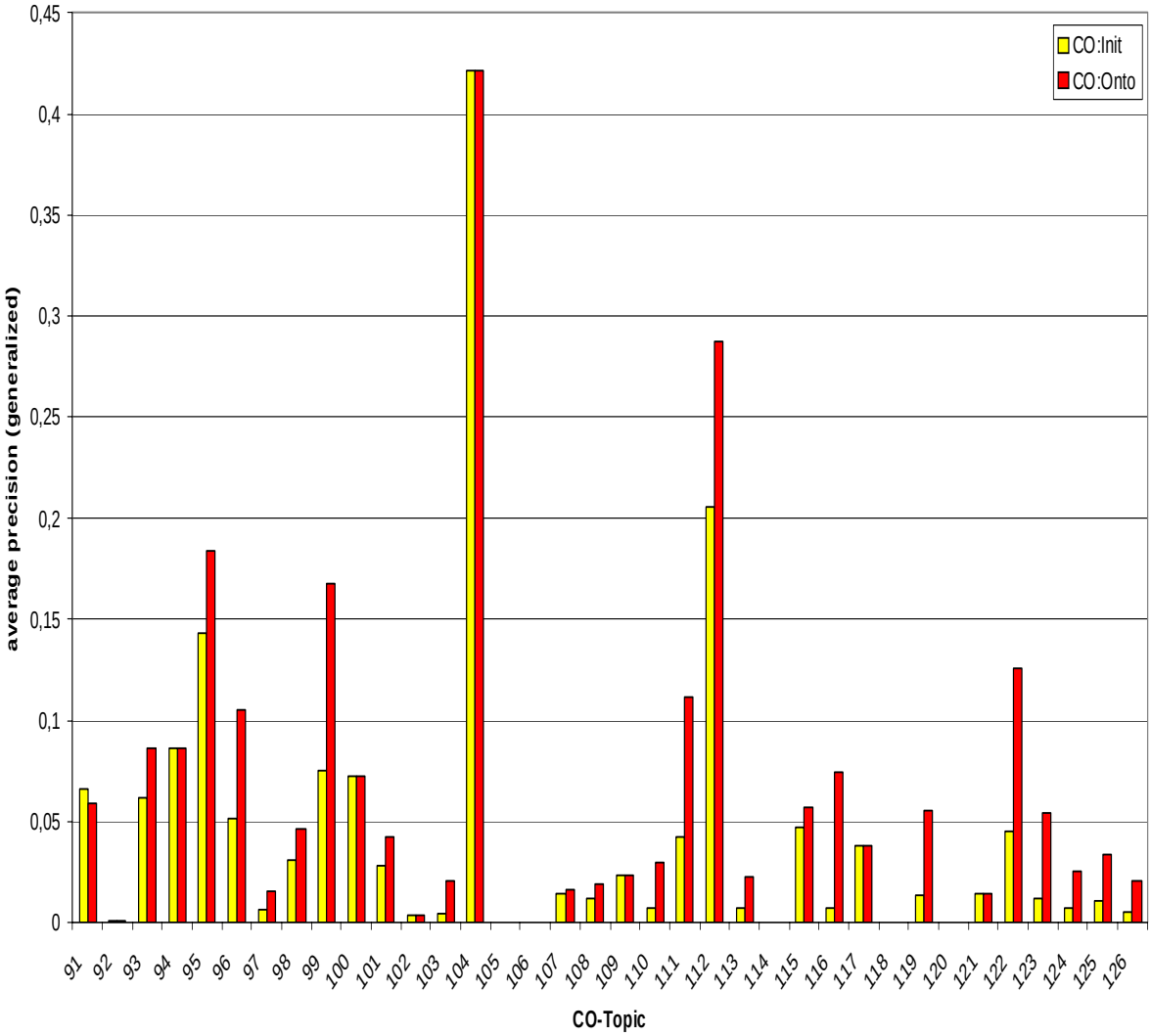
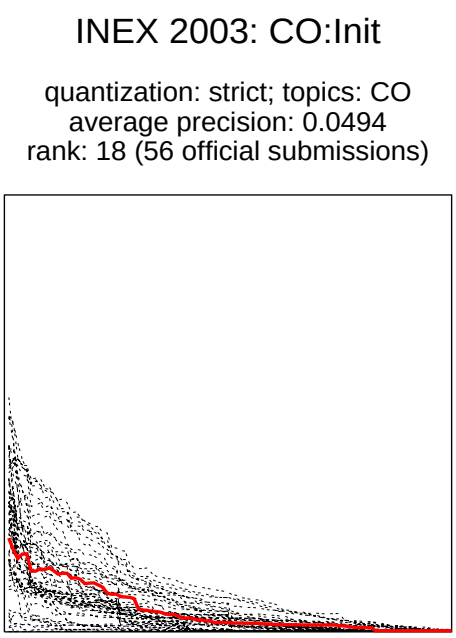
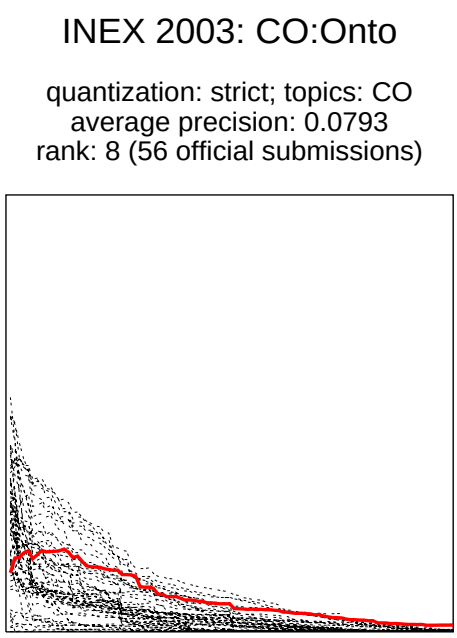


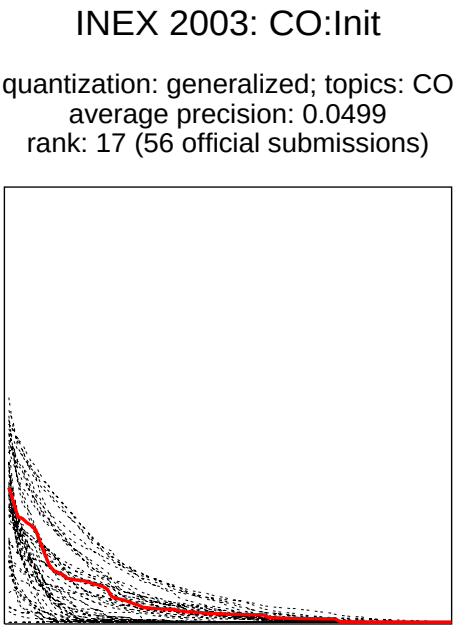
Abbildung 8.17: CO–Topics – Ergebnisse mit verallgemeinerter Quantisierung

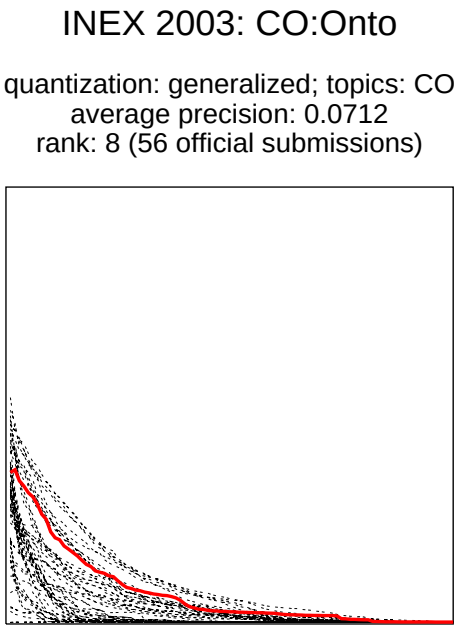
In beiden Abbildungen ist gut zu erkennen, dass der Einsatz der Ontologie bei der Anfrageauswertung in den meisten Fällen zu einer deutlichen Verbesserung der Ergebnisqualität führt. In jedem Fall verschlechtert sich die Qualität der ontologiebasierten Anfrageauswertung nicht.

Diese beiden eben beschriebenen Testläufe wurden außerhalb der offiziellen Benchmarktests durchgeführt. Dennoch geben wir den Vergleich zu den offiziell eingereichten Ergebnissen an, um die deutliche Verbesserung der Ergebnisse durch den Einsatz einer Ontologie hervorzuheben. Dabei zeigt die dunkle Kurve unser Testergebnis im Vergleich zu den jeweils anderen offiziell 56 eingereichten Testergebnissen (gepunktete Kurven).









Die Auswertung der gegebenen XXL–Anfragen würde im Vergleich zu den anderen Teilnehmern Platz 18 bzw. 17 von insgesamt 56 bedeuten. Die Verwendung einer Ontologie verbessert die Effektivität der Informationssuche auf der Basis semantischer Ähnlichkeit von Stichwörtern erheblich, so dass die Ergebnisse für den zweiten ontologiebasierten Testlauf Platz 8 bzw. Platz 7 bedeuten.

8.2.4.2 CAS–Topics

Bei der Auswertung der CAS–Topics sind wir auf enorme Laufzeit–Probleme bei der Auswertung von Strukturbedingungen aufgrund der Menge der Daten und der naiven Herangehensweise gestoßen. Aus diesem Grund haben wir nur wenige CAS–Topics ausgewertet, um die Effektivität der exakten Auswertung von Strukturbedingungen kombiniert mit der ontologiebasierten Auswertung von Inhaltsbedingungen zu demonstrieren.

Wir zeigen die Ergebnisse für die CAS–Topics 62 und 63 und gehen dabei auf die Besonderheiten gegeben durch die vorliegenden XML–Daten und die CAS–Topics ein.

CAS–Topic 62

Der CAS–Topic 62 enthält folgenden XPath–Ausdruck in seinem `title`–Element:

```
//article[about(.,'security +biometrics') AND
          about(.//sec, '"facial recognition"')]
```

Die zugehörige XXL–Anfrage *CAS:Init:62* für den ersten Testlauf ist von der Form:

```
SELECT $A
FROM   INDEX
WHERE  article AS $A
  AND  $A ~ "security & biometrics"
  AND  $A/#/sec ~ "facial recognition"
```

Die Ergänzung der Stichwörter um semantisch ähnliche Stichwörter mit Hilfe der Ontologie führt zu folgender XXL–Anfrage *CAS:Onto:62*:

```
SELECT $A
FROM   INDEX
WHERE  article AS $A
  AND  $A ~ "(security | safety) & (biometrics | biometrical)"
  AND  $A/#/sec ~ "(facial recognition) | (face recognition) |
                   (face identification)"
```

Die erste Anfrage erzeugt innerhalb weniger Sekunden eine Rangliste bestehend aus zwei relevanten Treffern. Die zweite Anfrage produziert 21 Treffer und benötigt dazu etwa eine Stunde. Die Qualität der Ergebnisse mit exakter Quantisierung ist durch die Diagramme der folgenden Abbildung 8.18 dargestellt.

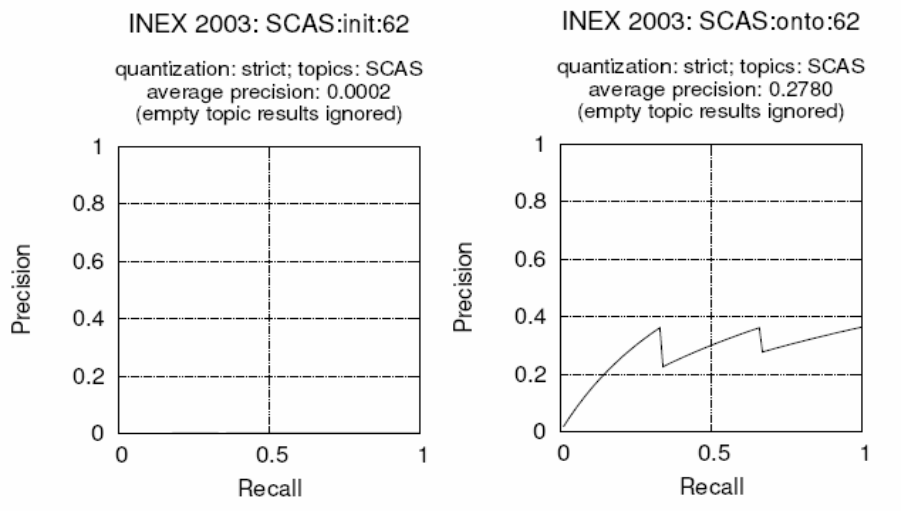


Abbildung 8.18: CAS–Topic 62 – Ergebnisse mit exakter Quantisierung

Die nächste Abbildung 8.19 beinhaltet die Ergebnisse zum CAS–Topic 62 mit verallgemeinerter Quantisierung.

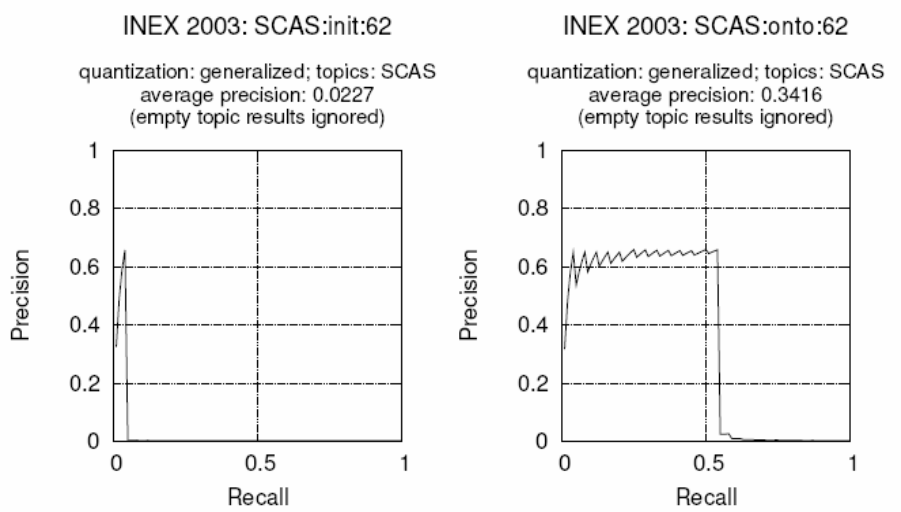


Abbildung 8.19: CAS–Topic 62 – Ergebnisse mit verallgemeinerter Quantisierung

Anhand dieser zwei Abbildungen wird sehr schön deutlich, dass XXL zum einen exakte Strukturbedingungen exakt auswertet und das auch hier die Erweiterung der Inhaltsbedingung mit semantisch ähnlichen Stichwörtern die Präzision und Ausbeute der Ergebnisse erheblich steigert.

CAS–Topic 63

Der CAS–Topic 63 enthält in seinem `title`–Element folgenden XPath–Ausdruck:

```
//article[about(.,'"digital library"') AND
          about(.//p, '+authorization +"access control +security"')]
```

Daraus wird die folgende XXL–Anfrage *CAS:Init:63* generiert:

```
SELECT $A
FROM   INDEX
WHERE  article AS $A
  AND  $A ~ "digital library"
  AND  $A/#/p ~ "authorization & (access control) & security"
```

Mit Hilfe der Ontologie gibt es zu den vorgegebenen Stichwörtern diverse semantisch ähnliche Stichwörter. Die daraus resultierende XXL–Anfrage *CAS:Onto:63* sieht wie folgt aus:

```
SELECT $A
FROM   INDEX
WHERE  article AS $A
  AND  $A ~ "(digital library) | (digital libraries)"
  AND  $A/#/p ~ "(authorization | identification | recognition)
                 & (access control)
                 & (security | safety)"
```

Die erste XXL–Anfrage liefert drei relevante Treffer und die zweite XXL–Anfrage sechs. Zu beiden Ranglisten ergibt die durchschnittliche Präzision mit exakter Quantisierung einen Wert von 0.0002 und mit verallgemeinerter Quantisierung einen Wert von 0.0020. Die Präzision und Ausbeute ist offensichtlich in beiden Fällen sehr gering und auch die Verwendung der Ontologie brachte keinen messbaren Erfolg. Ein genauerer Blick auf den ursprünglichen CAS–Topic legt die Vermutung nahe, dass die Stichwörter im gesuchten `p`–Element zu einem Thema gehören und deshalb disjunktiv verknüpft werden müssen. Für diesen Fall haben wir einen zweiten Versuch mit der XXL–Anfrage *CAS:Init:63'*

```
SELECT $A
FROM   INDEX
WHERE  article AS $A
  AND  $A ~ "digital library"
  AND  $A/#/p ~ "authorization | (access control) | security"
```

und der XXL–Anfrage *CAS:Onto:63'*

```
SELECT $A
FROM   INDEX
WHERE  article AS $A
  AND  $A ~ "(digital library) | (digital libraries)"
  AND  $A/#/p ~ "(authorization | identification | recognition |
                 (access control) | security | safety"
```

gestartet. Die Ergebnisse mit exakter Quantisierung sind in der folgenden Abbildung 8.20 zusehen. Die mittlere Präzision verbessert sich von den oben genannten geringen Werten auf jeweils 1.0. Aus diesem Grund sind die Kurven, die bei 1.0 verlaufen, schwer zu erkennen.

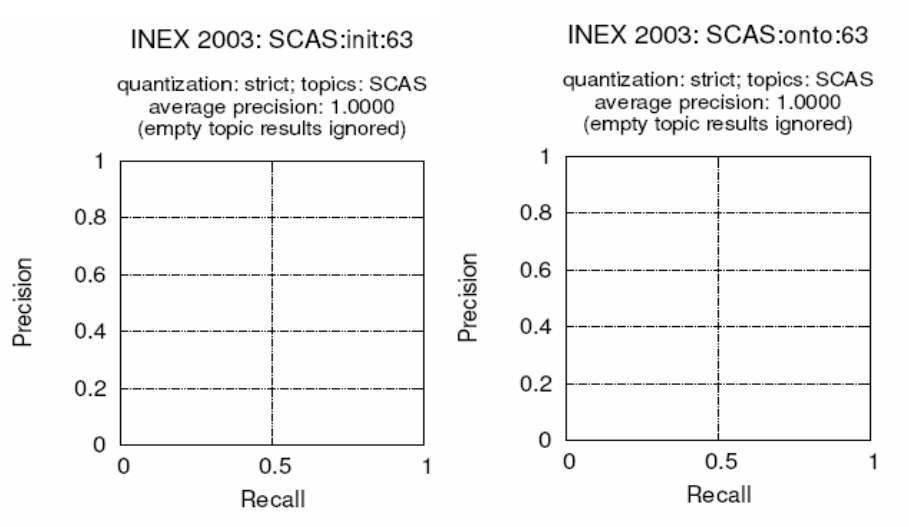


Abbildung 8.20: CAS–Topic 63' – Ergebnisse mit exakter Quantisierung

Die nächste Abbildung 8.19 beinhaltet die Ergebnisse zum CAS–Topic 63 mit verallgemeinerter Quantisierung. In beiden Testläufen zeigen die Ergebnisse die maximale Präzision von 1.0 unter den ersten Treffern, die dann schlagartig mit zunehmender Trefferzahl sehr gering wird.

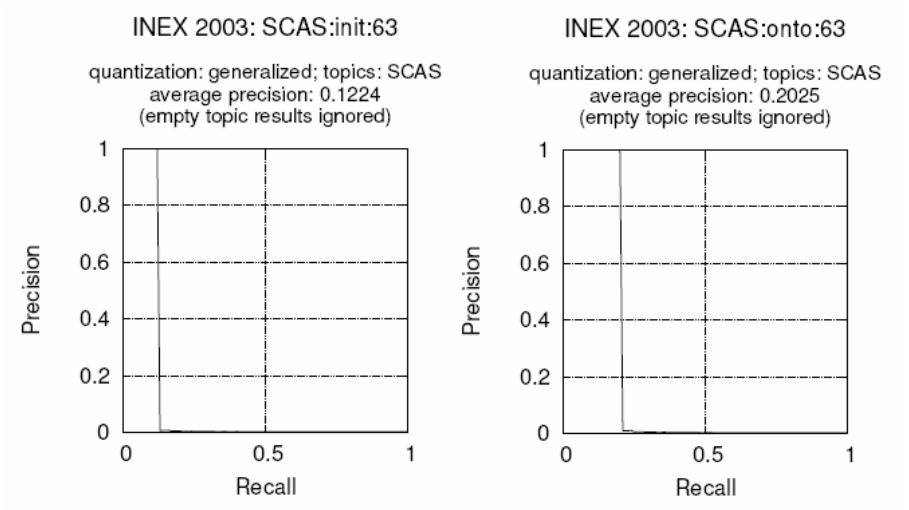


Abbildung 8.21: CAS–Topic 63' – Ergebnisse mit verallgemeinerter Quantisierung

Dieser Ergebnisse bestätigen unsere Vermutung über den Umgang mit den angegebenen Stichwörtern. Darüber hinaus wird hier die Effektivität der ontologiebasierten Anfrageauswertung deutlich. Die Präzision von 1.0 für die beiden Testläufe im Sinne der exakten Quantisierung ist nicht zu überbieten. Aber im Fall der verallgemeinerten Quantisierung wird deutlich, dass die Anfrageauswertung mit Hilfe der Ontologie nicht nur die durchschnittliche Präzision erheblich verbessert, sondern auch Präzision und Ausbeute für Treffer erhöht, die am Anfang der Rangliste stehen.

8.2.4.2 Fazit

Als Zusammenfassung aus diesem umfangreichen INEX–Benchmarktest verbleiben folgende Anmerkungen.

- Der Einsatz von Ontologien zur Verbesserung der Effektivität der Informationssuche in einer großen Menge von XML–Dokumenten wurde hier aus den Ergebnissen deutlich sichtbar.
- Die Ähnlichkeitssuche auf den Strukturdaten der gegebenen XML–Daten ist im Prinzip nicht möglich.
- Die Relevanzbewertungen von Ergebnissen durch Teilnehmer sind subjektiv und daher von Teilnehmer zu Teilnehmer verschieden. Das fhrt dazu, dass die durchschnittlichen Präzisionen nicht sehr hoch sind.
- Die CO–Topics lassen sich effizient auswerten. CAS–Topics mit einfachen Strukturbedingungen können ebenfalls sehr effizient ausgewertet werden. CAS–Topics mit vielen Ergebnissen oder Zwischenergebnissen oder komplexen Strukturbedingungen lassen sich aktuell nur sehr ineffizient auswerten.

Kapitel 9

Ausblick

Im Rahmen dieser Dissertation haben wir die XXL–Suchmaschine vorgestellt. Sie ist für die Informationssuche in beliebigen XML–Dokumenten konzipiert. XML–Dokumente zeichnen sich durch die Kombination von Strukturdaten (Element– und Attributnamen und deren Anordnung) und Inhaltsdaten (Element– und Attributwerte) aus. Die XXL–Suchmaschine wertet Anfragen im XXL–Format aus und verwendet dazu eine Ontologie, um semantische Ähnlichkeitsbedingungen an die Namen und Werte von Elementen bzw. Attributen auszuwerten. Als Ergebnis wird eine absteigend sortierte Rangliste von relevanten Teilgraphen des XML–Graphen, der die XML–Dokumente repräsentiert, erzeugt. Das Konzept der ontologiebasierten Ähnlichkeitssuche haben wir im Rahmen der XXL–Suchmaschine als Prototyp implementiert und umfangreich evaluiert.

Die Evaluation hat die Effektivität der XXL–Suchmaschine zur Verbesserung der Informationssuche in XML–Dokumenten demonstriert. Die Entwicklung des Konzepts zur ontologiebasierten Ähnlichkeitssuche in semistrukturierten XML–Dokumenten und die Evaluation im Rahmen des INEX–Benchmarks haben aber auch offene Fragestellungen und Probleme aufgezeigt.

Die Relevanzbewertung von Treffern ist ein zentrales Problem bei der ranglistenbasierten Informationssuche. Im Rahmen dieser Arbeit verwenden wir eine quantifizierte Ontologie zur semantischen Ähnlichkeitssuche nach Namen und Werten von Elementen und Attributen. Hierbei gibt es Aspekte, die unserer Meinung nach hinsichtlich ihres Einflusses auf die Relevanz von Elementen und Attributen betrachtet werden sollten. Dazu gehört z.B. die Länge eines Elementwertes, die Schachtelungstiefe und die Überlappung mit anderen relevanten Elementwerten.

Die Effizienz ist neben der Effektivität ein wichtiger Aspekt bei der Anfrageverarbeitung. Die Evaluation der XXL–Suchmaschine hat Schwächen in der Effizienz bei der Auswertung von XXL–Anfragen aufgedeckt, die sich durch die Wahl der Algorithmen erklären lassen. Hier eröffnet sich ein weites Feld für die Verbesserung der Anfrageverarbeitung durch effizientere Algorithmen und die Optimierung der Anfrageverarbeitung z.B. durch geschickte Wahl der globalen Auswertungsreihenfolge bei den XXL–Suchbedingungen sowie durch geeignete Wahl der lokalen Auswertungsstrategien für jede dieser XXL–Suchbedingungen.

Die Evaluation von Konzepten und Systemen ist zum einen bedeutsam für die qualitative Bewertung dieser Konzepte und Systeme und zum anderen wichtig zum weltweiten Vergleich mit anderen Arbeiten anderer Forschungsgruppen in diesem Fachgebiet. Unsere Erfahrungen, die wir durch die Teilnahme am INEX–Benchmark gemacht haben, unterstreichen vor allem unser Interesse an heterogenen, semistrukturierten Dokument-

kollektionen, um die Leistungsfähigkeit der XXL–Suchmaschine hinsichtlich der ontologiebasierten Ähnlichkeitssuche in Struktur– und Inhaltsdaten unter Beweis zu stellen. Darüber hinaus haben die ersten zwei Jahre INEX gezeigt, dass die Schaffung einer Grundlage zur einheitlichen Relevanzbewertung ein komplexer und komplizierter Vorgang ist, der einen potentiellen Schwachpunkt in der Subjektivität des Relevanzbegriffs hat.

Zusammenfassend lässt sich festhalten, dass die Effektivität der Informationssuche in XML–Dokumenten durch den Einsatz einer Ontologie zur Erkennung von semantischen Ähnlichkeiten erheblich verbessert wird. Dabei verstehen wir die XXL–Suchmaschine als Ausgangspunkt für weitere Forschungsarbeit in verschiedensten Teilgebieten der Informationssuche in semistrukturierten XML–Dokumente unter Verwendung von Technologien aus dem Information Retrieval und aus dem Datenbankbereich.

Anhang A

INEX–Topics 2003

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="61" query_type="CAS" ct_no="2">
<title>//article[about(.,'clustering +distributed') and about(.//sec,'java')]</title>
<description>Retrieve articles about clustering with a section describing usage of the java programming
language and the java virtual machine.</description>
<narrative> The article should contain a description of usage of java based applications or techiques
for running applications on several computers (clustering). Please note that I am not interested in the
"clustering" concept from statistics and pattern recognition. </narrative>
<keywords>clustering,distributed, java</keywords></inex_topic>

<inex_topic topic_id="62" query_type="CAS" ct_no="5">
<title>//article[about(.,'security +biometrics') AND about(.//sec,'"facial recognition"')]</title>
<description> Find articles about biometrical security systems with a section about facial recognition.
</description>
<narrative> To be relevant the document has to discuss the usage of facial recognition in biometrical
security systems. </narrative>
<keywords>facial recognition, face recognition, security, biometrics</keywords></inex_topic>

<inex_topic topic_id="63" query_type="CAS" ct_no="14">
<title>//article[about(.,'"digital library"') AND about(.//p,'+authorization +"access control"
+security')]</title>
<description>Find articles about digital libraries that also have one or more paragraphs dealing with
security, authorization or access control matters. </description>
<narrative> Relevant documents are about digital libraries and include one or more paragraphs discussing
security, authorization or access control in digital libraries. </narrative>
<keywords>digital library, authorization, access control,security </keywords></inex_topic>

<inex_topic topic_id="64" query_type="CAS" ct_no="16">
<title>//article[about(., 'hollerith')]//sec[about(., 'DEHOMAG')]</title>
<description>In articles discussing Herman Hollerith find sections that are about DEHOMAG. </description>
<narrative>Relevant sections deal with DEHOMAG (Deutsche Hollerith Maschinen Gesellschaft) in documents
that discuss work or life of Herman Hollerith.</narrative>
<keywords>Hollerith, DEHOMAG, Deutsche Hollerith Maschinen Gesellschaft </keywords></inex_topic>

<inex_topic topic_id="65" query_type="CAS" ct_no="23">
<title>//article[./fm//yr > '1998' AND about(., '"image retrieval"')]</title>
<description>Find articles about image retrieval that are written after 1998.</description>
<narrative>Relevant documents are research articles about image retrieval written after 1998. Editorials,
news or volume indices are not relevant.</narrative>
<keywords>image retrieval, retrieve images</keywords></inex_topic>

<inex_topic topic_id="66" query_type="CAS" ct_no="34">
<title>/article[./fm//yr &lt; '2000']//sec[about(.,'"search engines"')]</title>
<description>Looking for sections of articles published before 2000, which discuss search engines.
</description>
<narrative>Looking for sections of articles published before 2000, which discuss search engines.
Comparison of different search engines is of great interest.</narrative>
<keywords>search engines, metasearch engines, google, yahoo, hotbot, lycos, metaseek, altavista,
infoseek, excite, webcrawler, opentext, NECI METASEARCH ENGINE, metacrawler</keywords></inex_topic>

<inex_topic topic_id="67" query_type="CAS" ct_no="38">
<title>//article//fm[(about(.//tig, '+software +architecture') or about(.//abs, '+software
+architecture')) and about(., '-distributed -Web')]</title>
<description> An overview of software architecture as it applies to traditional software engineering,
not to distributed nor web software development. </description>
<narrative> To be relevant an abstract must discuss software architecture as it applies to stand-alone
```

application development. It should not discuss the software architecture as it applies to web software or distributed software engineering. The ideal abstract would be to a document giving an overview of the topic. </narrative>
<keywords>software, architecture, distributed, web</keywords></inex_topic>

<inex_topic topic_id="68" query_type="CAS" ct_no="40">
<title> //article[about(., '+Smalltalk') or about(., '+Lisp') or about(.,'+Erlang') or about(., '+Java')]//bdy//sec[about(., '+"garbage collection" +algorithm')] </title>
<description> Retrieve sections of documents discussing implemented and working garbage collection algorithms in programming languages such as Smalltalk, Lisp, Erlang or Java. </description>
<narrative> I have a Lisp compiler and a Smalltalk compiler in progress, and I'd like to find out about recent garbage collection methods. Relevant article sections will discuss recent advances in programming language garbage collection methods that are implemented and work. If a component simply praises a language for having (or not having) garbage collection, that component is not relevant. Footnote references are not relevant, general discussions are not relevant. The component must discuss how to make garbage collection work. </narrative>
<keywords>"garbage collection", algorithm, Smalltalk, Lisp, Erlang,Java</keywords></inex_topic>

<inex_topic topic_id="69" query_type="CAS" ct_no="46">
<title> /article/bdy/sec[about(.//st,'"information retrieval"')] </title>
<description>Retrieve sections of articles where the section or subsection title indicates that it is about information retrieval.</description>
<narrative>To be relevant the retrieved section (or its subsection) must indicate that it is about information retrieval.</narrative>
<keywords>information retrieval</keywords></inex_topic>

<inex_topic topic_id="70" query_type="CAS" ct_no="49">
<title> /article[about(./fm/abs,'"information retrieval" "digital libraries"')]</title>
<description>Retrieve articles with an abstract indicating the article is about information retrieval and/or digital libraries</description>
<narrative>To be relevant the retrieved articles must be about information retrieval, digital libraries or, preferably both. Articles about information retrieval from digital libraries will receive the highest relevance judgements.</narrative>
<keywords>information retrieval,digital libraries</keywords></inex_topic>

<inex_topic topic_id="71" query_type="CAS" ct_no="53">
<title>//article[about(.,'formal methods verify correctness aviation systems')]/bdy//*[about(.,'case study application model checking theorem proving')]</title>
<description>Find documents discussing formal methods to verify correctness of aviation systems. From those articles extract parts discussing a case study of using model checking or theorem proving for the verification. </description>
<narrative>To be considered relevant a document must be about using formal methods to verify correctness of aviation systems, such as flight traffic control systems, airplane- or helicopter- parts. From those documents a bdy-part must be returned (I do not want the whole bdy element, I want something smaller). That part should be about a case study of applying a model checker or a theorem prover to the verification. </narrative>
<keywords>SPIN, SMV, PVS, SPARK, CWB</keywords></inex_topic>

<inex_topic topic_id="72" query_type="CAS" ct_no="54">
<title>//article[about(./fm/au/aff,'United States of America')]/bdy//*[about(.,'weather forecasting systems')]</title>
<description>Find information about weather forecasting systems, from articles written by authors affiliated in the United States of America.</description>
<narrative>I'm only looking for information by authors that are clearly affiliated in the U.S. Therefore I'm only interested in articles where the author's affiliation is mentioned via the /article/fm/au/aff tag. That tag must be referring to an Institute, Company, Reseach Center, and so on, which is located in the US. From those articles I am only interested in parts that are contained somewhere within the body of the document (I do not want the whole bdy element, I want something smaller). That part must be about weather forecasting systems. Parts discussing weather reporting or weather conditions are not considered relevant.</narrative>
<keywords>meteorology, weather prediction</keywords></inex_topic>

<inex_topic topic_id="73" query_type="CAS" ct_no="56">
<title>//article[about(.//st,'+comparison') and about(.//bib,'"machine learning"')]</title>
<description>We are searching for articles containing a section entitled comparison (maybe between systems, methods, works, solutions, etc.) and referencing works (articles) about machine learning. </description>
<narrative>The documents to be considered as relevant are those having section with title about comparison between systems, methods, results, etc. and containing references to articles related to the domain of machine learning within the bibliographic part.</narrative>
<keywords>comparison, machine, learning, evaluation, performance</keywords></inex_topic>

<inex_topic topic_id="74" query_type="CAS" ct_no="75">
<title> //article[about(., 'video streaming applications')]//sec[about(., 'media stream synchronization') OR about(., 'stream delivery protocol')] </title>
<description>I am looking for algorithms for stream delivery in audio and video streaming applications, including audio and video stream synchronization algorithms used for stream delivery. </description>

<narrative>Documents/document components describing delivery control for a specific setting are considered relevant. Document/document components describing audio and video stream synchronization (if this is necessary due to the architecture) are considered relevant. Documents/document components describing architectures or servers for stream delivery (media servers) are relevant, if a part of the document component describes a communication or delivery protocol (even if lowlevel).</narrative>
<keywords>algorithms, audio, video, streaming, applications, stream, delivery, media, stream, synchronization, stream, delivery, protocol</keywords></inex_topic>

<inex_topic topic_id="75" query_type="CAS" ct_no="78">
<title>//article[about(., 'Petri net') AND about(.//sec, 'formal definition') AND about(.//sec, 'algorithm efficiency computation approximation')]</title>
<description>Return articles about Petri nets containing a section with the formal definition of Petri nets and another section discussing algorithms for the exact or approximate computation of properties of a Petri net. </description>
<narrative> The formal definition must include mathematical notation and definitions of symbols in order to be relevant. The section about the efficient computation of Petri nets should describe algorithms, but pseudo-code is not required. The two sections may be the same section. Also, it is implied that the formal definition and the algorithm sections are the formal definition of Petri nets and algorithms for Petri nets. The article does not have to discuss Petri nets theory and computation only, applications of Petri nets are also acceptable, as long as the article has the required sections.
</narrative>
<keywords>petri net, algorithm, computation, efficiency, formal definition, equation, approximation
</keywords></inex_topic>

<inex_topic topic_id="76" query_type="CAS" ct_no="81">
<title>//article[(./fm//yr = '2000' OR ./fm//yr = '1999') AND about(.,'"intelligent transportation system"')]//sec[about(.,'automation +vehicle')]</title>
<description>Automated vehicle applications in articles from 1999 or 2000 about intelligent transportation systems.</description>
<narrative>To be relevant, the target component must be from an article on intelligent transportation systems published in 1999 or 2000 and must include a section which discusses automated vehicle applications, proposed or implemented, in an intelligent transportation system.</narrative>
<keywords>intelligent transportation system, automated vehicle, automobile, application, driving assistance, speed, autonomous driving</keywords></inex_topic>

<inex_topic topic_id="77" query_type="CAS" ct_no="82">
<title>//article[about(.//sec,'"reverse engineering"')]//sec[about(., 'legal') OR about(.,'legislation')]</title>
<description>A section of an article that discusses the legal implications of reverse software engineering either specific to a particular case of reverse engineering or in general.</description>
<narrative>To be relevant, the component must come from an article containing a section on reverse engineering and deal with the legal issues surrounding reverse engineering. Relevant articles may discuss a specific instance of reverse engineering or the topic at large. The target element need not specifically mention reverse engineering as long as it is related to the broader legal issues that encompass reverse engineering.</narrative>
<keywords>reverse engineering, legal, software patent, copyright legislation, fair use, intellectual property rights, licensing</keywords></inex_topic>

<inex_topic topic_id="78" query_type="CAS" ct_no="87">
<title>//vt[about(.,'"Information Retrieval" student')] </title>
<description>Vitae of graduate students researching Information Retrieval</description>
<narrative>Each vita should be about an author who was a graduate student at the time of the article's publication. If a vt field has multiple authors, the vt should be considered relevant if at least one of tha authors is a graduate student studying Information Retrieval.</narrative>
<keywords>Information Retrieval, student, studies, studying, researches, researching</keywords>
</inex_topic>

<inex_topic topic_id="79" query_type="CAS" ct_no="91">
<title>//article[about(.,'XML') AND about(.,'database')]</title>
<description>The articles discussing how to integrate XML and database techniques</description>
<narrative>Return article elements, which talk about the current status of integration of XML into relational database systems, and the problems/challenges associated with this integration.</narrative>
<keywords>relational database, integration, XML</keywords></inex_topic>

<inex_topic topic_id="80" query_type="CAS" ct_no="92">
<title>//article/bdy/sec[about(.,'"clock synchronization" "distributed systems"')]</title>
<description>the sections discussing the problem of clock synchronization in distributed systems
</description>
<narrative>Return section elements, which discuss the problem of clock synchronization in distributed systems. The section can present the algorithm to synchronize clocks or provide a survey of existing algorithms. The section element talking about clock synchronization in digital circuit design is not relevant.</narrative>
<keywords>distributed system, clock synchronization, logical</keywords></inex_topic>

<inex_topic topic_id="81" query_type="CAS" ct_no="93">
<title>//article[about(.//p,'"multi concurrency control"') AND about(.//p, 'algorithm') AND about(./fm//atl, 'databases')]</title>

<description>We are interested in articles that can provide information or reference to information on algorithms for multiversion concurrency control in databases.</description>
<narrative>An article is considered relevant if it provides information about a concurrency control algorithm(s)/protocol(s) based on keeping multiple versions of data items in a database. It can be either a description of the algorithm/protocol or a reference to a paper on the subject in its "Related work" section.</narrative>
<keywords>multiversion, concurrency, control, database, algorithm, protocol, multiple, versions, MV, MV2PL, DBMS, RDBMS, OODBMS</keywords></inex_topic>

<inex_topic topic_id="82" query_type="CAS" ct_no="94">
<title>//article[about(.,'handwriting recognition') AND about(./fm/au,'kim')]</title>
<description>We are looking for articles about handwriting recognition written by someone whose first name or surname is kim.</description>
<narrative>To be relevant, a document must be an article, have handwriting recognition or an example of handwriting recognition as a major subject. One of the authors must have a name similar to 'kim' (kimh or kimm would be accepted, kimbelnatenbaum wouldn't). Kim does not have to be the first author. A bibliographic reference to a relevant article is not a relevant element. An author index or a call for paper is not a relevant element. A correction to a relevant article IS a relevant article, as it is important that the retrieved information comes with its eventual corrections.</narrative>
<keywords>cursive, script, handwriting, recognition, OCR, kim</keywords></inex_topic>

<inex_topic topic_id="83" query_type="CAS" ct_no="104">
<title>/article/fm/abs[about(., '"data mining" "frequent itemset"')]</title>
<description>Find abstracts relevant to data mining which report on using the concept of frequent itemsets. </description>
<narrative>The result of this query provides an overview about work in the field of data-mining which treats frequent itemsets. Frequent itemsets are widely used in data-mining algorithms. A difficulty with this query is that obviously also abstracts about algorithms (such as the apriori algo) are relevant if they apply itemsets. In such a case, the query keywords do not directly appear in the abstract. Nevertheless, the abstract should be considered as relevant.</narrative>
<keywords>frequent itemset,data mining</keywords></inex_topic>

<inex_topic topic_id="84" query_type="CAS" ct_no="105">
<title>//p[about(.,'overview "distributed query processing" join')]</title>
<description>Find paragraphs that give an overview about distributed query processing techniques with a focus on join implementations.</description>
<narrative>Paragraphs from overview articles or related work are of the highest interest since they typically provide a good overview over the field. However, it is seemingly difficult to say when the criterion of 'overview' is fulfilled or not. For my relevance assessment, I found much formalism etc. a good hint for non-relevance (at least to the criterion of overview)</narrative>
<keywords>distributed query processing,join</keywords></inex_topic>

<inex_topic topic_id="85" query_type="CAS" ct_no="106">
<title>//article[./fm//yr >= 1998 and .//fig//no > 9]//sec[about(.//p,'VR "virtual reality" "virtual environment" cyberspace "augmented reality"')]</title>
<description>Find sections about Virtual Reality. Retrieve sections only from articles that were published in or after the year 1998 and have many figures (at least 10).</description>
<narrative>To be relevant a section must have paragraphs that discuss some aspects of VR. More over the section must belong to an article that was published in or after the year 1998 and have more than 9 figures. Sections from articles with less than 10 figures are not counted relevant.</narrative>
<keywords>VR,virtual reality,virtual environment,cyberspace, augmented reality</keywords></inex_topic>

<inex_topic topic_id="86" query_type="CAS" ct_no="107">
<title>//sec[about(.,'mobile electronic payment system')] </title>
<description>Find sections that describe technologies for wireless mobile electronic payment systems at consumer level. </description>
<narrative>To be relevant, a section must describe security-related technologies that exist in electronic payment systems that can be implemented in hardware devices. The main interests are systems that can be used by mobile or handheld devices. A section should be considered irrelevant if it describes systems that are designed to be used in a PC or laptop. </narrative>
<keywords>mobile, electronic payment system, electronic wallets, e-payment, e-cash, wireless, m-commerce, security</keywords></inex_topic>

<inex_topic topic_id="87" query_type="CAS" ct_no="116">
<title>//article[(./fm//yr = '1998' OR ./fm//yr = '1999' OR ./fm//yr = '2000' OR ./fm//yr = '2001' OR ./fm//yr = '2002') AND about(., '"support vector machines"')]</title>
<description>Articles on Support Vector Machines from year 1998 to 2002</description>
<narrative>To be relevant a document must contain information on support vector machines or their applications. A support vector machine is a learning mechanism based on linear programming and is a part of concept learning.</narrative>
<keywords>support vector machines, svm</keywords></inex_topic>

<inex_topic topic_id="88" query_type="CAS" ct_no="121">
<title>//article[(./fm//yr = '1998' OR ./fm//yr = '1999' OR ./fm//yr = '2000' OR ./fm//yr = '2001') AND about(., '"web crawler"')]</title>
<description>Retrieve articles published between 1998 and 2001 about automated programs, web crawlers, meant to search webpages methodically and catalogue them.</description>

<narrative>To be relevant, a document must be published between 1998 and 2001 and contain information regarding various automated search programs, search-engines, searching capabilities and methods. Crawlers that document webpages in databases later used by say search engines.</narrative>
<keywords>web crawler, spider, bot, search-engine</keywords></inex_topic>

<inex_topic topic_id="89" query_type="CAS" ct_no="123">
<title>//article[about(./bdy,'clustering "vector quantization" +fuzzy +k-means +c-means -SOFM -SOM')]
//bm//bb[about(.,'"vector quantization" +fuzzy clustering +k-means +c-means') AND
about(./pdt,'1999') AND ./au/snm != 'kohonen'] </title>
<description>find articles about vector quantization or clustering and return bibliography details of cited publications about clustering and vector quantization methods, from recent years, not authored by Kohonen. </description>
<narrative>Bibliography elements of publications, preferably from around 2000 (1996 to 2002 is fine, descending relevance thereafter). Preferred documents have reference to k-means or c-means clustering. Not interested in publications where the author is Kohonen, or in his work on self organizing feature maps (SOM SOFM). The citing article and the cited publication should be about clustering or vector quantization methods. </narrative>
<keywords>cluster analysis,adaptive clustering,Generalized Lloyd, LBG, GLA</keywords></inex_topic>

<inex_topic topic_id="90" query_type="CAS" ct_no="126">
<title> //article[about(.//sec,'+trust authentication "electronic commerce" e-commerce e-business marketplace')]//abs[about(., 'trust authentication')] </title>
<description>Find abstracts of articles that discuss automated tools for establishing trust between parties on the internet. The article should discuss applications of trust for authenticating parties in e-commerce. </description>
<narrative>We look for tools that can be used to automate trust establishment between entities that have no prior relation on the internet. We don't look for traditional solutions that requires user registration to a site. The article should describe how trust establishment can improve e-commerce. The returned component is the article's abstract which by itself should summarize the trust concept described in the article. </narrative>
<keywords>trust, authentication, electronic commerce, e-commerce, e-business, marketplace</keywords>
</inex_topic>

<inex_topic topic_id="91" query_type="CO" ct_no="7">
<title>Internet traffic</title>
<description>What are the features of Internet traffic?</description>
<narrative>A relevant document/component describes the features of the internet traffic, i.e. it contains information on traffic evolution, its measurement and its possible congestion</narrative>
<keywords>internet, web, traffic, measurement, congestion </keywords></inex_topic>

<inex_topic topic_id="92" query_type="CO" ct_no="10">
<title>"query tightening" "narrow the search" "incremental query answering"</title>
<description> Retrieve articles/components that discuss techniques to deal with the information overload problem in web-based searching when users underspecify queries, in particular query tightening techniques for incremental query answering. </description>
<narrative> Users in web-based searching usually underspecify their queries. Traditional search engines often return much more information than users can digest and most of which are irrelevant. Thus information overload is one of the key challenges in web-based searching. With some preprocessing of the data sources, or by consulting some Knowledge Bases, the original queries can be tightened and search engines can narrow their search such that only relevant answers are returned to users. To be relevant, the document/component should state the problem of query tightening, define the similarities between some concept terms/phrases, and discuss techniques to automatic and intelligent query tightening, which allow users to learn the domain incrementally. </narrative>
<keywords>query tightening,search narrowing,incremental query answering,knowledge base,concept similarities</keywords></inex_topic>

<inex_topic topic_id="93" query_type="CO" ct_no="11">
<title>"Charles Babbage" -institute -inst.</title>
<description>The life and work of Charles Babbage.</description>
<narrative>A relevant document describes (any details of) the life of Charles Babbage or his work.
</narrative>
<keywords>Charles Babbage</keywords>
</inex_topic>

<inex_topic topic_id="94" query_type="CO" ct_no="15">
<title>"hyperlink analysis" +"topic distillation"</title>
<description>How is hyperlink analysis used to retrieve information on the Web and what is topic distillation</description>
<narrative>To be relevant a document must show how useful hyperlinks are to deliver focused results to user queries and must describe how methods such as topic distillation are useful to find good resources in information retrieval</narrative>
<keywords>hyperlink, analysis, topic, distillation, web</keywords></inex_topic>

<inex_topic topic_id="95" query_type="CO" ct_no="17">
<title>+"face recognition" approach</title>
<description>Approaches to face recognition.</description>
<narrative>To be relevant a document has to describe an approach to face recognition.Face

identification, expression recognition or specialized approaches which consider certain circumstances in face recognition are also relevant, given that the approaches are explicitly described and not merely mentioned in summaries, guest editor's introductions or likewise. </narrative>
<keywords>face recognition, approach, method,face identification</keywords></inex_topic>

<inex_topic topic_id="96" query_type="CO" ct_no="18">
<title>+"software cost estimation"</title>
<description>Software cost estimation.</description>
<narrative>To be relevant a document has to deal with methods or approaches concerning software cost estimation in the area of software development. Related approaches to software effort estimation are relevant if the applicability to cost estimation is mentioned in detail. General discussions on effort estimation are not relevant. </narrative>
<keywords>software, cost, estimation</keywords></inex_topic>

<inex_topic topic_id="97" query_type="CO" ct_no="24">
<title>Converting Fortran source code</title>
<description>Documents/components about methods for converting older Fortran programs to C or C++
</description>
<narrative>I am looking for information about the future potential of the programming language FORTRAN. Special emphasis is placed on documents/components discussing pros and cons of conversion from Fortran to C or C++ versus interfacing from Fortran to C or C++ and reverse. </narrative>
<keywords>fortran, converting, conversion, transition, interfacing</keywords></inex_topic>

<inex_topic topic_id="98" query_type="CO" ct_no="26">
<title>"Information Exchange", +"XML", "Information Integration"</title>
<description>How to use XML to solve the information exchange (information integration) problem, especially in heterogeneous data sources? </description>
<narrative>Relevant documents/components must talk about techniques of using XML to solve information exchange (information integration) among heterogeneous data sources where the structures of participating data sources are different although they might use the same ontologies about the same content. </narrative>
<keywords>information exchange, XML, information integration, heterogeneous data sources</keywords>
</inex_topic>

<inex_topic topic_id="99" query_type="CO" ct_no="29">
<title>perl features </title>
<description>Looking for perl features that distinguishes it from other languages</description>
<narrative>To be relevant a document/component must contain information about perl features. Comparison of perl with other programming languages will be marked relevant. Not interested in minute details of languages like syntax etc. </narrative>
<keywords>perl, programming languages</keywords></inex_topic>

<inex_topic topic_id="100" query_type="CO" ct_no="35">
<title>+association +mining +rule +medical</title>
<description> Retrieve information about association rule mining in medical databases </description>
<narrative> We have a medical data mining project and I want to find out how other people have applied association rule mining in this area. A relevant snippet will discuss the application of association rule mining to a medical database. Discussion of association rule mining outside medical databases is not relevant. Discussion of medical databases without association rule mining is also not relevant.
</narrative>
<keywords>association, mining, rule, medical</keywords></inex_topic>

<inex_topic topic_id="101" query_type="CO" ct_no="37">
<title>+"t test" +information </title>
<description>use of the t-test in information retrieval </description>
<narrative>Information retrieval experimenters are advised to compare their new mean-average-precision results with the baseline using a t test. We have reason to believe that this may be bad, even very bad, advice, and are interested in papers that apply the t-tests to information retrieval (and possibly other software engineering tasks). We are also interested in papers that mention alternative statistical techniques to determine significance. </narrative>
<keywords>t-test, information, retrieval </keywords></inex_topic>

<inex_topic topic_id="102" query_type="CO" ct_no="42">
<title>distributed storage systems for grid computing</title>
<description>Distributed storage systems for use in distributed computational grids</description>
<narrative>To be relevant the retrieved elements must discuss systems for use in computational grids that provide distributed data storage. The best matching items will describe these systems, their types, performance and algorithms.</narrative>
<keywords>grid computing, distributed, storage</keywords></inex_topic>

<inex_topic topic_id="103" query_type="CO" ct_no="50">
<title>UML formal logic</title>
<description>Find information on the use of formal logics to model or reason about UML diagrams.
</description>
<narrative>To be relevant, a document/component must discuss the use of formal logics, such as first-order-, temporal-, or description-logics, to model or reason about UML diagrams. "Business-logics" and "Client-logics" do not have a proof system and are therefore not considered

```
to be formal logics.</narrative>
<keywords>unified modeling language, first order logic, temporal logic, description logic</keywords>
</inex_topic>

<inex_topic topic_id="104" query_type="CO" ct_no="52">
<title>Toy Story</title>
<description>Find information on the making of the animated movie Toy Story, discussing the used
techniques, software, or hardware platforms. </description>
<narrative>To be relevant, a document/component must discuss some detail of the techniques or
computer infrastructure used in the creation of the first entirely computer-animated feature-length
movie, "Toy Story." </narrative>
<keywords>Pixar, Disney, Buzz Lightyear, 3D-rendering, Sun, SGI </keywords></inex_topic>

<inex_topic topic_id="105" query_type="CO" ct_no="59">
<title>+"categorization" "textual document" "learning" "evaluation"</title>
<description>We are searching for documents/components dealing with categorization methods for
textual documents, which include learning mechanisms and an evaluation of the proposed solutions.
</description>
<narrative>The documents/components to be considered as relevant are those that discuss categorization
especially for textual documents/data. Furthermore, we are looking for documents/components mentioning
an evaluation method or an evaluation process of the solution proposed in the article/component.
Finally, the relevant documents/components are those that discuss a learning phase or learning
techniques.</narrative>
<keywords>categorization, textual document, learning evaluation, experimentation</keywords>
</inex_topic>

<inex_topic topic_id="106" query_type="CO" ct_no="60">
<title>Content protection schemes</title>
<description>Content protection schemes in relation to the Digital Rights Management concept.
</description>
<narrative>Documents must have focus on at least one technological method of preserving copyright of
digital media, eg. audio and video material.</narrative>
<keywords>content protection, copy protection, media piracy, digital rights management</keywords>
</inex_topic>

<inex_topic topic_id="107" query_type="CO" ct_no="61">
<title>"artificial intelligence" AI practical application industry "real world" </title>
<description>Retrieve articles or sections of articles that discuss the practical application of
Artificial Intelligence. </description>
<narrative>To be relevant, a document/component has to discuss a practical application of algorithms
or software in the field of Artificial Intelligence like the use of expert systems, application of
genetic algorithms or fuzzy logic... Documents/components that describe only the algorithm or
formulas are irrelevant. There has to be a connection to the real world.</narrative>
<keywords>artificial intelligence, AI practical application industry real world, genetic
algorithm,fuzzy logic</keywords></inex_topic>

<inex_topic topic_id="108" query_type="CO" ct_no="63">
<title>ontology ontologies overview "how to" practical example</title>
<description>Retrieve articles or sections of articles that introduce ontologies or discuss
ontologies with a practical application.</description>
<narrative>To be relevant, a document/component has to discuss an introduction or practical
application of ontologies. Documents about the building of ontologies are irrelevant.</narrative>
<keywords>ontology, ontologies, overview, how to, practical example</keywords></inex_topic>

<inex_topic topic_id="109" query_type="CO" ct_no="65">
<title>"CPU cooling" "cooling fan design" "heatsink design" "heat dissipation" airflow casing
</title>
<description>I am looking for techniques for CPU cooling, especially dealing with cooling fan
and heatsink design as well as airflow inside the casing.</description>
<narrative>Documents/document components describing measures and techniques for cooling in the
context of wearable computing are considered as not relevant. Documents/document components
describing specific system and/or hardware examples are considered relevant. Documents/document
components describing CPU operation optimization for conserving energy usage are not relevant.
Documents/document components describing passive cooling techniques are considered as not relevant.
Documents/document components describing fan and heatsink design (such as fan blade specifics)
are considered relevant.</narrative>
<keywords>optimal, cooling, CPU, cooling, fan, design, heatsink, design, airflow, casing,
computing, heat, dissipation, heat, generation</keywords> </inex_topic>

<inex_topic topic_id="110" query_type="CO" ct_no="66">
<title>"stream delivery" "stream synchronization" audio video streaming applications </title>
<description>I am looking for algorithms for stream delivery in audio and video streaming
applications, including audio and video stream synchronization algorithms used for stream
delivery.</description>
<narrative>Documents/document components describing delivery control for a specific setting are
considered relevant. Document/document components describing audio and video stream
synchronization (if his is necessary due to the architecture) are considered relevant.
```

```
</narrative>
<keywords>algorithms, audio, video, streaming, applications, stream, delivery, media, stream,
synchronization, stream, delivery, protocol</keywords></inex_topic>

<inex_topic topic_id="111" query_type="CO" ct_no="69">
<title>"natural language processing" -"programming language" -"modeling language" +"human
language"</title>
<description>Natural language processing methods and applications.</description>
<narrative>In order to be relevant, a document or component must contain information about one
or several methods or applications related to the field of natural language processing.
Alternatively, it may also contain a definition of the field of natural language processing.
</narrative>
<keywords>natural language processing, human language technology, computational linguistics,
speech processing, parsing natural language, natural language understanding, natural language
interface </keywords>
</inex_topic>

<inex_topic topic_id="112" query_type="CO" ct_no="71">
<title>+"Cascading Style Sheets" -"Content Scrambling System"</title>
<description> References to World Wide Web Consortium(W3C)'s Cascading Style Sheets(CSS) to meet
design and style requirements of information. </description>
<narrative> In order to be relevant, the result must simply contain any reference to the World Wide
Web Consortium(W3C) endorsed Cascading Style Sheet specification. An explanation of the concept of
separating style and content with CSS is especially relevant. We are not interested in
Content-Scrambling Systems (CSS), which are encryption systems used on some DVDs. </narrative>
<keywords>css, style, design, standards, w3c, web, internet, www, html, xml, xhtml</keywords>
</inex_topic>

<inex_topic topic_id="113" query_type="CO" ct_no="72">
<title>"Markov models" "user behaviour" </title>
<description>Find text about the use of Markov models to model user behaviour. Additional
state-based stochastic processes that are generalizations of Markov models used to model human
patterns of behaviour or interaction are also relevant. </description>
<narrative>Relevant text should be about modeling a user's action or movements over time. Models
of user interaction with a computer are preferable, but models of general user behaviour such as
gestures or actions are also acceptable. The use of markov or related models for static recognition
tasks such as handwriting recognition or face detection are not relevant. Text about of user load
on distributed systems are not relevant. </narrative>
<keywords>markov model, user interaction, human behaviour, graphical, stochastic, sequence</keywords>
</inex_topic>

<inex_topic topic_id="114" query_type="CO" ct_no="84">
<title>+women "history of computing"</title>
<description>Women's role in the history of computing</description>
<narrative>To be relevant, the document/component must reference an individual or group of women
and discuss her/their place in computing history. The individual or group's contributions should
ideally be noted in relation to the impact, large or small, of their efforts. </narrative>
<keywords>women, history of computing, gender, Grace Hopper, Ada Lovelace, contribution</keywords>
</inex_topic>

<inex_topic topic_id="115" query_type="CO" ct_no="86">
<title> +"IP telephony" +challenges</title>
<description> Challenges in the IP telephony industry</description>
<narrative>To be relevant, the document/component must discuss some challenge in the evolution of
IP telephony. The document/component should name a specific problem, whether technological or
social, that IP telephony has faced or currently faces. </narrative>
<keywords>IP telephony, Internet, telephone, web, problem, challenge, evolution, prevent, advance
</keywords></inex_topic>

<inex_topic topic_id="116" query_type="CO" ct_no="95">
<title>"computer assisted art" "computer generated art"</title>
<description>How can computer systems create or help creating art</description>
<narrative>To be relevant, a document/document component must explain how computer systems can
improve or replace the creative process in arts, including music, graphic arts, etc. or give
examples of such assistance.</narrative>
<keywords>art, drawing, graphics, music, computer assisted art, computer generated art, art
simulation</keywords></inex_topic>

<inex_topic topic_id="117" query_type="CO" ct_no="98">
<title>Patricia Tries </title>
<description>Find documents/elements that describe Patricia tries and their use.</description>
<narrative>To be relevant, a document/element must deal with the use of Patricia Tries for text
search. Description of the standard algorithm, optimisied implementation and use in Information
retrieval applications are all relevant. </narrative>
<keywords>Patricia tries, tries, text search, string search algorithm, string pattern matching
</keywords></inex_topic>
```

<inex_topic topic_id="118" query_type="CO" ct_no="101">
<title>"shared nothing" database</title>
<description>Retrieve all document components about (distributed) database systems implemented using a shared-nothing architecture</description>
<narrative>Document components that report on anc investigation of shared-nothing architecture are relevant. While looking through the articles, some used the notion of shared-nothing only as a means of differentiation, i.e., those paper actually are on shared-disk, shared-memory etc. architectures and shared-nothing appears in related work, intro etc. and should not be considered as relevant</narrative>
<keywords>shared nothing,distributed database systems</keywords></inex_topic>

<inex_topic topic_id="119" query_type="CO" ct_no="108">
<title>Optimizing joins in relational databases.</title>
<description>Articles/components in the area of query optimization on the specific topic of optimizing joins in relational databases.</description>
<narrative>We would like to find articles/components in the area of query optimization on the specific topic of optimizing joins in relational databases. The articles/components describe algorithms such as reordering joins. </narrative>
<keywords>query optimization, join, relational database</keywords></inex_topic>

<inex_topic topic_id="120" query_type="CO" ct_no="31">
<title>information retrieval models</title>
<description>Documents/components about information retrieval models.</description>
<narrative>To be relevant a document/component should discuss any retrieval models, such as the boolean, vector space, probablistic, set theoretic, or fuzzy model, etc. </narrative>
<keywords>boolean model, vector space model, probablistic model, set theoretic model, Extended Boolean model, fuzzy model, statistical model,information retrieval techniques</keywords>
</inex_topic>

<inex_topic topic_id="121" query_type="CO" ct_no="117">
<title>Real Time Operating Systems</title>
<description>Operating systems reacting to inputs in real time</description>
<narrative>To be relevant, a document must explain and/or give examples of real-time operating systems along with some applications</narrative>
<keywords>real-time, operating systems, embedded systems, critical systems, rtos</keywords>
</inex_topic>

<inex_topic topic_id="122" query_type="CO" ct_no="118">
<title>Lossy Compression Algorithm</title>
<description>Compression algorithms which are lossy in nature</description>
<narrative>To be relevant, a document must talk about lossy forms of compression of any data e.g. images</narrative>
<keywords>lossy compression algorithm, image compression, jpeg, mp3</keywords></inex_topic>

<inex_topic topic_id="123" query_type="CO" ct_no="122">
<title>multidimensional index "nearest neighbour search"</title>
<description>What multidimensional indices and index structures are used for nearest neighbour search?</description>
<narrative>To be relevant, a document or component must contain information about databases using multidimensional indices such as R-tree, X-tree, TV-tree, and so on. It may contain information about nearest neighbour search.</narrative>
<keywords>multidimension, dimension, index, structure, tree, nearest, neighbour, search</keywords>
</inex_topic>

<inex_topic topic_id="124" query_type="CO" ct_no="125">
<title>application algorithm ??? means means "vector quantization" "speech compression" "image compression" "video compression" </title>
<description>find elements about clustering procedures, particularly k-means, aka c-means, aka Generalized Lloyd algorithm, aka LBG, and their application to image speech and video compression.
</description>
<narrative>interested only in application, and or the algorithm or procedure applied to speech video or image compression. </narrative>
<keywords>vector, quantization, VQ, LVQ,Generalized Lloyd, GLA, LBG, cluster analysis, clustering, image compression, video compression, speech compression</keywords></inex_topic>

<inex_topic topic_id="125" query_type="CO" ct_no="127">
<title>+wearable ubiquitous mobile computing devices </title>
<description>Wearable computing devices.</description>
<narrative>To be relevant, a document or component must contain information about wearable computers, and it may contain information about mobile devices or ubiquitous computers, such as head mounted display, Cellular phone, RFID and IrDA.</narrative>
<keywords>ubiquitous, device, wearable, mobile, equipment, GPS, augment, reality</keywords>
</inex_topic>

<inex_topic topic_id="126" query_type="CO" ct_no="25">
<title>Open standards for digital video in distance learning</title>
<description>Open technologies behind media streaming in distance learning projects</description>

<narrative> I am looking for articles/components discussing methodologies of digital video production and distribution that respect free access to media content through internet or via CD-Roms or DVDs in connection to the learning process. Discussions of open versus proprietary standards of storing and sending digital video will be appreciated. </narrative>
<keywords>media streaming,video streaming,audio streaming, digital video,distance learning,open standards,free access</keywords></inex_topic>

Literaturverzeichnis

[ABS00] S. Abiteboul, P. Buneman, D. Suciu: Data on the Web – From Relations to Semistructured Data and XML.
San Francisco: Morgan Kaufmann Publishers, 2000.

[ABS04] S. Amer–Yahia, C. Botev, J. Shanmugasundaram: TeXQuery: A Full–Text Search Extension to XQuery.
In: Proceedings of the 13th International World Wide Web Conference (WWW), Manhattaen, NY, USA, May 2004.

[ACS02] S. Amer–Yahia, S. Cho, D. Srivastava: Tree Pattern Relaxation.
In: Proceedings of the 8th International Conference on Extending Database Technology (EDBT) 2002, pages 496–513, Prague, Czech Republic, March 2002.

[AKM01] S. Abiteboul, H. Kaplan, T. Milo: Compact labeling schemes for ancestor queries.
In: Proceedings of the 12nd Annual ACM/SIAM Symposium on Discrete Algorithms (SODA), pages 547–556, Washington, DC, USA, January 2001.

[AL02] L. Ahmedi, G. Lausen: Ontology-Based Querying of Linked XML Documents.
In: Proceedings of the 2nd International Workshop on the Semantic Web (SemWeb) 2001, Hongkong, China, May 2001.

[ANSI92] American National Standards Institute: American National Standard for Information Systems – Database Language – SQL.
ANSI (includes ANSI X3.168–1989, formerly ANSI X3.135–1992 (R1998)), 1992.

[AQM+97] S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. L. Wiener: The Lorel Query Language for Semistructured Data.
International Journal on Digital Libraries 1 (1), pages 68–88, 1997.

[Ave95] J. Avenhaus: Reduktionssysteme - Rechnen und Schließen in gleichungsdefinierten Strukturen.
Springer–Verlag: Berlin, 1995.

[AR02] S. Alstrup, T. Rauhe: Improved labeling scheme for ancestor queries.
In: Proceedings of the 13th Annual ACM/SIAM Symposium on Discrete Algorithms (SODA), pages 947–953, San Francisco, CA, USA, January 2002.

[BBC+03] A. Berglung, S. Boag, D. Chamberlin, M. F. Fernandez, M. Kay, J. Robie, J. Simeon: XML Path Language (XPath) Version 2.0.
http://www.w3.org/TR/xpath20, W3C Working Draft 12 November 2003.

[BCF+03] S. Boag, D. Chamberlin, M.F. Fernandez, D. Florescu, J. Robie, J. Simon
XQuery 1.0: An XML Query Language.
http://www.w3.org/TR/xquery, W3C Working Draft 12 November 2003.

[BDB+00] I. Bruder, A. Düsterhöft, M. Becker, J. Bedersdorfer, G. Neumann: GETESS: Constructing a Linguistic Search Engine Index for an Internet Search Engine.
In: Proceedings of the 5th International Conference on Applications of Natural Language to Databases (NLDB) 2000, pages 227–238, Versailles, France, 2000.

[BEH+01] E. Bozsak, et. al.: KAON - Towards a large scale Semantic Web.
In: Proceedings of 3rd ACM Conference on Electronic Commerce (EC) 2001, Tampa, FL, USA, October 2001.

[Ber03] A. Berglund: Extensible Stylesheet Language (XSL) Version 1.1.
http://www.w3.org/TR/2003/WD-xsl11-20031217/, W3C Working Draft 17 December 2003.

[BFL98] C.F. Baker, C.J. Fillmore, J.B. Lowe: The Berkeley FrameNet Projekt.
In: Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (COLIN–ACL), Montreal, Canada, August 1998.

[BGK03] P. Buneman, M. Grohe, C. Koch: Path Queries on Compressed XML.
In: Proceedings of the 29th International Very Large Database Connference (VLDB), pages 141–152, Berlin, Germany, September 2003.

[BGS+03] H. Blanken, T. Grabs, H.-J. Schek, R. Schenkel, G. Weikum (Eds.): Intelligent Search on XML Data.
Springer–Verlag: Berlin, 2003.

[BH01] A. Budanitsky, G. Hirst: Semantic distance in WordNet: An experimental, application–oriented evaluation of five measures.
In: Proceedings of the Workshop on WordNet and other Lexiacal Resources, in the North American Chapter of the Association for Computational Lingustics (NAACL) 2001, Pittsburgh, PA, USA, June 2001.

[BHL99] T. Bray, D. Hollander, A. Layman: Namespaces in XML.
http://www.w3.org/TR/REC-xml-names/, W3C Recommendation 14 January 1999.

[BHP94] M.W. Bright, A.R. Hurson, S. Pakzad: Automated Resolution of Semantic Heterogeneity in Multidatabases.
ACM Transactions on Database Systems, 19(2) 1994, pages 212-253.

[BIFM98] T. Berners-Lee, U.C. Irvine, R. Fielding, L. Masinter: Uniform Resource Identifiers (URI): Generic Syntax.
http://www.ietf.org/rfc/rfc2396.txt, Internet Society, August 1998.

[BKS02] N. Bruno, N. Koudas, D. Srivastava: Holistic Twig Joins: Optimal XML Pattern Matching.
In: Proceedings of the ACM International Conference on Management of Data (SIGMOD), pages 310–321, Madison, Wisconsin, USA, June 2002.

[BLL+02] S. Bressan, M. L. Lee, Y. G. Li, Z. Lacroix, U. Nambiar: The XOO7 Benchmark.
In: Proceedings of Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web, VLDB 2002 Workshop EEXTT and CAiSE 2002 Workshop DIWeb, pages 146–147, Hongkong, China, 2002

[BM99] T. Bench-Capon, G. Malcolm: Formalising Ontologies and Their Relations.
In: Proceedings of the 10th International Conference on Database and Expert Systems (DEXA) 1999, pages 250-259, Florence, Italy, 1999.

[BM01] P. V. Biron, A. Malhotra: XML Schema Part 2: Datatypes.
http://www.w3.org/TR/xmlschema-2/, W3C Recommendation 02 May 2001.

[Bos99] J. Bosak: XML Examples (The Plays of Shakespeare 2.0).
http://www.ibiblio.org/bosak/, 1999.

[BPS+04] T. Bray, J. Paoli, C. M. Sperberg–McQueen, E. Maler, F. Yergeau, J. Cowan: Extensible Markup Language (XML) 1.1.
http://www.w3.org/TR/REC-xml11-20040204/, W3C Recommendation 04 February 2004.

[BR99] R. Baeza–Yates, B. Ribeiro–Neto: Modern Information Retrieval.
Addison–Wesley: New York, 1999.

[BR01] T. Böhme, E. Rahm: XMach–1: A Benchmark for XML Data Management.
In: A. Heuer, F. Leymann, D. Priebe (Eds.): Datenbanksysteme in Büro, Technik und Wissenschaft (BTW) 2001. 9. GI–Fachtagung, Oldenburg, März 2001.
In: Informatik aktuell, pages 264–273, Berlin: Springer, 2001.

[BR02] T. Böhme, E. Rahm: Multi–User Evaluation of XML Data Management Systems with XMach–1.
In: Proceedings of the International Workshop on Efficiency and Effectiveness of XML Tools and Techniques (EEXTT) in conjunction with VLDB 2002 / CAiSE 2002, pages 148–158, Hongkong, China, August 2002.

[CBB+97] R.G.G. Cattell, D.K. Barry, D. Bartels, M. Berler, J. Eastman, S. Gamerman, D. Jordan, A. Springer, H. Strickland, D. Wade (Eds.): The Object Database Standard: ODMG 2.0.
Morgan Kaufmann: San Francisco, 1997.

[CCD+98] S. Ceri, S. Comai, E. Damiani, P. Fratenali, S. Paraboschi, L. Tanca: XML–GL: A Graphical Language for Querying and Reshaping XML Documents.
http://www.w3.org/TandS/QL/QL98/pp/xml-gl.html, 1998.

[CDN93] M. J. Carey, D. J. DeWitt, J. F. Naughton: The OO7 Benchmark.
In: Proceedings of the International Conference on Management of Data (SIGMOD), pages 12–21, Washington D.C., USA, 1993.

[Cha03] S. Chakrabarti: Mining the Web.
Morgan Kaufmann Publishers: San Francisco, 2003.

[CHH+01] D. Connolly, F. van Harmelen, I. Horrocks, D. McGuinness, P. F. Patel–Schneider, L. A. Stein: Annotated DAML+OIL Ontology Markup.
http://www.w3.org/TR/daml+oil-walkthru/, W3C Note 18 December 2001.

[CHKZ02] E. Cohen, E. Halperin, H. Kaplan, U. Zwick: Reachability and distance queries via 2-hop labels.
In: Proceedings of the 13th Annual ACM/SIAM Symposium on Discrete Algorithms (SODA), pages 937-946, San Francisco, CA, USA, January 2002.

[CK02] T. T. Chinenyanga, N. Kushmerick: An Expressive and Efficient Language for Information Retrieval.
In: Journal of the American Society for Information Science and Technology, 53(6): 438–453, 2002.

[CKK+02] S. Cohen, Y. Kanza, Y. Kogan, Y. Sagiv: EquiX – A Search and Query Language for XML.
In: Journal of the American Society for Information Science and Technology, 53(6): 454–466, 2002.

[CKM02] E. Cohen, H. Kaplan, T. Milo: Labeling Dynamic XML Trees.
In: Proceedings of the 21st Symposium on Principles of Database Systems 2002 (PODS), pages 271–281, Madison, Wisconsin, USA, June 2002.

[CLO03] Q. Chen, A. Lim, K. W. Ong: D(k)–Index: An Adaptive Structural Summary for Graph–Structured Data.
In: Proceedings of the International Conference on Management of Data (SIGMOD) 2003, pages 1133–144, San Diego, California, USA, June 2003.

[CLR90] T. H. Cormen, C. E. Leiserson, R. L. Rivest: Introduction to Algorithms.
MIT Press, 1990.

[CMBT02] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan: GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications.
In: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL) 2002, Philadelphia, PA, USA, July 2002.

[CMKS03] S. Cohen, J. Mamou, Y. Kanza, Y. Sagiv: XSEarch: A Semantic Search Engine for XML.
In: Proceedings of the 29th International Very Large Database Connference (VLDB), pages 45–56, Berlin, Germany, September 2003.

[CMPB04] Y. Chen, G. Mihaila, S. Padmanabhan, R. Bordawekar: L–Tree: A Dynamic Labeling Structure for Ordered XML Data.
In. Proceedings of the International Workshop on Database Technolgies for Handling XML Information on the Web (DataX) 2004, pages 31–45, in conjunction with EDBT 2004, Heraklion, Crete, Greece, March 2004.

[CMS02] C.–W. Chung, J.–K. Min, K. Shim: APEX: An adaptive path index for XML data.
In: Proceedings of the International Conference on Management of Data (SIGMOD) 2002, pages 121–132, Madison, Wisconsin, USA, June 2002.

[Coh00] W. W. Cohen: WHIRL: A word–based information representation language.
In: Artificial Intelligence 118(1–2): 163–196, 2000.

[CRF00] D. Chamberlin, J. robie, D. Florescu: Quilt: An XML Query Language for Heterogeneous Data Sources.

In: Proceedings of the 3rd International Workshop on the Web and Databases (WebDB) 2000, pages 53–62, Dallas, Texas, USA, May 2000.

[CSF+01] B. F. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason, M. Shadmon: A Fast Index for Semistructured Data.
In: Proceedings of the 23rd Very Large Databases Conference 2001 (VLDB), pages 341–351, Roma, Italy, 2001.

[CS00] S. Cluet, J. Simeon: YATL: a Functional and Declarative Language for XML. Technical Report, May 2000.

[DEFS99] S. Decker, M. Erdmann, D. Fensel, R. Studer: OntoBroker: Ontology based Access to Dstributed and Semi-Structured Information.
In: Proceedings of the 8th Working Conference on Database Semantics (DS) 1999, pages 351-369, Rotorua, New Zealand, 1999.

[DFF+98] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Suciu: XML-QL: A Query Language for XML.
http://www.w3.org/TR/NOTE-xml-ql/, Submission to the W3C, 19.08.1998.

[Dij59] E. W. Dijkstra: A note on two problems in connexion with graphs.
Numerische Mathematik, 1:269-271, 1959.

[Enc01] Microsoft Encarta Reference Suite 2001.
Microsoft Corporation, http://www.microsoft.com/encarta, 2001.

[DMO01] S. DeRose, E. Maler, D. Orchard: XML Linking Language (XLink) Version 1.0.
http://www.w3.org/TR/xlink, W3C Recommendation 27 June 2001.

[ESA+04] T. Eda, Y. Sakurai, T. Amagasa, M. Yoshikawa, S. Uemura, T. Honishi: Dynamic Range Labeling for XML Trees.
In. Proceedings of the International Workshop on Database Technolgies for Handling XML Information on the Web (DataX) 2004, pages 61–75, in conjunction with EDBT 2004, Heraklion, Crete, Greece, March 2004.

[Fall01] D. C. Fallside: XML Schema Part 0: Primer.
http://www.w3.org/TR/xmlschema-0, W3C Recommendation 2 May 2001.

[Fell98] C. Fellbaum (ed.): WordNet: An Electronic Lexical Database.
MIT Press, 1998.

[FFR96] A. Farquhar, R. Fikes, J. Rice: The Ontolingua Server: a Tool for Collaborative Ontology Construction.
Technical Report KSL-96-26, http://www-ksl-svc.stanford.edu:5915/project-papers.html, 1996.

[FG01] N. Fuhr, K. Großjohann: XIRQL: A Query Language for Information Retrieval in XML Documents.
In: Proceedings of the 24th International ACM Conference on Research and Development in Information Retrieval (SIGIR), pages 172–180, New Orleans, L, USA, September 2001.

[FGK03] N. Fuhr, K. Großjohann, S. Kriewel: A Query Language and User Interface for XML Information Retrieval.
In: H. Blanken, T. Grabs, H.-J. Schek, R. Schenkel, G. Weikum (Eds.): Intelligent Search on XML Data.
Pages 59–75, Springer–Verlag: Berlin, 2003.

[FHLW03] D. Fensel, J. A. Hendler, H. Lieberman, W. Wahlster: Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential.
MIT Press, 2003.

[Flo03] H. Flörke: The SearX–Engine at INEX'03: XML enabled probabilistic retrieval.
In: Proceedings of the 2nd Workshop of the Initiative for the Evaluation of the XML Retrieval (INEX), page 57, Dagstuhl, Germany, December 2003.

[GJK+02] S. Guha, H.V. Jagadish, N. Koudas, D. Srivastava, T. Yu: Approximate XML Joins.
In: Proceedings of the 21st International ACM Conference on Management of Data (SIGMOD), pages 287–298, Madison, Wisconsin, USA, June 2002.

[GMMW03] P. Grosso, E. Maler, J. Marsh, N. Walsh: XPointer Framework.
http://www.w3.org/TR/xptr-framework, W3C Recommendation 25 March 2003.

[Gray91] J. Gray: Database and Transaction Processing Performance Handbook.
Morgan Kaufmann, San Mateo, CA, USA, 1991.

[GRGK97] V. N. Gudivada, V. V. Raghavan, W. I. Grosky, R. Kasanagottu: Information Retrieval on the World Wide Web.
In: IEEE Internet Computing, 1(5): 58-68, 1997.

[Gru93] T.R. Gruber: Towards Principles for the Design of Ontologies used for Knowledge Sharing.
In: Proceedings of the International Workshop on Formal Ontology, 1993.

[Gru02] T. Grust: Accelerating XPath Location Steps.
In: Proceedings of the 21st International ACM Conference on Management of Data (SIGMOD), pages 109-120, Madison, Wisconsin, USA, June 2002.

[GSBS03] L. Guo, F. Shao, C. Botev, J. Shanmugasundaram: XRANK: Ranked Keyword Search over XML Documents.
In: Proceedings of the 22nd International ACM Conference on Management of Data (SIGMOD), pages 16–27, San Diego, California, USA, June 2003.

[Gua98] N. Guarino: Formal Ontology in Information Systems.
In: Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS) 1998, pages 3-15, Trento, Italy, June 1998.

[GW97] R. Goldman, J. Widom: DataGuides: Enabling query formulation and optimization in semistructured databases.
In: Proceedings of the 23rd Very Large Databases Conference 1997 (VLDB), pages 436–445, Athen, Greece, 1997.

[Har95] F. Harary: Graph Theory.
Addison Wesley, 1995.

[Hef04] J. Heflin: OWL Web Ontology Language: Use Cases and Requirements. http://www.w3.org/TR/2004/REC-webont-req-20040210/, W3C Recommendation 10 February 2004.

[HNVV03] M. Halkidi, B. Ngyuen, I. Varlamis, M. Vazirgiannis: THESUS: Organizing Web document collections based on link semantics. In: Very Large Databases (VLDB) Journal 12(4): 320-332, 2003.

[Hor02] I. Horrocks: DAML+OIL: A Reasonable Web Ontology Language. In: Proceedings of the 8th International Conference on Extending Database Technology (EDBT) 2002, pp. 2-13., Prague, Czech Republic, March 2002.

[HPH03] I. Horrocks, P. F. Patel–Schneider, F. van Harmelen: From SHIQ and RDF to OWL: The making of a web ontology language. In: Journal of Web Semantics 1(1): 7–26, 2003.

[INEX] Initiative for the Evaluation of XML Retrieval (INEX). http://www.is.informatik.uni-duisburg.de/projects/inex.

[INEX02] Initiative for the Evaluation of XML Retrieval. http://qmir.dcs.qmul.ac.uk/inex/index.html

[INEX03] Initiative for the Evaluation of XML Retrieval. http://inex.is.informatik.uni-duisburg.de:2003/

[ISO8879] ISO 8879: Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML). http://www.iso.ch/iso/en/ISOOnline.openerpage, 1986.

[Java] Java 2 Platform, Standard Edition v 1.4.2 (J2SE) Sun Microsystems Inc., http://java.sun.com/

[JLS+04] H.V. Jagadish, L.V.S. Lakshmanan, M. Scannapieco, D. Srivastava, N. Wiwatwattana: Colorful XML: One Hierarchy Isn't Enough. In: Proceedings of the International ACM Conference on Management of Data (SIGMOD), Paris, France, June 2004.

[JWLY03] H. Jiang, W. Wang, H. Lu, J. X. Yu: Holistic Twig Joins on Indexed XML Documents. In: Proceedings of the 29th International Very Large Database Connference (VLDB), pages 273–284, Berlin, Germany, September 2003.

[JWNL] J. Didion: Java WordNet Library http://sourceforge.net/projects/jwordnet

[Kay03] M. Kay: XSL Transformations (XSLT) Version 2.0 http://www.w3.org/TR/2003/WD-xslt20-20031112/, W3C Working Draft 12 November 2003.

[KBNK02] R. Kaushik, P. Bohannon, J. F. Naughton, H. F. Korth: Covering Indexes for Branching Path Queries. In: Proceedings of the 21st International ACM Conference on Management of Data (SIGMOD), pages 133–144, Madison, Wisconsin, USA, June 2002.

[KI90] H. Kimoto, T. Iwadera: Construction of a Dynamic Thesaurus and its Use for Associated Information Retrieval. In: Proceedings of the SIGIR Conf. 1990, pp. 227-240.

[KKC94] O. Kwon, M.-C. Kim, K.-S. Choi: Query Expansion Using Domain Adapted, Weighted Thesaurus in an Extended Boolean Model.
In: Proceedings of the CIKM Conf. 1994, pp. 140-146.

[KKNR04] R. Kaushik, R. Krishnamurthy, J. F. Naughton, R. Ramakrishnan: On the Integration of Structure Indexes and Inverted Lists.
In: Proceedings of the International ACM Conference on Management of Data (SIGMOD), Paris, France, June 2004.

[KKS01] L. Kerschberg, W. Kim, A. Scime: A Semantic Taxonomy-Based Personalizable Meta- Search Agent.
In: Proceedings of the WISE Conf. 2001.

[KM01] H. Kaplan, T. Milo: Short and simple labels for small distances and other functions.
In: Proceedings of the 7th International Workshop on Algorithms and Data Structures (WADS) 2001, Providence, RI, USA, 2001, pages 246–257.

[KM03] M. Klettke, H. Meyer: XML & Datenbanken – Konzepte, Sprachen und Systeme.
Heidelberg: dpunkt.verlag GmbH, 2003.

[KMS02] H. Kaplan, T. Milo, R. Shabo: A comparison of labeling schemes for ancestor queries.
In: Proceedings of the 13th Symposium on Discrete Algorithms 2002 (SODA), pages 954–963, San Francisco, California, USA, 2002.

[KPS04] M. Kratky, J. Pokorny, V. Snasel: Implementation of XPath Axes in the Muli–Dimensional Approach to Indexing XML Data.
In. Proceedings of the International Workshop on Database Technolgies for Handling XML Information on the Web (DataX) 2004, pages 46–60, in conjunction with EDBT 2004, Heraklion, Crete, Greece, March 2004.

[Lan92] H. Langendörfer: leistungsanalyse von Rechensystemen – Messen, Modellieren, Simulation.
München, Wien: Carl Hanser Verlag, 1992.

[LG90] D. Lenat, R.V. Guha: Building Large Knowledge Based Systems.
Addison-Wesley, 1990.

[LJ99] A. Le Hors, I. Jacobs: HTML 4.01 Specification.
http://www.w3.org/TR/html4/, W3C Recommendation 24 December 1999.

[LLD+02] R. W. P. Luk, H. V. Leong, T. S. Dillon, A. T. S. Chan, W. B. Croft, J. Allan: A Survey in Indexing and Searching XML Documents.
In: Journal of the American Society for Information Science and Technology (JASIST), 53(6): 415–437, 2002.

[LM01] Q. Li, B. Moon: Indexing and Querying XML Data for Regular Path Expressions.
In: Proceedings of the 27th International Very Large Database Connference (VLDB), pages 361–370, Roma, Italy, September 2001.

[LSN+00] Q. Li, P. Shilane, N. F. Noy, M. A. Musen: Ontology Acquisition from Online Knowledge Sources.
In: Proceedings of the AMIA Annual Symposium 2000.

[Microsoft] Windows XP Professional
Microsoft Corporation: http://www.microsoft.com/

[MS99] T. Milo, D. Suciu: Index Structures for Path Expressions.
In: Proceedings of the 7th International Conference on Database Theory (ICDT) 1999, pages 277–295, Jerusalem, Israel, 1999.

[MS00] A. Maedche, S. Staab: Semi-automatic Engineering of Ontologies from Text.
In: Proceedings of the 12th SEKE Conf. 2000.

[MS01a] A. Maedche, S. Staab: Learning Ontologies for the Semantic Web.
In: Proceedings of the 2nd International Workshop on the Semantic Web (SemWeb) 2001, Hongkong, China, May 2001.

[MS01] C. D. Manning, H. Schütze: Foundations of statistical natural language processing.
London: MIT Press, 2001.

[MWK00] P. Mitra, G. Wiederhold, M.L. Kersten: Articulation of Ontology Interdependencies Using a Graph-Oriented Approach.
In: Proceedings of the 7th EDBT Conf., Constance, Germany, 2000.

[NFM00] N. F. Noy, R. W. Fergerson, M. A. Musen: The knowledge model of Protege–2000: combining interoperaibility and flexibility.
In: Proceedings of the 12th International Conference on Knowledge Acquisition, Modeling and Management (EKAW) 2000, France, October, 2000, pages 17–32.

[NLB+01] U. Nambiar, Z. Lacroix, S. Bressan, M. L. Lee, Y. G. Li: XML Benchmarks put to the Test.
In: Proceedings of the 3rd International Conference on Information Integration and Web–based Applications and Services (IIWAS), Linz, Austria, September 2001.

[NLB+02] U. Nambiar, Z. Lacroix, S. Bressan, M. L. Lee, Y. G. Li: Efficient XML Data Management: An Analysis.
In: Proceedings of the 3rd International Conference on E–Commerce and Web Technologies (EC–Web), pages 87–98, France, September 2002.

[Oracle] Oracle 9i Release 2 (9.2.0.1)
Oracle Corporation: http://www.oracle/com/

[Pae96] M. Paetau: Informationsgesellschaft, Kommunikationsweise und sozialer Raum. Soziale Betrachtung der Entkopplung von Kommunikation.
In: Forum Wissenschaft, BdWi e.V., pp. 29–32, 13(1) 1996.

[PM98] M.P. Papazoglou, S. Milliner: Subject-based Organization of the Information Space in Multi-Database Networks.
In: Proceedings of the 10th CAiSE Conf. 1998.

[RLJ99] D. Raggett, A. Le Hors, I. Jacobs: HTML 4.01 Specification.
http://www.w3.org/TR/html4

[RLS98] J. Robie, J. Lapp, D. Schach: XML Query Language (XQL).
http://www.w3.org/TandS/QL/QL98/pp/xql.html.

[RM04] P. Rao, B. Moon: PRIX: Indexing And Querying XML using Prüfer Sequences.
In: Proceedings of the 20th International Conference on Data Engineering (ICDE) 2004, Boston, MA, USA, March 2004.

[RN95] S. Russel, P. Norvig: Artificial Intelligence - A Modern Approach.
Prentice Hall, 1995

[RPJA02] K. Runapongsa, J. M. Patel, H.V. Jagadish, S. Al–Khalifa: The Michigan Benchmark: A Microbenchmark for XML Query Processing Systems.
In: Proceedings of the International Workshop on Efficiency and Effectiveness of XML Tools and Techniques (EEXTT) in conjunction with VLDB 2002 / CAiSE 2002, pages 160–161, Hongkong, China, August 2002.

[RPJA02a] K. Runapongsa, J. M. Patel, H.V. Jagadish, S. Al–Khalifa: The Michigan Benchmark: Towards XML Query Performance Diagnostics.
http://www.eecs.umich.edu/db/mbench/mbench-short.pdf, 2002.

[RV03] E. Rahm, G. Vossen (Hrsg.): Web & Datenbanken – Konzepte, Architekturen, Anwendungen.
Heidelberg: dpunkt.verlag GmbH, 2003.

[SAD+00] S. Staab, J. Angele, S. Decker, M. Erdmann, A. Hotho, A. Mdche, H.-P. Schnurr, R. Studer: Semantic Community Web Portals.
In: 9th WWW Conference, 2000.

[SBB+99] S. Staab, C. Braun, I. Bruder, A. Dserthft, A. Heuer, M. Klettke, G. Neumann, B. Prager, J. Pretzel, H.-P. Schnurr, R. Studer, H. Utzkoreit, B. Wrenger: GETESS - Searching the Web Exploiting German Texts.
In: Proceedings of the 3rd CIA Conf. 1999, pp. 113-124.

[Sch95] U. Schöning: Logik für Informatiker.
Heidelberg: Spektrum Akademischer Verlag GmbH, 1995.

[Sch01] T. Schlieder: ApproXQL: Design and Implementation of an Approximate Pattern Matching Language for XML.
Technical Report B 01–02, FU Berlin, May 2001.

[SemWeb] World Wide Web Consortium: Semantic Web Activity, http://www.w3.org/2001/sw/

[SM00] S. Staab, A. Maedche: Ontology Engineering beyond the Modeling of Concepts and Relations.
In: Proceedings of the 14th ECAI Conf., Workshop on Applications of Ontologies and Problem–Solving Methods, 2000.

[SM02] T. Schlieder, H. Meuss: Querying and Ranking XML Documents.
In: Journal of the American Society for Information Science and Technology (JASIST), 53(6): 489–503, 2002.

[STW01] S. Sizov, A. Theobald, G. Weikum: Ähnlichkeitssuche auf XML–Daten.
In: A. Heuer, F. Leymann, D. Priebe (Eds.): Datenbanksysteme in Büro, Technik und Wissenschaft (BTW) 2001. 9. GI–Fachtagung, Oldenburg, März 2001.
In: Informatik aktuell, pp. 364–383, Berlin: Springer, 2001.

[STW01a] S. Sizov, A. Theobald, G. Weikum: Ähnlichkeitssuche auf XML–Daten.
In: Datenbank–Spektrum

[STW03] R. Schenkel, A. Theobald, G. Weikum: Ontology–enabled XML Search.
In: H. Blanken, T. Grabs, H.-J. Schek, R. Schenkel, G. Weikum (Eds.): Intelligent Search on XML Data.
Pages 119–131, Springer–Verlag: Berlin, 2003.

[STW04] R. Schenkel, A. Theobald, G. Weikum: HOPI: An Efficient Connection Index for Complex XML Document Collections.
In: Proceedings of the 9th International Conference on Extending Database Technology (EDBT) 2004, pages 237–255, Crete, Greece, March 2004.

[SW97] K. Sparck–Jones, P. Willet: Readings in Information Retrieval.
San Francisco: Morgan Kaufmann, 1997.

[SWK+02] A. Schmidt, F. Waas, M. Kersten, M. J. Carey, I. Manolescu, R. Busse: XMark: A Benchmark for XML Data Management.
In: Proceedings of the 28th International Very Large Database Connference (VLDB), pages 974–985, Hongkong, China, August 2002.

[TBMM01] H. S. Thompson, D. Beech, M. Maloney, N. Mendelsohn: XML Schema Part 1: Structures.
http://www.w3.org/TR/xmlschema-1/, W3C Recommendation 2 May 2001.

[The03] A. Theobald: An Ontology for Domain-oriented Semantic Similarity Search on XML Data.
In: (Eds.): Datenbanksysteme für Business, Technologie und Web (BTW) 2003. 10. GI–Fachtagung, Leipzig, Februar 2003.
In: Informatik aktuell, pp. . . . , Berlin: Springer, 2003.

[TW00] A. Theobald, G. Weikum: Adding Relevance to XML.
In: Proceedings of the 3rd International Workshop on the Web and Databases (WebDB) 2000, Dallas, Texas, USA, May 2000.

[TW00a] A. Theobald, G. Weikum: Adding Relevance to XML.
In: D. Suciu, G. Vossen (Eds.): The World Wide Web and Databases. Lecture Notes in Computer Science (LNCS) 1997, pp. 105–124, Berlin: Springer, 2001.

[TW02] A. Theobald, G. Weikum: The Index–based XXL Search Engine for Querying XML Data with Relevance Ranking.
In: Ch.S. Jensen, K.G. Jeffrey, J. Pokorny, S. Saltenis, E. Bertino, K. Böhm, M. Jarke (Eds.): Advances in Database Technology 2002: 8th International Conference on Extending Database Technology (EDBT) 2002, Prague, Czech Republic, March 2002.
Lecture Notes in Computer Science (LNCS) 2287, pp. 477–495, Berlin: Springer, 2002.

[TW02a] A. Theobald, G. Weikum: The XXL Search Engine: Ranked Retrieval of XML Data using Indexes and Ontologies (Demo Paper).
In: Proceedings of the ACM International Conference on Management of Data (SIGMOD) 2002, Madison, Wisconsin, USA, June 2002.

[UC00] The Unicode Consortium: The Unicode Standard, Version 3.0.
Reading, Massachusetts, Addison Wesley Developers Press, 2000.

[WH96] R. Wilhelm, R. Heckmann: Grundlagen der Dokumentverarbeitung.
Bonn: Addison Wesley Longman, 1996.

[WL02] E. Wilde, D. Lowe: XPath, XLink, XPointer, and XML – A Practical Guide to Web Hyperlinking and Transclusion. Boston: Addison Wesley, 2002.

[WinXP] Windows XP Professional, Version 2002, Servicepack 1.
Microsoft Corporation: http://www.microsoft.com/windowsxp/default.asp

[WM97] R. Wilhem, D. Maurer: Übersetzerbau: Theorie, Konstruktion, Generierung.
Berlin: Springer, 2. Auflage, 1997.

[WMB99] I.H. Witten, A. Moffat, T.C. Bell: Managing Gigabytes: Compressing and Indexing Documents and Images.
San Francisco: Morgan Kaufman Publishing, Second Edition, 1999.

[WordNet] WordNet – A Lexical Database for the English Language.
Cognitive Science Laboratory at Princeton University,
http://www.cogsci.princeton.edu/ wn/index.shtml

[YOK02] B. B. Yao, M. T. Özsu, J. Keenleyside: XBench – A Family of Benchmarks for XML DBMSs.
In: Proceedings of the International Workshop on Efficiency and Effectiveness of XML Tools and Techniques (EEXTT) in conjunction with VLDB 2002 / CAiSE 2002, pages 162–164, Hongkong, China, August 2002.

[YOK04] B. B. Yao, M. T. Özsu, N. Khandelwal: XBench Benchmark and Performance Testing of XML DBMSs.
In: Proceedings of the 20th International Conference on Data Engineering (ICDE), Boston, MA, USA, March 2004.

[Zak02] R. Zakon: Hobbes' Internet Timeline v5.6.
http://www.zakon.org/robert/internet/timeline

[ZAR03] P. Zezula, G. Amato, F. Rabitti: Processing XML Queries with Tree Signatures.
In: H. Blanken, T. Grabs, H.-J. Schek, R. Schenkel, G. Weikum (Eds.): Intelligent Search on XML Data.
Pages 247–258, Springer–Verlag: Berlin, 2003.

Index