

Probabilistische Online-Wissensgraphkonstruktion aus natürlicher Sprache

Clemens Damke

10. Oktober 2017
Version: Entwurf 1



PADERBORN UNIVERSITY
The University for the Information Society

Department of Electrical Engineering,
Computer Science and Mathematics
Warburger Straße 100
33098 Paderborn



INTELLIGENT
SYSTEMS

Intelligent Systems Group (ISG)

Bachelorarbeit

Probabilistische Online-Wissensgraphkonstruktion aus natürlicher Sprache

Clemens Damke

1. Korrektor Prof. Dr. Eyke Hüllermeier
 Institut für Informatik
 Universität Paderborn

2. Korrektor Prof. Dr. Axel-Cyrille Ngonga Ngomo
 Institut für Informatik
 Universität Paderborn

Betreuer Dr. Theodor Lettmann

10. Oktober 2017

Clemens Damke

Probabilistische Online-Wissensgraphkonstruktion aus natürlicher Sprache

Bachelorarbeit, 10. Oktober 2017

Korrektoren: Prof. Dr. Eyke Hüllermeier und Prof. Dr. Axel-Cyrille Ngonga Ngomo

Betreuer: und

Universität Paderborn

Intelligente Systeme

Institut für Informatik

Pohlweg 51

33098 Paderborn

Abstract

Hallo Welt. Test5.

Inhaltsverzeichnis

1	Einleitung	3
1.1	Motivation	3
1.2	Ziele der Arbeit	4
1.3	Aufbau der Arbeit	6
2	Verwandte Arbeiten	7
2.1	Ansätze zur Wissensrepräsentation	7
2.1.1	Logische Grundlagen	7
2.1.2	Entwicklung maschineller Wissensrepräsentation	9
2.1.3	Aktuelle Wissensrepräsentationsprojekte	11
2.2	NLP-Werkzeuge	12
2.3	Wissensgraphkonstruktionsverfahren	13
3	Grundlagen und Hilfsmittel	17
3.1	Wissensmodellierung mit Konzeptgraphen	17
3.1.1	Syntax	17
3.1.2	Dominierende Knoten	19
3.2	Stanford CoreNLP	20
3.3	Modellierung von HL-MRFs mit PSL	22
3.3.1	Markov Random Fields	23
3.3.2	Hinge-Loss MRFs	24
3.3.3	Probabilistic Soft Logic	27
3.3.4	Inferenzverfahren	31
4	Wissensgraphkonstruktion aus Kommunikationsdaten	33
4.1	Wissensgraphontologie	34
4.1.1	Verwendete Prädikate	34
4.1.2	Modale Kontexte	36
4.2	NLP-Phase	39
4.2.1	Nachrichtenformat	39
4.2.2	CoreNLP Annotation	40
4.2.3	Konzeptgraphtransformation	42
4.3	Wissensgraphkonstruktionsphase	48
4.3.1	Konstruktion des Konjunktionsgraphen	48

4.3.2	Relationsinferenz mit PSL	49
4.4	Implementation	53
5	Bewertung	55
5.1	Testmethode	56
5.2	Ergebnisse	56
6	Zusammenfassung	57
A	Anhang	61
	Literatur	63

Einleitung

“ *The actual world cannot be distinguished from a world of imagination by any description. Hence the need of pronoun and indices, and the more complicated the subject the greater the need of them.*

— **Charles Sanders Peirce**
Mathematiker und Philosoph

1.1 Motivation

In den letzten Jahren hat die Repräsentation von Wissensbasen durch Graphen, sog. Wissensgraphen, immer mehr an Bedeutung gewonnen. Google, Bing und IBM Watson benutzen solche Wissensgraphen z. B. zum Beantworten von komplexen Suchanfragen.

Die Grundidee dabei ist es, Entitäten durch Knoten und Relationen durch Kanten abzubilden. Entitäten können konkrete Dinge, wie z. B. Personen, aber auch abstrakte Konzepte, wie z. B. historische Epochen, sein. Relationen beschreiben beliebige Beziehungen zwischen den Entitäten, z. B. $person(\text{Da Vinci}) \xrightarrow{\text{lived in}} epoch(\text{Renaissance})$. Die Entität, von der eine solche Relation ausgeht, wird als Subjekt und die Zielentität als Objekt der Relation bezeichnet.

Die Typen von Entitäten bzw. Relationen (z. B. *person* bzw. *lived in*) und deren Bedeutung sind dabei i. d. R. formal in einer sog. Ontologie spezifiziert. Die Ontologie beschränkt also die Menge gültiger Wissensgraphen, was eine effiziente maschinelle Verarbeitung der im Graph enthaltenen Informationen ermöglicht.

Da Wissensgraphen in zahlreichen Domänen einsetzbar sind, wird deren automatisierte Konstruktion bereits seit Jahren erforscht. Manuelles Konstruieren und vor allem anschließendes Warten und Aktualisieren von Wissensgraphen, ist aufgrund der abzubildenden Datenmengen nicht praktikabel. Bei einer maschinellen automatisierten Konstruktion sind insbesondere zwei Anforderungen problematisch:

1. Das Verarbeiten von unstrukturierten Eingaben, wie z. B. natürlichsprachlichen Texten.

2. Effizientes Eingliedern neuer Informationen in einen bestehenden Wissensgraphen. Dieses Eingliedern von Informationen umfasst im Speziellen:

- **Entity Resolution:** Hinzukommende Entitäten, die bereits im Graphen enthalten sind, müssen als Duplikate erkannt werden. Dies ist i. d. R. nicht trivial, da die selbe Entität durch viele verschiedene, oftmals vom Kontext abhängige, Token repräsentiert werden kann; z. B. *Bob* vs. *Robert* oder *Der Papst* vs. *Franziskus*.
- **Link Prediction:** Hinzukommende Entitäten müssen mit bereits bestehenden Entitäten in Relation gesetzt werden. Hinzukommende Relationen können zudem benutzt werden um andere Relationen zu inferieren; z. B.

$$female(A) \wedge B \xrightarrow{\text{son of}} A \implies A \xrightarrow{\text{mother of}} B$$

Die Kombination dieser beiden Anforderungen ist interessant, da das meiste verfügbare Wissen in natürlichsprachlicher Textform vorliegt und zudem permanent neues Wissen entsteht. Ein automatisiertes Wissensgraphkonstruktionsverfahren, welches beide Anforderungen berücksichtigt, ist daher in diversen Domänen von Nutzen. Ein Beispiel hierfür ist die Auswertung von Kommunikationsdaten aus E-Mails oder Chat-Nachrichten mit dem Ziel die sozialen Beziehungen und Intentionen der Kommunikationspartner zu ermitteln.

1.2 Ziele der Arbeit

Das übergeordnete Ziel dieser Arbeit ist es, ein Verfahren zu finden, welches das soeben beschriebene Problem der automatisierten Wissensgraphkonstruktion für Kommunikationsdaten löst. Konkret sei ein Stream von Textnachrichten, wie z. B. E-Mails, gegeben, denen jeweils ein Inhalt, ein Absender, ein Empfänger und ggf. weitere Metadaten, wie z. B. die Absendezeit, zugeordnet ist. Die Nachrichteninhalte werden der Einfachheit halber als ausschließlich englischsprachig angenommen. Außerdem wird eine, für E-Mails und andere Kurznachrichten typische, eingeschränkte Sprachkomplexität angenommen. Die Nachrichten sollen nacheinander in das zu konstruierende System eingefügt werden, welches sukzessive einen Wissensgraphen daraus erzeugt.

Für diese Erzeugung müssen im Wesentlichen drei Teilprobleme gelöst werden:

1. **Onotologie:** Spezifikation einer Wissensgraphontologie, die mächtig genug ist, um möglichst diverse natürlichsprachlich beschriebene Informationen abzubilden.

2. **Sprachverarbeitung:** Finden eines Verfahrens, welches die natürlichsprachlichen Inhalte der Nachrichten in eine für die Wissensgraphkonstruktion geeignete Form bringt.
3. **Grapherweiterung:** Finden eines Verfahrens, um eine eintreffende Nachricht in den bestehenden Wissensgraphen einzufügen.

Die Teillösungen können zu einem Konstruktionsverfahren kombiniert werden, welches grob wie folgt aufgebaut ist:



Abb. 1.1. Grober Aufbau des Konstruktionsverfahrens

Mittels eines Sprachverarbeitungs- und Grapherweiterungsverfahrens wird ein Wissensgraph konstruiert, der die spezifizierte Ontologie benutzt. Der Zweck dieses Wissensgraphen ist, dass er mittels eines Anfragesystems strukturiert ausgelesen werden kann und so zur Beantwortung von Fragen über den Inhalt der eingegebenen Nachrichten benutzt werden kann. In dieser Arbeit wird sich allerdings auf die Konstruktion beschränkt, der Aufbau eines Anfragesystems wird nicht beschrieben.

Das derart zusammengesetzte Verfahren muss folgende technische Anforderungen erfüllen:

1. **Erweiterbarkeit:** Es soll prinzipiell möglich sein, zusätzlich zur Sprachverarbeitung auch andere Verarbeitungsverfahren, z. B. für Bilder, hinzufügen zu können. Insbesondere die Graphontologie darf also nicht zu sehr auf die Struktur natürlicher Sprache zugeschnitten sein, damit auch Informationen aus nicht natürlichsprachlichen Datenquellen repräsentierbar sind.
2. **Parallelisierbarkeit:** Das Verfahren soll in der Lage sein die Rechenleistung mehrerer Prozessorkerne bzw. Prozessoren zu nutzen. Diese Anforderung betrifft insbesondere das Grapherweiterungsverfahren.

Diese Arbeit entstand in Kooperation mit der Firma Atos. Die soeben beschriebenen Ziele und technischen Anforderungen waren dabei vorgegeben. Eine weitere Anforderung war die prototypische Implementation des entwickelten Verfahrens. Ziel ist es dabei jedoch nicht, eine Software zu entwickeln, welche sich in der Praxis einsetzen lässt. Vielmehr geht es darum, prototypisch aufzuzeigen, wie die Wissensgraphkonstruktion aus Kommunikationsdaten prinzipiell ablaufen kann.

1.3 Aufbau der Arbeit

Die drei Komponenten Ontologie, Sprachverarbeitung und Grapherweiterung bilden die Basis des Aufbaus dieser Arbeit. In den Kapiteln 2 bis 4 werden diese Komponenten schrittweise genauer betrachtet, was am Ende in einem konkreten Verfahren zur Wissensgraphkonstruktion resultiert.

Kapitel 2: Verwandte Arbeiten Im ersten Schritt wird ein Überblick über die bisherige Forschung und vorhandenen Technologien gegeben. Das Finden einer Graphontologie lässt sich als ein Problem der Wissensrepräsentation verstehen, daher wird zuerst ein Überblick über die bisherige Entwicklung von Wissensrepräsentationsverfahren gegeben. Anschließend folgt ein Überblick über verbreitete Werkzeuge zur Sprachverarbeitung. Zuletzt wird beschrieben, welche Ansätze es zur Konstruktion von Wissensgraphen mit einer gegebenen Ontologie aus gegebenen Eingabedaten gibt.

Kapitel 3: Grundlagen und Hilfsmittel Da diese Arbeit direkt auf manchen der in Kapitel 2 vorgestellten Arbeiten aufbaut, werden diese Arbeiten hier näher erläutert. Zuerst werden die sog. Konzeptgraphen beschrieben, da die verwendete Graphontologie auf ihnen aufbaut. Es folgt ein Überblick über den Aufbau von CoreNLP, dem verwendeten Werkzeug zur Sprachverarbeitung. Zuletzt wird eine kurze Einführung in HL-MRFs gegeben, einer Klasse von graphischen Modellen, welche zur Konstruktion von Wissensgraphen benutzt werden können.

Kapitel 4: Wissensgraphkonstruktion aus Kommunikationsdaten Basierend auf den in Kapitel 3 vorgestellten Arbeiten wird nun die konkret verwendete Ontologie, das Sprachverarbeitungsverfahren und die Grapherweiterung beschrieben. Außerdem wird kurz erläutert, wie das resultierende Gesamtverfahren implementiert wurde.

Kapitel 5: Bewertung Um die Stärken und Schwächen des in Kapitel 4 beschriebenen Verfahrens aufzuzeigen, wird es hier mit realen Kommunikationsdaten getestet. Da es bislang keine Testdatensets gibt, die gut für die Evaluation des entwickelten Verfahrens geeignet sind, erfolgen keine umfangreichen empirischen Tests. Stattdessen werden stichprobenartig kleine Nachrichtensets eingefügt. Der resultierende Wissensgraph wird anschließend manuell untersucht.

Kapitel 6: Zusammenfassung Abschließend werden die Ergebnisse zusammengefasst und ein Ausblick auf offen gebliebene Fragestellungen gegeben.

Verwandte Arbeiten

Die in 1.2 beschriebenen Ziele werden bereits seit langem erforscht. Der Begriff *Wissensgraph* wurde 2012 durch Google popularisiert, die Ideen dahinter lassen sich allerdings bis ins Ende des 19. Jahrhunderts zurückverfolgen. Dieses Kapitel zeigt auf, wie sich die Themen dieser Arbeit in die bisherige Forschung einfügen. 2.1 ordnet hierfür das Konzept des Wissensgraphen in die Entwicklungsgeschichte der Wissensrepräsentation ein. In 2.2 wird ein Überblick über die momentan verbreiteten *Natural Language Processing* (NLP) Werkzeuge gegeben. Das Wissensgraphkonzept und NLP wird schließlich in 2.3 kombiniert und es werden die aktuell verwendeten Verfahren zur Wissensgraphkonstruktion beschrieben.

2.1 Ansätze zur Wissensrepräsentation

2.1.1 Logische Grundlagen

Die Entwicklung der Wissensrepräsentation hängt eng mit der Entwicklung der Logik zusammen. Während in der formalen Logik und Mathematik die Prädikatenlogik das allgemein verwendete Kalkül ist und alternative Formalismen kaum verbreitet sind, finden im Bereich der Wissensrepräsentation bis heute diverse andere Kalküle Verwendung. Diese werden im Folgenden kurz vorgestellt.

Begriffsschrift (1879) Gottlob Freges Buch über die *Begriffsschrift* [Fre79] gilt als eines der bedeutsamsten Werke der Logik. Sie ist der erste Formalismus mit der Ausdrucksstärke der modernen Prädikatenlogik zweiter Ordnung mit Identität. Frege benutzt hierfür eine zweidimensionale Notation, die sich stark von der heute gebräuchlichen, linearen, an die Algebra angelehnte Notation unterscheidet.

$$\begin{array}{c} \vdash^a \quad \vdash \\ \quad \vdash \quad \vdash \\ \quad \vdash \quad \vdash \\ \quad \vdash \quad \vdash \end{array} \begin{array}{c} \mathfrak{A}(a) \\ \mathfrak{B}(a) \end{array} \Leftrightarrow \exists a : P(a) \vee R(a)$$

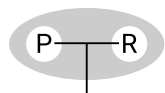
Im Gegensatz zur Prädikatenlogik gibt es keine eigene Syntax für *UND* und *ODER*; diese Operatoren müssen durch die Kombination von Negation und Implikation abgebildet werden. Zudem gibt es ausschließlich den Allquantor; eine existenzquantisierte Aussage muss durch Negation der negierten allquantisierten Aussage ausgedrückt werden.

Prädikatenlogik: Peirce Notation (1885) Unabhängig von Frege entwickelte der amerikanische Mathematiker Charles Sanders Peirce ebenfalls ein prädikatenlogisches Kalkül [Pei85]. Peirces Notation hatte starke Ähnlichkeiten mit der heute benutzten linearen Schreibweise. Statt den modernen Symbolen hat Peirce allerdings die algebraischen Operatoren benutzt, um die Analogien zwischen Logik und Algebra auszudrücken.

$$\Sigma_a P_a + R_a \Leftrightarrow \exists a : P(a) \vee R(a)$$

Existential Graphs (1897) Neben seiner zuvor entwickelten linearen prädikatenlogischen Notation, hat Peirce zudem viele Jahre an einem alternativen graphischen Kalkül gearbeitet, welches er *Existential Graphs* (zu dt. Existenzgraphen) [Sow11] nannte. Ähnlich wie die Begriffsschrift werden Existenzgraphen zweidimensional dargestellt. Von dieser Gemeinsamkeit abgesehen, funktionieren sie allerdings fundamental verschieden. Ein logischer Ausdruck wird hier durch einen ungerichteten Graphen beschrieben. Die konkrete räumliche Anordnung der Knoten und Kanten hat dabei keine semantische Relevanz.

Peirce hat Existenzgraphen als ein dreistufiges aufeinander aufbauendes System konzipiert. Die erste Stufe, die sog. α -Graphen, umfasst alle notwendigen syntaktischen Elemente, um ein Kalkül mit der Ausdruckstärke der Aussagenlogik zu erhalten. Die β -Graphen bilden die zweite Stufe und erweitern die Syntax der α -Graphen, sodass die Ausdruckstärke der Prädikatenlogik erster Ordnung erreicht wird. Sowohl für α -, als auch für β -Graphen, ist die Vollständigkeit und Korrektheit bewiesen. Die dritte Stufe (γ -Graphen) wurde von Pierce nie vollendet; sie deckt in etwa die Ausdruckstärke der heutigen Prädikatenlogik höherer Ordnung sowie der Modallogik ab.



$$\Leftrightarrow \exists a : P(a) \vee R(a)$$

Wie schon die Begriffsschrift, sind Existenzgraphen syntaktisch minimal. Direkt ausdrücken lässt sich lediglich *UND*, der Existenzquantor und die Negation. Ein weiterer Unterschied zur heutigen Prädikatenlogik ist die Beschreibung logischer Inferenzen. Im Gegensatz zu den prädikatenlogischen Ersetzungsaxiomen, die auf der syntaktischen Struktur von logischen Ausdrücken operieren (z. B. für Kommutativität), lassen sich die Ersetzungsaxiome für Existenzgraphen als Graphtransformationsregeln verstehen, die bestimmte Teilmengen der Knoten und Kanten eines Ausdrucks durch andere äquivalente Knoten- und Kantenmengen ersetzen.

Prädikatenlogik: Peano-Russell Notation (1910) Die zweidimensionalen Notationen wurde häufig kritisiert, da sie die lineare, algebraische Notation der symbolischen Logik von Boole und De Morgan verwarfen [Slu87]. Freges Begriffsschrift und Peirces Existenzgraphen konnten sich daher nicht durchsetzen. Peirces algebraische prä-

dikatenlogische Notation hingegen, stieß auf größere Akzeptanz. Giuseppe Peano hat auf deren Basis eine ähnliche Notation entwickelt, welche allerdings nicht die algebraischen Operatoren benutzt, damit sich logische Ausdrücke besser mit mathematischen Ausdrücken kombinieren lassen. Bertrand Russell hat Peanos Notation anschließend in leicht abgewandelter Form in den *Principia Mathematica* [WR10] benutzt. Diese sog. Peano-Russell-Notation ist im Wesentlichen identisch mit der modernen Schreibweise.

Trotz des Verschwindens der zweidimensionalen Notationen, finden sich noch heute Anlehnungen daran [Wik]. So ist z. B. die Negation $\neg A$ auf Freges negierten Inhaltsstrich $\neg\text{---} A$ und der Ableitungsoperator \vdash auf Freges Urteilsstrich mit angefügtem Inhaltsstrich $\vdash\text{---}$ zurückzuführen.

2.1.2 Entwicklung maschineller Wissensrepräsentation

Die Idee Computer zur Lösung beliebiger Probleme zu benutzen ist nicht neu. Da ein solches maschinelles Problemlösen die Verfügbarkeit von Hintergrundwissen über die Problemdomäne erfordert, wurden Methoden zur Wissensrepräsentation immer im Zusammenhang mit Problemlösern erforscht [HR+83]. So wie effiziente Datenstrukturen die Implementation effizienter Algorithmen ermöglichen, ermöglichen gute Wissensrepräsentationen die Implementation guter Problemlöser. Was genau nun als ein guter Problemlöser verstanden wird, hat sich im Laufe der Jahre allerdings immer wieder verändert.

Universelle Problemlöser Einer der ersten maschinellen Problemlöser war der von Newell und Simon 1955 entwickelte *Logic Theorist* (LT) [NS56]. LT war in der Lage logische Aussagen zu beweisen, indem er systematisch Ersetzungsaxiome auf eine gegebene Aussage angewandt hat, bis die gesuchte Lösung abgeleitet wurde.

Die Grundidee des LT haben Newell, Simon und Shaw 1959 im *General Problem Solver* (GPS) [New+59] erweitert. Es wurden Heuristiken hinzugefügt, um den Suchraum geschickter zu durchlaufen. GPS war ein universeller Problemlöser, konnte also jedes Problem lösen, das sich durch eine Menge von Horn-Klauseln ausdrücken lässt. Zwar war es so theoretisch möglich Probleme aus diversen Domänen zu lösen, aufgrund der kombinatorischen Explosion war GPS allerdings nicht zur Lösung komplexer praktischer Probleme geeignet.

Expertensysteme Aufgrund der Misserfolge universeller Problemlöser für praktische Probleme, hat die Forschung begonnen sich mehr auf die Entwicklung von Expertensystemen zu fokussieren. Expertensysteme besitzen für gewöhnlich eine Wissensbasis in der domänenspezifisches Wissen in Form von Regeln und Fakten kodiert ist. Eine sog. Inferenzmaschine benutzt diese Regeln und Fakten um Probleme zu lösen.

Semantic Networks (1956) Die Idee, Graphen als Datenstruktur für Wissensbasen zu verwenden, taucht erstmal in den sog. *Semantic Networks* [Leh92] (zu dt. semantische Netzwerke) auf. Dieser Ansatz beschreibt Wissen als Menge von *(subject, predicate, object)*-Tripeln. Es gibt darüber hinaus allerdings keine klaren Regeln, wie ein semantisches Netz strukturiert sein muss. Semantische Netzwerke sind daher primär als ein Oberbegriff für die große Vielfalt konkreter graphbasierter Wissensbasen zu verstehen. Dennoch lässt sich eine Gemeinsamkeit zwischen den diversen Ansätze festmachen: Meist kommt die sog. *IS-A*-Relation vor [Bra83], welche das Konzept der Vererbung repräsentiert. Die Beschreibung von baumartigen Taxonomien mittels *IS-A* ist ein häufiges Einsatzgebiet semantischer Netzwerke.

Conceptual Graphs (1976) Wie genau mit Graphen komplexes Wissen beschrieben werden kann, das über eine reine Taxonomie hinaus geht, blieb bei semantischen Netzen unklar. John F. Sowa's *Conceptual Graphs* [Sow76][Har+07] (zu dt. Konzeptgraphen) lösen dieses Problem. Statt Wissen lediglich als eine einfache Menge von Beziehungen abzubilden, wird es als prädikatenlogischer Ausdruck verstanden. Hierfür baut Sowa auf Peirces Existenzgraphen auf, die bis dahin weitestgehend unbeachtet waren.



Dieser Ansatz erlaubt es komplexe Wissensbasen zu konstruieren, in denen nicht nur gespeichert werden kann, ob ein Konzept existiert, sondern auch, ob es nicht oder nur möglicherweise existiert. Da Konzeptgraphen, so wie schon die Existenzgraphen, ein vollständiges und korrektes Logikkalkül sind, lassen sich zudem Inferenzregeln für sie definieren. Der Vorteil hierfür einen Graphen statt eines prädikatenlogischen Ausdrucks zu verwenden ist, dass eine Graphstruktur einen deutlich effizienteren Zugriff auf gespeichertes Wissen ermöglicht.

Knowledge Graphs (1987) Der Begriff *Knowledge Graph* [RM92] (zu dt. Wissensgraph) bezeichnete ursprünglich eine Klasse semantischer Netze, deren Relationsmenge formal spezifiziert ist. Die Menge erlaubter Graphen wird hierdurch so eingeschränkt, dass das repräsentierte Wissen eindeutig interpretierbar und ohne Redundanz ist. Dies erlaubt die Definition von Inferenzregeln, um Schlussfolgerungen aus einem gegebenen Graphen zu ziehen. Im Laufe der Jahre ist die Grenze zwischen semantischen Netzen und Wissensgraphen allerdings so stark verschwommen, dass heute auch mehrdeutige, redundanzbehaftete semantische Netze, die lediglich wenige Relationstypen benutzen, als Wissensgraphen bezeichnet werden [McC+16]. Wissensgraphen und Konzeptgraphen müssen weiterhin streng unterschieden werden, da erstere oftmals Negation und Modalität nicht unterstützen.

2.1.3 Aktuelle Wissensrepräsentationsprojekte

Manuelle Ansätze

Semantic Web Das sog. *Semantic Web* bezeichnet eine Menge von W3C-Standards, die das bestehende Web um eine formale Wissensbeschreibungssyntax erweitern [BH]. Zentral ist dabei das *Resource Description Framework* (RDF), mit dem sich beliebige Konzepte, auch Ressourcen genannt, beschreiben und verknüpfen lassen. Ziel ist es über die unstrukturierte Netzstruktur des bestehenden Webs, eine strukturierte, leicht maschiell verarbeitbare, Netzstruktur zu legen. Durch die Anfragesprache *SPARQL* ist es möglich Wissen aus diesem Netz auszulesen. Das Web würde somit zu einem großen dezentralen Wissensgraphen. Tim Berners-Lee beschreibt diese Idee als das “Web 3.0” [Sha06]. Obwohl die Technologien hierfür bereits seit Jahren existieren, sind bislang nur wenige Webseiten mit RDF-Tags annotiert. Häufige Kritik ist, dass das Semantic Web zu viel theoretisches Hintergrundwissen über Wissensrepräsentationsverfahren erfordert, um für die meisten Webseitenbetreiber zugänglich zu sein [MS03].

WordNet Das *WordNet* der Universität Princeton [Wor] ist ein frei verfügbares lexikalisch-semantisches Netz für die englische Sprache, d. h. ein semantisches Netz, welches die Bedeutung von Worten in Relation zueinander setzt. Relationen werden dabei z. B. für Synonyme, Hyperonyme (Oberbegriffe) und Meronyme (Bestandteile) eingefügt. Der Datenbestand des WordNets wird manuell gepflegt und resultiert aus der Kombination der Einträge verschiedener Wörterbücher.

Automatisierte Ansätze

Neben den manuellen Grapherzeugungsansätzen des Semantic Webs und des WordNets, gibt es diverse voll- und semiautomatische Ansätze. Diese bauen die Graphstruktur selbstständig aus gegebenen Datenquellen auf.

NELL Das *Never-Ending Language Learning* (NELL) System [Car+10] traversiert selbstständig das Internet und fügt die gefundenen textuellen Informationen in einen Wissensgraphen ein. Hierfür wird eine Kombination verschiedener Modelle verwendet, die regelmäßig angepasst wird. Menschen können optional Feedback für die extrahierten Fakten geben, um die Inferenzqualität weiter zu verbessern.

Google Knowledge Graph Basierend auf den Ideen der in 2.1.2 vorgestellten Wissensgraphen, stellte Google 2012 eine eigene, ebenfalls *Knowledge Graph* genannte, Wissensgraphentechnologie vor [Sin12]. Sie wird benutzt, um Suchanfragen semantisch, statt per String-Matching, zu beantworten. So können z. B. zum Suchbegriff

verwandte Ergebnisse angezeigt werden, selbst wenn es keine textuelle Ähnlichkeit zu jenem gibt. Laut Googles Aussagen stammen die Quelldaten u. a. aus Wikipedia Infoboxen, Wikidata und dem CIA World Factbook. Da es sich hierbei, im Gegensatz zu NELL, primär um strukturierte Daten handelt, ist das automatisierte Einpflegen mit hoher Genauigkeit möglich. Wie genau die Daten im Graph repräsentiert werden, ist nicht öffentlich bekannt.

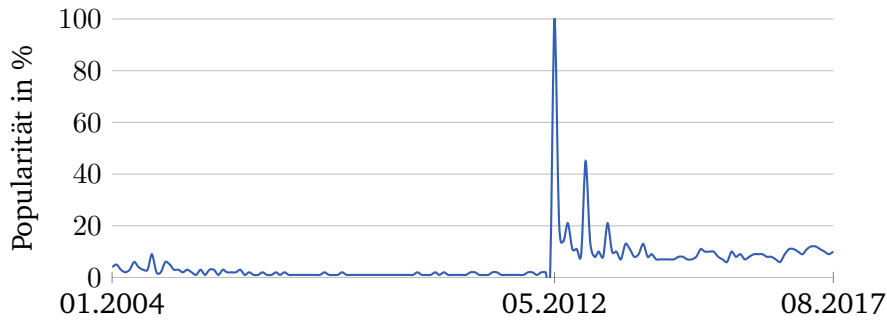


Abb. 2.1. Popularität des Begriffs “knowledge graph” (Quelle: Google Trends [Goo])

2.2 NLP-Werkzeuge

Neben der Repräsentation von Wissen, ist auch die Verarbeitung natürlicher Sprache ein Kernbestandteil dieser Arbeit. Hierfür existiert bereits eine Vielzahl von *Natural Language Processing* (NLP) Werkzeugen. Trotz dieser Vielfalt lassen sich Kernverarbeitungsschritte festmachen, die in den meisten Werkzeugen verwendet werden.

1. **Tokenization:** Oftmals einer der ersten Verarbeitungsschritte einer NLP Bibliothek. Eine Eingabezeichenkette wird dabei in eine Liste von Tokens zerlegt. Token sind u. a. Wörter, Satzzeichen und numerische Literale.

“Today I’m testing myself.” → (Today, I, ’m, testing, myself, .)

2. **Lemmatization:** Abbildung von Tokens auf ihre Lemmata (Grundformen).

(Today, I, ’m, testing, myself, .) → (today, I, be, test, myself, .)

3. **Part-of-speech Tagging:** Abbildung von Tokens auf ihre Wortarten und Flexionen (POS-Tags).

Today	I	’m	testing	myself	.
adverb	personal pronoun	present tense first-person singular verb	present participle verb	personal pronoun	sentence terminator

4. **Named Entity Recognition:** Klassifizierung von Tokens in Kategorien wie z. B. Person, Ort oder Zeitpunkt.

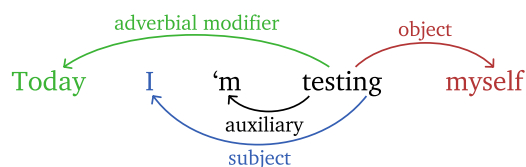
(Today, I, 'm, testing, myself, .)

date

5. **Coreference Resolution:** Bestimmung von Token-Äquivalenzklassen, die jeweils auf das selbe Konzept verweisen (insbesondere Pronomina und ihr Antezedens).

(Today, I, 'm, testing, myself, .)

6. **Dependency Parsing:** Eine auf den POS-Tags aufbauende syntaktische Analyse, welche die grammatikalischen Abhängigkeiten der Token untereinander ausgibt. Die Menge dieser Abhängigkeiten bildet einen Baum oder baumähnlichen Graphen, der *Treebank* bzw. *Dependency Graph* genannt wird.



Wie sich erkennen lässt, bauen die Verarbeitungsschritte sukzessive aufeinander auf und bilden eine Art Pipeline [Usz]. Dieses Pipeline-Modell findet sich in vielen NLP-Werkzeugen wieder. Ein solches ist z. B. das quelloffene Stanford CoreNLP Projekt [Man+14][Cor], welches u. a. Module für alle der soeben vorgestellten Verarbeitungsschritte beinhaltet. Ein alternatives NLP-Toolkit ist Apache OpenNLP [Ope]; es bietet ähnliche Module wie CoreNLP an.

2.3 Wissensgraphkonstruktionsverfahren

Wie in 2.1.3 gezeigt, gibt es diverse Ansätze um Graphen aus Daten zu konstruieren. Da für das Thema dieser Arbeit insbesondere automatisierte Verfahren relevant sind, die mit unstrukturierten Daten, wie z. B. natürlicher Sprache, umgehen können, werden diese im Folgenden näher beschrieben.

Üblicherweise arbeiten Wissensgraphkonstruktionsverfahren nicht direkt mit den unstrukturierten Eingabedaten, wie z. B. den Inhalten von E-Mails, sondern mit einer Knoten- bzw. Konzeptmenge und ggf. auch einer Kanten- bzw. Relationsmenge, die zuvor, z. B. mittels eines in 2.2 vorgestellten NLP-Verfahrens, aus den Rohdaten extrahiert wurden. Die Wissensgraphkonstruktion ist somit äquivalent zum Problem der *Link Prediction*, also dem Finden von Relationen zwischen den gegebenen Konzepten.

Die Link Prediction wiederum lässt sich als ein Problem des *Statistical Relational Learnings* (SRL) auffassen. In der Literatur finden sich im Wesentlichen drei Klassen von SRL-Verfahren [Nic+16], die auf verschiedenen Annahmen über die Korrelation der zu verknüpfenden Informationen basieren:

1. **Latent Feature Models:** Alle Relationen werden als bedingt unabhängig angenommen, sofern bestimmte Eigenschaften über Subjekte und Objekte der Relationen gegeben sind.
2. **Graph Feature Models:** Alle Relationen werden als bedingt unabhängig angenommen, sofern bestimmte Eigenschaften der Struktur des Graphen gegeben sind.
3. **Markov Random Fields:** Es wird angenommen und erlaubt, dass alle Relationen lokale Abhängigkeiten voneinander haben können.

Latent Feature Models Ein Beispiel für ein Latent Feature Modell ist das RESCAL-Verfahren [Nic13], welches auf Tensorfaktorisierung basiert. Die Grundidee dabei ist es, allen Entitäten i einen Feature-Vektor $e_i \in \mathbb{R}^H$ zuzuordnen und für alle Relationen k eine Gewichtsmatrix $W_k \in \mathbb{R}^{H \times H}$ zu finden. Die Konfidenz in die Existenz einer Relation $i \xrightarrow{k} j$ wird durch $e_i^\top W_k e_j$ beschrieben. Diese Definition ermöglicht eine sehr schnelle Link Prediction, da lediglich ein Vektor-Matrix-Vektor-Produkt berechnet werden muss. RESCAL liefert gute Ergebnisse, wenn die vorherzusagenden Relationen globale Abhängigkeiten aufweisen. Lokal stark zusammenhängende Teilgraphen werden allerdings schlecht erkannt, da nur der Feature-Vektor und nicht die Nachbarschaft einer Entität berücksichtigt wird; ein Beispiel hierfür sind symmetrische Relationen.

$$A \xrightarrow{\text{married to}} B \implies B \xrightarrow{\text{married to}} A$$

Graph Feature Models Komplementär zu den Latent Feature Modellen sind die Graph Feature Modelle. Statt Entitäten in einen Feature-Raum einzubetten, wird hier die Nachbarschaft der Entitäten betrachtet. Ein Beispiel hierfür ist der *Path Ranking Algorithmus* (PRA) [Lao+11]. PRA ermittelt Relationen durch zufälliges Durchwandern des Graphen. Um die Stärken der Latent Feature und Graph Feature Modelle zu kombinieren, wurden Hybrid-Modelle, wie z. B. das *Additive Relational Effects* (ARE) Verfahren [Nic+14], entwickelt, welches die Konfidenzen von RESCAL und PRA addiert.

Markov Random Fields Fundamental verschieden von diesen beiden Verfahren sind *Markov Random Fields* (MRFs). Hier sind prinzipiell Abhängigkeiten zwischen allen Relationen möglich, was MRFs sehr flexibel macht. Da dies hinsichtlich der Laufzeit schnell impraktikabel wird, wird das Modell um einen Abhängigkeitsgraphen erweitert, der die Anzahl von betrachteten Abhängigkeiten reduziert. Der Abhängig-

keitsgraph darf dabei nicht mit dem Wissensgraphen verwechselt werden: Ersterer beschreibt statistische Abhängigkeiten zwischen Relationen, während letzterer Relationen zwischen Konzepten beschreibt. Zur Modellierung von Abhängigkeitsgraphen werden i. d. R. Kalküle verwendet, die an eine Prädikatenlogik erster Ordnung angelehnt sind. Das Finden eines Wissensgraphen ist in diesem Modell analog zum Lösen des MAX-SAT-Problems. Wählt man ein Kalkül in dem die Atome (i. e. Zufallsvariablen des MRFs) aus $[0, 1]$ sind, und Formeln zudem ausschließlich Disjunktionen und Negationen gemäß Łukasiewicz T-Norm enthalten, erhält man ein sog. *Hinge-Loss-MRF* [Bac+13][Bac+15].

Ein konkretes Kalkül, welches sich zur Spezifikation von HL-MRFs eignet, ist die *Probabilistic Soft Logic* (PSL) [Br10][Bac+15]. MAX-SAT lässt sich für solche HL-MRFs effizient und parallelisierbar mit dem konvexen *Alternating Direction Method of Multipliers* (ADMM) Optimierungsverfahren [Boy+11] lösen. In dessen ursprünglichen Form ist ADMM allerdings ausschließlich für offline Inferenz geeignet; der Wissensgraph müsste also bei jeder Eingabe neu konstruiert werden. Um dieses Problem zu lösen, wurde das *Budgeted Online Collective Inference* (BOCI) Verfahren [Puj+15] entwickelt. BOCI nutzt Metadaten, die während der Ausführung von ADMM anfallen, um eine Bewertung für jedes Atom zu berechnen. Die Bewertung eines Atoms beschreibt, wie groß die erwartete Wertveränderung beim Eintreffen neuer Informationen ist. Kommen nun neue Informationen an, müssen ausschließlich die m höchstbewerteten Atome betrachtet werden, die Werte aller anderen Atome werden fixiert. Je höher das Budget m , desto höher ist die Qualität im Vergleich zu einer Neukonstruktion des Graphen. Es wurde empirisch gezeigt, dass die Inferenzqualität mit BOCI oft nur unwesentlich schlechter ist, als bei einer kompletten offline Inferenz.

Die Kombination von PSL, ADMM und BOCI ist daher ein guter Ausgangspunkt für den Entwurf eines online Wissensgraphkonstruktionsverfahrens. Der Vorteil dieses Ansatzes gegenüber eines Latent Feature oder Graph Feature Modells ist, dass sich andere domänenspezifische Expertensysteme leicht in eine PSL Inferenz integrieren lassen. PSL erlaubt nämlich die Inklusion von benutzerdefinierten Funktionen und Prädikaten. Diese können benutzt werden, um z. B. die Levenshtein-Distanz zweier Zeichenketten oder domänenspezifisches Hintergrundwissen, wie die Distanz zwischen zwei namentlich genannten Orten, mit in die Entity Resolution einfließen zu lassen.

Grundlagen und Hilfsmittel

In Kapitel 2 wurde ein Überblick über das Problemumfeld der Wissensgraphkonstruktion gegeben. Diese Arbeit baut insbesondere auf den bereits kurz vorgestellten Konzeptgraphen, Stanfords CoreNLP Bibliothek und der PSL auf. Für die folgenden Kapitel ist ein Grundverständnis dieser drei Themen notwendig. Sie werden daher in den folgenden Abschnitten näher beschrieben.

3.1 Wissensmodellierung mit Konzeptgraphen

John F. Sowa's Konzeptgraphen bilden die Basis der Graphontologie dieser Arbeit. Wie in 2.1.2 beschrieben, sind sie ein auf Existenzgraphen basierendes logisches Kalkül. Die vollständige Konzeptgraphsyntax geht allerdings weit über die Prädikatenlogik hinaus, da auch Modallogik und natürlichsprachliche Konzepte, wie z. B. Fragen und Betonungen, unterstützt werden. Da Sowa's eigene Beschreibungen diesbezüglich teils etwas unklar sind, werden im folgenden lediglich die sog. *Conceptual Graphs with Cuts* [Dau03] vorgestellt. Sie sind eine zur Prädikatenlogik erster Ordnung äquivalente, formal spezifizierte Teilmenge der Konzeptgraphen, deren Vollständigkeit und Korrektheit bewiesen ist.

3.1.1 Syntax

In ihrer einfachsten Form lassen sich Konzeptgraphen als Graphen mit drei Arten von Knoten und zwei Arten von Kanten beschreiben.

Konzeptknoten (*concepts*) Entsprechen in etwa existenzquantisierten gebundenen Variablen. Wie auch in der Prädikatenlogik, haben die Bezeichner von Konzeptknoten keine semantische Relevanz und können frei gewählt werden.

$$\boxed{a} \quad \boxed{b} \quad \Leftrightarrow \quad \exists a, b \quad (3.1)$$

Relationsknoten (*conceptual relations*) und Argumentkanten (*arguments*) Relationsknoten entsprechen prädikatenlogischen Atomen. Das Symbol innerhalb eines Relationsknoten gibt die Relation des Atoms an. Für die Repräsentation der Argumente werden sog. Argumentkanten zwischen Relationsknoten und Konzeptknoten verwendet. Die Position der Argumente bei mehrstelligen Relationen werden durch Nummerierung der Argumentkanten oder bei zweistelligen Relationen durch gerichtete Argument-

kanten abgebildet. Wenn in einem Graphen mehrere Relationsknoten bzw. Atome auftauchen, werden diese als *UND*-verknüpft interpretiert; für die Abbildung von *ODER* wird die Negation verwendet.

$$\begin{array}{c} \text{a} \end{array} \xrightarrow{\text{P}} \begin{array}{c} \text{b} \end{array} \quad \Leftrightarrow \quad \exists a, b : P(a, b) \wedge R(b, a) \quad (3.2)$$

Negationskontexte (negation contexts oder cuts) Für die Negation von Aussagen werden in Konzeptgraphen sog. Kontexte verwendet. Sie lassen sich neben der Negation auch zur Modellierung anderer Zusammenhänge nutzen, diese werden hier allerdings ausgelassen, um den Vergleich mit der Prädikatenlogik zu ermöglichen.

$$\begin{array}{c} \text{a} \end{array} \xrightarrow{\text{P}} \begin{array}{c} \text{b} \end{array} \quad \Leftrightarrow \quad \exists a \neg \exists b : P(a, b) \quad (3.3)$$

$$\Leftrightarrow \quad \exists a \forall b : \neg P(a, b)$$

Die Darstellung von Kontexten mit Knoten und Kanten wird schnell unübersichtlich, daher werden stattdessen Boxen verwendet, die die Kindknoten umschließen.

$$\begin{array}{c} \text{a} \end{array} \xrightarrow{\text{P}} \begin{array}{c} \text{b} \end{array} \quad \Leftrightarrow \quad \boxed{\begin{array}{c} \text{a} \end{array} \xrightarrow{\text{P}} \begin{array}{c} \text{b} \end{array}}$$

Kontexte können nicht nur Konzeptknoten und Relationsknoten enthalten, sondern auch andere Kontexte. Hierbei ist zu beachten, dass alle Knoten und Kontexte höchstens einen Elternkontext haben können; die Linien zweier Kontextboxen dürfen sich also nicht schneiden.

$$\boxed{\begin{array}{c} \text{a} \end{array} \xrightarrow{\text{P}} \begin{array}{c} \text{b} \end{array}} \quad \Leftrightarrow \quad \neg \exists a \neg \exists b : P(a, b) \quad (3.4)$$

$$\Leftrightarrow \quad \forall a \exists b : P(a, b)$$

$$\begin{array}{c} \text{a} \end{array} \xrightarrow{\text{P}} \begin{array}{c} \text{b} \end{array} \quad \Leftrightarrow \quad \exists a, b : \neg(\neg P(a, b) \wedge \neg R(b, a)) \quad (3.5)$$

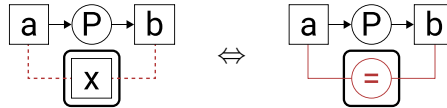
$$\Leftrightarrow \quad \exists a, b : P(a, b) \vee R(b, a)$$

Koreferenzkanten (coreference links) Entspricht der Äquivalenzrelation.

$$\begin{array}{c} \text{a} \end{array} \xrightarrow{\text{P}} \begin{array}{c} \text{b} \end{array} \quad \Leftrightarrow \quad \exists a, b : P(a, b) \wedge \neg \exists x : a = x \wedge x = b \quad (3.6)$$

$$\Leftrightarrow \exists a, b : P(a, b) \wedge a \neq b$$

Prinzipiell ließe sich die Äquivalenz auch durch Relationsknoten ausdrücken. Um syntaktisch zu kennzeichnen, dass es sich nicht um eine beliebige Relation, sondern um eine Äquivalenzrelation handelt, wird dies jedoch i. d. R. nicht getan. Koreferenzkanten können also als eine Kurzschreibweise verstanden werden, die den Zweck hat die für die Inferenz relevanten Symmetrie-, Transitivitäts- und Reflexivitätseigenschaften zu kennzeichnen.



3.1.2 Dominierende Knoten

So wie die Syntaxelemente prädikatenlogischer Ausdrücke nicht beliebig kombiniert werden können, unterliegen auch Konzeptgraphen gewissen Einschränkungen. Die Einschränkung, dass alle Knoten und Kontexte höchstens einen Elternkontext haben dürfen, wurde bereits erwähnt. Die zweite wichtige Einschränkung ist das Verbot nicht dominierender Knoten (*dominating nodes*). Was genau dies bedeutet, wird im Folgenden erläutert. Zuerst müssen Konzeptgraphen jedoch formal spezifiziert werden.

$$\begin{aligned}
 G &:= (V, E), \text{ mit globalem Kontext } \top \in V \\
 \text{concept}(v) &:\Leftrightarrow v \in V \text{ ist ein Konzeptknoten} \\
 \text{relation}(v) &:\Leftrightarrow v \in V \text{ ist ein Relationsknoten} \\
 \text{context}(v) &:\Leftrightarrow v \in V \text{ ist ein Kontext, es gilt } \text{context}(\top) \\
 \text{neg}(v) &:\Leftrightarrow v \in V \text{ ist ein Negationskontext} \\
 \text{nest}(c, v) &:\Leftrightarrow \text{context}(c) \wedge (c, v) \in E \\
 \text{coref}(v_1, v_2) &:\Leftrightarrow \text{concept}(v_1) \wedge \text{concept}(v_2) \wedge (v_1, v_2) \in E \\
 \text{arg}(r, v) &:\Leftrightarrow \text{relation}(r) \wedge \text{concept}(v) \wedge ((r, v) \in E \vee (v, r) \in E) \\
 E &\supseteq \{(\top, v) : v \in V \wedge \neg \exists c \in V \setminus \{\top\} : \text{nest}(c, v)\} \\
 a \leq b &:\Leftrightarrow (\exists x \in V : a \leq x \wedge x \leq b) \\
 &\quad \vee \text{nest}(b, a) \vee (\exists c \in V : \text{nest}(c, a) \wedge \text{nest}(c, b))
 \end{aligned} \tag{3.7}$$

Um die nachfolgenden Definitionen einfacher zu machen, wird der globale Kontext \top eingeführt, der, in Anlehnung an Peirce, *Sheet of Assertion* genannt wird. \top enthält alle Knoten, die keinen explizit dargestellten Elternkontext haben. Die Kontextbox von \top umschließt also den gesamten Konzeptgraphen. \leq ist eine Quasiordnung und bildet die *enthalten-in*-Relation zwischen Knoten ab, d. h. $a \leq b$ gdw. a innerhalb der Box des Elternkontextes von b liegt. Das größte Element gemäß \leq ist also immer \top .

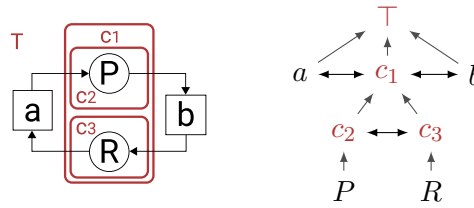


Abb. 3.1. Zusammenhang zwischen Kontexten und der \leq -Ordnung. Die Existenz eines Pfades von x nach y im obigen baumartigen Graphen, entspricht $x \leq y$.

Auf Basis von \leq lässt sich nun das Konzept dominierender Knoten definieren.

$$\begin{aligned} \text{dom}(G) &:= \forall r, v \in V : \text{arg}(r, v) \rightarrow r \leq v \\ &\wedge \forall v_1, v_2 \in V : \text{coref}(v_1, v_2) \rightarrow (v_1 \leq v_2 \vee v_2 \leq v_1) \end{aligned} \quad (3.8)$$

Für jeden Konzeptgraphen G muss $\text{dom}(G)$ gelten. Eine Intuition für diese Einschränkung ist, dass es nicht sinnvoll ist die Existenz eines Atoms auszudrücken, welches durch nicht existente Variablen parametrisiert ist. Eine detaillierte Untersuchung des Zwecks dominierender Knoten und eine Beschreibung der entstehenden Probleme, wenn auf die Notwendigkeit dominierender Knoten verzichtet wird, findet sich in [Dau03, Abschnitt 14.3].

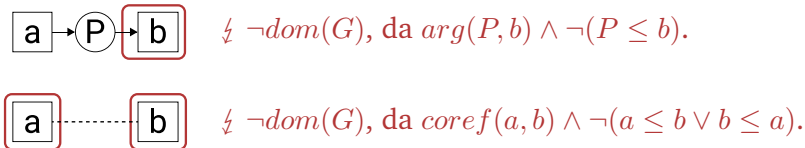


Abb. 3.2. Beispiele für fehlerhafte Konzeptgraphen ohne dominierende Knoten.

3.2 Stanford CoreNLP

Um natürlichsprachliche Daten in einen Konzeptgraphen zu transformieren, ist im ersten Schritt eine Sprachanalyse notwendig. Hierfür wurde die Stanford CoreNLP [Cor] und die Apache OpenNLP [Ope] Bibliothek in Erwägung gezogen, da beide häufig genutzt, aktiv weiterentwickelt, frei verfügbar und JVM-basiert sind. Die JVM-Integration ist wichtig, um mit anderen verwendeten Bibliotheken kompatibel zu sein; mehr hierzu in Abschnitt 4.4. Für die Implementation wurde schließlich CoreNLP gewählt, da es mit den mitgelieferten Modellen häufig bessere Ergebnisse als OpenNLP liefert. Da beide Bibliotheken bzgl. ihrer Funktionalität allerdings recht

ähnlich sind, kann die NLP Komponente als substituierbar angesehen werden. Ein Wechsel von CoreNLP auf OpenNLP wäre mit relativ geringem Aufwand möglich.

Im Folgenden wird nun die grundlegende Architektur von CoreNLP beschrieben. CoreNLP verwendet das in 2.2 vorgestellte Pipeline-Modell. Die verschiedenen Verarbeitungsstufen der Pipeline werden Annotatoren genannt. Da die genaue Funktionsweise der Annotatoren ist für diese Arbeit weniger relevant, wichtiger ist ein Überblick über die Art der Ergebnisse, die die Annotatoren liefern.

Tokenization und Lemmatization Liefern, wie erwartet, eine Liste von Token bzw. eine Liste der Lemmata der Token. Es werden neben Englisch zahlreiche andere Sprachen und ein Großteil des Unicode Zeichensatzes unterstützt. Der Tokenizer verwendet zum Finden der Tokens intern einen deterministischen endlichen Automaten.

POS-Tagging Ordnet jedem Token eine Wortart und Flexion (POS-Tag) zu. Die Menge der möglichen POS-Tags wurde aus dem *Penn Treebank Tag Set* [San90] übernommen. Für das Finden der Tags benutzt CoreNLP sog. *Cyclic Dependency Networks* [Tou+03], eine Erweiterung bayesscher Netze, in denen zyklische Abhängigkeiten erlaubt sind.

Named Entity Recognition (NER) Findet sog. Entitäten. CoreNLP benutzt hierfür eine Menge von Entitätsklassen, die sich in drei Kategorien von Klassen unterteilen lässt:

1. **Benannte Entitäten:** Person, Ort, Organisation und Sonstige. Diese Entitätsklassen werden mittels *Conditional Random Fields* [Fin+05], einer Variante von *Markov Random Fields* (siehe 3.3.1), erkannt.
2. **Numerische Entitäten:** Geldbetrag, Zahl, Ordinalzahl und Prozentzahl. Hierfür wird ein regelbasiertes System verwendet. Die so erkannten Token werden zudem normalisiert, um eine leichtere Weiterverarbeitung zu ermöglichen.
3. **Temporale Entitäten:** Datum, Uhrzeit, Dauer und Menge von Zeitpunkten. Diese Klassen werden ebenfalls mit einem regelbasierten System erkannt. Mittels SUTime [CM12] werden die erkannten Token anschließend normalisiert und relative Zeitangaben in absolute Zeitpunkte aufgelöst, sofern ein Referenzzeitpunkt gegeben ist. Für die Normalisierung wird das TimeML TIMEX3-Format [Tim] benutzt, mit dem sich auch komplexe Zeitangaben, wie "twice a week" (type="set" value="P1W" freq="2X"), formal ausdrücken lassen.

Coreference Resolution Ermittelt Äquivalenzklassen von Token, die auf das selbe Konzept bzw. die selbe Entität verweisen. CoreNLP stellt hierfür drei verschiedene Systeme bereit: Ein schnelles, regelbasiertes, deterministisches System, ein etwas

langsames statistisches System und zuletzt ein langsames System, das auf neuronalen Netzen basiert.

Dependency Parsing Dieser Annotator ermittelt die grammatikalischen Beziehungen zwischen Worten. Das Ergebnis ist ein sog. Abhängigkeitsgraph (*Dependency Graph*), in dem die Knoten Token und die Kanten Beziehungen repräsentieren. CoreNLP verwendet für die Kantentypen *Universal Dependencies Version 2* (UD v2) [Udv], eine Menge von 37 Arten grammatikalischer Beziehungen, die für eine Vielzahl natürlicher Sprachen nutzbar ist. Die Struktur der zurückgegebenen Abhängigkeitsgraphen, basiert auf einem “*head-modifier*”-Pattern; d. h. dass, ausgehend von einem *head*-Token, Kanten zu *modifier*-Token gehen, die die Bedeutung des *heads* verändern.

Peter's $\xleftarrow{\text{possessive nominal modifier}}$ ball $\xrightarrow{\text{adjectival modifier}}$ red

Der CoreNLP Dependency Parser nutzt ein sog. *Transition-based Parsing* [Niv04], bei dem alle Token der Reihe nach aus einem Buffer auf einen Stack von aktuell betrachteten Token gelegt werden. Ein Klassifikator (im Falle von CoreNLP ist dies ein neuronales Netz) wählt dabei in jedem Schritt einen von drei Zustandsübergängen:

1. **LEFT-ARC:** Fügt eine Abhängigkeitskante (i, j) vom ersten Token i des Stacks zum zweiten Token j des Stacks ein und entfernt dann j vom Stack.
2. **RIGHT-ARC:** Fügt eine Abhängigkeitskante (j, i) vom zweiten Token j des Stacks zum ersten Token i des Stacks ein und entfernt dann i vom Stack.
3. **SHIFT:** Verschiebt das erste Token des Buffers auf den Stack.

Diese drei Zustandsübergänge werden so lange angewandt, bis der Buffer leer ist. Durch die richtige Kombination von Übergängen lässt sich jeder beliebige Abhängigkeitsgraph beschreiben.

3.3 Modellierung von HL-MRFs mit PSL

In 3.1 wurde beschrieben, wie komplexes Wissen durch Konzeptgraphen repräsentiert werden kann; in 3.2 wurde beschrieben, wie der Inhalt natürlichsprachlicher Texte extrahiert und durch eine Menge von Abhängigkeiten repräsentiert werden kann. Dieser Abschnitt beschreibt nun, wie aus einer Menge gegebener Abhängigkeiten und Fakten neue Abhängigkeiten und Fakten inferiert werden können. Konkret werden hierfür *Hinge-Loss Markov Random Fields* (HL-MRFs) und die *Probabilistic Soft Logic* (PSL) vorgestellt.

3.3.1 Markov Random Fields

MRFs sind, so wie auch bayessche Netze, eine Klasse von *Probabilistischen Graphischen Modellen* (PGM); d. h. sie sind Graphen, deren Knoten als Zufallsvariablen und deren Kanten als stochastische Abhängigkeiten interpretiert werden. Im Gegensatz zu bayesschen Netzen, sind die Kanten in MRFs allerdings ungerichtet, es sind also zyklische Abhängigkeiten erlaubt. Formal beschreibt ein MRF G die multivariate Verteilung P eines Zufallsvektors X gemäß einer Potentialfunktion Φ :

$$\begin{aligned}
X &:= (X_1, \dots, X_n) = \text{Zufallsvektor} \\
\mathcal{X} &:= \text{Menge aller möglichen Werte } (x_1, \dots, x_n) \text{ von } X \\
G &:= (X, E) \\
\mathcal{C} &:= \{c_1, \dots, c_m\} = \text{Menge der maximalen Cliquen in } G \\
X_c &:= \text{Vektor der Zufallsvariablen in der Clique } c \in \mathcal{C} \\
\Phi_c(x_c) &:= \text{Cliquenpotential} \in \mathbb{R}_0^+ \text{ der Werte } x_c \text{ von } X_c \\
P(X = x) &= \frac{1}{Z} \prod_{i=1}^m \Phi_{c_i}(x_{c_i}), \text{ mit Normalisierkonst. } Z := \sum_{x \in \mathcal{X}} \prod_{i=1}^m \Phi_{c_i}(x_{c_i}) \quad (3.9)
\end{aligned}$$

Für MRFs gelten drei wichtige Eigenschaften bzgl. der Unabhängigkeit der Zufallsvariablen in X :

1. Globale Markov-Eigenschaft:

$$\forall X_A, X_B, X_S \subseteq X : \text{sep}_{X_A, X_B}(X_S) \rightarrow (X_A \perp X_B \mid X_S)$$

Alle Paare (X_A, X_B) von Teilmengen von X sind bedingt unabhängig, sofern die Werte einer separierenden Teilmenge X_S gegeben sind. X_S ist separierend ($\Leftrightarrow \text{sep}_{X_A, X_B}(X_S)$), wenn alle Pfade von $a \in X_A$ nach $b \in X_B$ einen Knoten $s \in X_S$ enthalten.

2. Lokale Markov-Eigenschaft:

$$\forall X_i \in X : X_i \perp (X \setminus \Gamma(X_i) \setminus \{X_i\}) \mid \Gamma(X_i)$$

Eine direkte Konsequenz der globalen Markov-Eigenschaft ist die lokale Markov-Eigenschaft. Jede Variable X_i ist bedingt unabhängig von ihren nicht benachbarten Variablen, sofern ihre Nachbarschaft $\Gamma(X_i)$ gegeben ist.

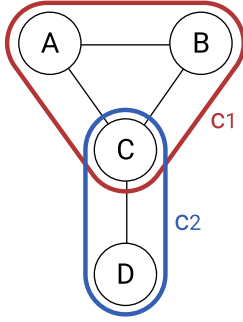
3. Paarweise Markov-Eigenschaft:

$$\forall X_i, X_j \in X : \{X_i, X_j\} \notin E \rightarrow (X_i \perp X_j \mid X \setminus \{X_i, X_j\})$$

Aus der lokalen Markov-Eigenschaft folgt, dass jedes nicht adjazente Varia-

blenpaar (X_i, X_j) bedingt unabhängig voneinander ist, sofern alle anderen Variablen gegeben sind.

Beispiel Die obigen Definitionen sind bislang noch recht abstrakt. Ein exemplarisches praktisches Einsatzgebiet von MRFs ist das Lösen von SAT-Problemen. Gegeben sei die SAT-Instanz $(\neg A \vee B \vee C) \wedge (\neg C \vee \neg D)$. MRFs können benutzt werden, um dieses Problem zu modellieren und eine erfüllende Belegung zu finden.



$$X := (A, B, C, D), \text{ mit } \text{Bild}(X) = \{0, 1\}^4$$

$$\mathcal{C} := \{\underbrace{\{A, B, C\}}_{c_1}, \underbrace{\{C, D\}}_{c_2}\}$$

$$\Phi_{c_1}(a, b, c) := \min\{1 - a + b + c, 1\}$$

$$\Phi_{c_2}(c, d) := \min\{2 - c - d, 1\}$$

$$P(X = (a, b, c, d)) := \frac{1}{Z} \Phi_{c_1}(a, b, c) \Phi_{c_2}(c, d)$$

Eine Clique repräsentiert in diesem MRF eine Disjunktionsklausel und das Cliquenpotential gibt an, ob eine gegebene Belegung die Klausel erfüllt. Gemäß dieser Definition, lässt sich die Erfüllbarkeit durch $\max_x P(X = x) > 0$ ausdrücken, d. h. die Formel ist erfüllbar, gdw. es eine Variablenbelegung mit Eintrittswahrscheinlichkeit > 0 gibt. Die Normalisierungskonstante Z hat auf das Ergebnis keinen Einfluss und kann daher ignoriert werden.

Das MRF-Inferenzproblem Da sich SAT, wie soeben exemplarisch gezeigt, auf MRFs reduzieren lässt, ist das Inferenzproblem, d. h. das Finden einer maximal wahrscheinlichen Belegung der Zufallsvariablen, ein NP-schweres Problem. Allgemeine, exakte und effiziente Lösungsalgorithmen existieren daher nicht. Durch Einschränken der Struktur von G und Φ , oder durch das erlauben von Approximationen, lassen sich MRF-Inferenzen jedoch deutlich effizienter finden. Wenn z. B. G ein Baum ist, kann mit dem *Belief Propagation* Algorithmus eine exakte Lösung in polynomieller Zeit gefunden werden.

3.3.2 Hinge-Loss MRFs

Eine Unterart von MRFs, sind die sog. Hinge-Loss MRFs. Sie sind so strukturiert, dass sich das Inferenzproblem effizient und exakt durch konvexe Optimierungsverfahren lösen lässt. Es gibt drei wesentliche Unterschiede zu allgemeinen MRFs:

1. Die Bedeutung von Zufallsvariablen und Kanten zwischen Zufallsvariablen ist klar definiert, da Φ nicht mehr frei wählbar ist. Ähnlich zum Beispiel aus 3.3.1,

repräsentieren Zufallsvariablen aussagenlogische Variablen und Cliques Disjunktionsklauseln.

2. Für den Zufallsvektor X muss $\text{Bild}(X) = [0, 1]^n$ gelten. Diese Einschränkung besteht, da jede HL-MRF-Zufallsvariable als die Wahrscheinlichkeit, dass eine aussagenlogische Variable wahr ist, interpretiert wird.
3. Die Definition der Verteilung P ist etwas anders:

$$P(X = x) := \frac{1}{Z} \prod_{i=1}^m e^{w_i \Phi_i(x)} \propto \exp \left(\sum_{i=1}^m w_i \Phi_i(x) \right) = \exp \left(w \Phi(x)^\top \right) \quad (3.10)$$

$$w := (w_1, \dots, w_m) \in [0, \infty]^m, \quad \Phi := (\Phi_1, \dots, \Phi_m)$$

Auf die Cliquespotentialle Φ_i wird nun die Exponentialfunktion angewandt, zudem erhalten alle Cliques bzw. Klauseln c_i ein Gewicht w_i . Das Inferenzproblem $\arg \max_x P(X = x)$ ist somit äquivalent zu $\arg \max_x w \Phi(x)^\top$.

KNF-Formel Interpretation Aufgrund der Einschränkung von HL-MRFs auf aussagenlogische Ausdrücke, wird im Folgenden die Graphterminologie fallen gelassen und stattdessen die entsprechende aussagenlogische Terminologie verwendet. Ein HL-MRF wird nun als Repräsentation einer KNF-Formel $C_1 \wedge \dots \wedge C_m$ interpretiert. Jede Disjunktionsklausel $C_j \in C$ wird durch eine Menge von Variablenindizes positiver Atome $I_j^+ \subseteq \{1, \dots, n\}$ und eine Menge von Variablenindizes negativer Atome $I_j^- \subseteq \{1, \dots, n\}$ beschrieben.

$$C_j \cong \left(\bigvee_{i \in I_j^+} X_i \right) \vee \left(\bigvee_{i \in I_j^-} \neg X_i \right)$$

Łukasiewicz Logik Da die Variablen der KNF-Formel, gemäß obiger Definition, Werte $\in [0, 1]$ annehmen können, ist nun noch unklar, wie die Operatoren \wedge , \vee und \neg funktionieren sollen. HL-MRFs benutzen hierfür die sog. Łukasiewicz Logik aus der Klasse der T-Norm Fuzzy Logiken; sie ist eine Erweiterung der booleschen Logik, d. h. die Łukasiewicz Operatoren verhalten sich für die Extrema 0 und 1 so, wie die booleschen Operatoren, sind aber ebenfalls für alle dazwischen liegenden Eingabewerte definiert.

$$x_1 \wedge x_2 := \max\{x_1 + x_2 - 1, 0\} \quad (3.11)$$

$$x_1 \vee x_2 := \min\{x_1 + x_2, 1\} \quad (3.12)$$

$$\neg x := 1 - x \quad (3.13)$$

Mittels der Łukasiewicz Logik können Disjunktionsklauseln $C_j \in C$ für eine gegebene

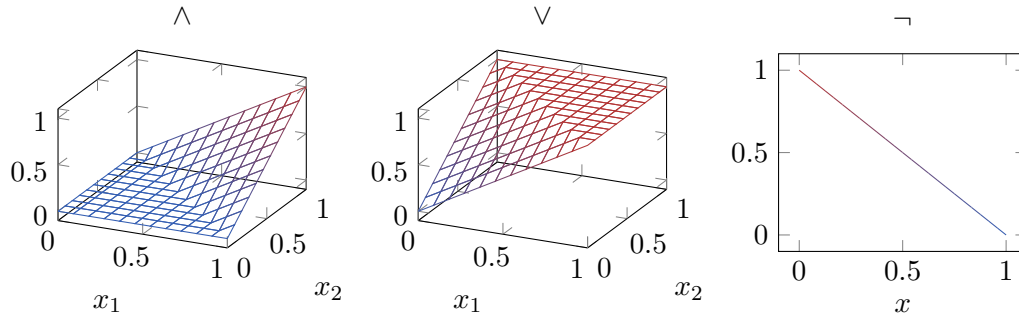


Abb. 3.3. Visualisierung der Łukasiewicz Operatoren \wedge , \vee und \neg .

Variablenbelegung x nun Wahrheitswerte $\in [0, 1]$ zugeordnet werden. Dieser Wahrheitswert wird als Klauselpotential $\Phi_j(x)$ verwendet. Das HL-MRF-Inferenzproblem für KNF-Formeln in Łukasiewicz Logik ist demnach

$$\begin{aligned}
& \arg \max_{x \in [0,1]^n} \sum_{C_j \in C} w_j \quad \Phi_j(x) \\
&= \arg \max_{x \in [0,1]^n} \sum_{C_j \in C} w_j \quad \left(\left(\bigvee_{i \in I_j^+} x_i \right) \vee \left(\bigvee_{i \in I_j^-} \neg x_i \right) \right) \\
&= \arg \max_{x \in [0,1]^n} \sum_{C_j \in C} w_j \min \left\{ \left(\sum_{i \in I_j^+} x_i \right) + \left(\sum_{i \in I_j^-} (1 - x_i) \right), 1 \right\} \quad (3.14)
\end{aligned}$$

Statt die Summe der Wahrheitswerte $\Phi_j(x)$ zu maximieren, kann alternativ auch die Summe der Distanzen zur Erfüllung $\ell_j(x)$, genannt *Distance to Satisfaction*, minimiert werden; es gilt $\ell_j(x) = 1 - \Phi_j(x)$. Gemäß dieser Interpretation ist ein HL-MRF somit eine Menge von gewichteten Constraints $\ell_j(x) \leq 0$, für die eine Lösung mit möglichst wenigen Verletzungen gesucht wird. Das Inferenzproblem lässt sich also das Finden des folgenden Minimums beschreiben:

$$\begin{aligned}
& \arg \min_{x \in [0,1]^n} \sum_{C_j \in C} w_j \max\{\ell_j(x), 0\} \\
&= \arg \min_{x \in [0,1]^n} \sum_{C_j \in C} w_j \max \left\{ 1 - \left(\sum_{i \in I_j^+} x_i \right) - \left(\sum_{i \in I_j^-} (1 - x_i) \right), 0 \right\} \quad (3.15)
\end{aligned}$$

Wie Abbildung 3.4 für den zwei-elementigen Klauselfall veranschaulicht, handelt es sich bei ℓ um eine Hinge-Loss Funktion. Hierher rührt die Bezeichnung Hinge-Loss MRF. Da Hinge-Loss Funktionen konvex sind und die Summe konvexer Funktionen ebenfalls konvex ist, handelt es sich bei der HL-MRF-Inferenz um ein konvexes Optimierungsproblem. Es existieren also effiziente und exakte Lösungsalgorithmen. Einer dieser Algorithmen ist das *Alternating Direction Method of Multipliers* Verfahren

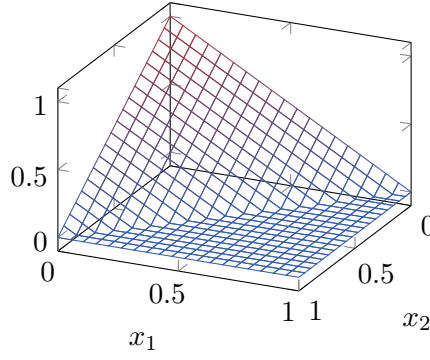


Abb. 3.4. Visualisierung der Loss-Funktion $\ell_j(x_1, x_2)$ für $C_j \cong X_1 \vee X_2$

(ADMM), es wird in 3.3.4 näher vorgestellt.

MAX-SAT Äquivalenz In 3.3.1 wurden MRFs anhand des Beispiels der SAT-Instanz $(\neg A \vee B \vee C) \wedge (\neg C \vee \neg D)$ veranschaulicht. Diese KNF-Formel hat folgendes Inferenzproblem, wenn sie als HL-MRF repräsentiert wird:

$$\arg \min_{(a,b,c,d) \in [0,1]^4} w_1 \max\{a - b - c, 0\} + w_2 \max\{c + d - 1, 0\}$$

Da in der Verteilung P von HL-MRFs die Exponentialfunktion auf die Potentiale angewandt wird, bewirkt eine unerfüllte Klausel mit $\Phi_j(x) = 0$ nicht, dass $P(X = x) = 0$. Stattdessen ist $P(X = x)$ proportional zu der Summe der Wahrheitswerte der Klauseln. Die HL-MRF-Inferenz beschreibt also nicht SAT, sondern eine Fuzzy-Logik-Entsprechung von MAX-SAT. Ein wichtiger Unterschied zum booleschen MAX-SAT ist, dass Klauseln gewichtet sind; das Erfüllen einer Klausel mit hohem Gewicht kann das Nicht-Erfüllen mehrerer anderer Klauseln mit niedrigem Gewicht ausgleichen.

3.3.3 Probabilistic Soft Logic

Wie soeben beschrieben, sind HL-MRFs ein flexibles Werkzeug, um Probleme, die sich durch MAX-SAT ausdrücken lassen, zu lösen. Der Schritt von einem konkreten domänenspezifischen Problem in eine Menge von Klauseln C und einen Gewichtsvektor w ist bislang allerdings noch unklar. An dieser Stelle setzt die *Probabilistic Soft Logic* (PSL) an. PSL ist eine formale Sprache, um mit einer intuitiven Syntax Klassen von HL-MRFs zu beschreiben.

PSL Syntax

Die Syntax von PSL ist an die Prädikatenlogik angelehnt und besteht aus sieben Arten von Elementen:

1. **Konstanten:** Repräsentieren konstante Werte, wie z. B. Strings oder Zahlen. Werden für domänenspezifische Daten, wie z. B. Namen benutzt.
2. **Variablen:** Werden während einer Inferenz mit Konstanten belegt. PSL Variablen sind nicht zu verwechseln mit den Zufallsvariablen in MRFs.
3. **Terme:** Ein Term ist entweder eine Konstante oder eine Variable.
4. **Prädikate:** Entsprechen in etwa den prädikatenlogischen Prädikaten. Jedes PSL Prädikat hat einen eindeutigen Bezeichner und eine Signatur aus Konstantentypen.

$$Person : \text{UUID}, \quad Name : \text{UUID} \times \text{String}$$

5. **Atome:** Ein Atom ist ein Prädikat der Arität n , kombiniert mit einem n -Tupel von Termen. Das Term-Tupel enthält die Argumente des Prädikates. Wenn alle Argumente Konstanten sind, spricht man von einem Grundatom (*ground atom*).

$$Person(x), \quad Name(x, \text{"Alice"})$$

6. **Literale:** Ein Literal ist entweder ein Atom oder ein negiertes Atom.

$$Name(x, \text{"Alice"}), \quad \neg Name(x, \text{"Bob"})$$

7. **Regeln:** Eine Regel ist eine gewichtete Disjunktionsklausel von Literalen. Die negativen Atome der Klausel bilden dabei den sog. Körper B (*body*), die positiven Atome den sog. Kopf H (*head*) der Regel. Die so zerlegte Disjunktionsklausel lässt sich als Implikationsregel interpretieren:

$$\left(\bigvee_{b \in B} \neg b \right) \vee \left(\bigvee_{h \in H} h \right) \Leftrightarrow \left(\bigwedge_{b \in B} b \right) \rightarrow \left(\bigvee_{h \in H} h \right)$$

Mit Implikationen lassen sich nun intuitiv Zusammenhänge zwischen Prädikaten modellieren.

$$0.65 : Person(x) \wedge Name(x, \text{"Alice"}) \rightarrow Female(x)$$

Eine Menge von PSL-Regeln wird PSL-Programm genannt. Als Input erwartet ein PSL-Programm R eine sog. Basis \mathcal{A} . Die Basis ist dabei eine Menge von Grundatomen, die während der Inferenz in Betracht gezogen werden sollen, und setzt sich aus zwei disjunkten Teilmengen $\mathbb{C} \dot{\cup} \mathbb{O} = \mathcal{A}$ zusammen. \mathbb{C} ist die Menge der geschlossenen (*closed*) Grundatome, d. h. der Wahrheitswert $\in [0, 1]$ dieser Atome ist bekannt. \mathbb{O} umfasst die offenen (*open*) Grundatome, deren Wahrheitswert noch unbekannt ist

und daher inferiert werden soll.

Beispiel Ein Anwendungsgebiet von PSL ist z. B. die Modellierung sozialer Beziehungen. Angenommen, es sollen Freundschaftsbeziehungen zwischen Personen inferiert werden. Ein stark vereinfachtes PSL-Programm hierfür könnte so aussehen:

$$0.4 : \neg \text{Friends}(A, B) \quad (r_1)$$

$$0.5 : \text{Friends}(A, B) \wedge \text{Friends}(B, C) \wedge A \neq C \rightarrow \text{Friends}(A, C) \quad (r_2)$$

$$1 : \text{Interest}(A, X) \wedge \text{Interest}(B, X) \rightarrow \text{Friends}(A, B) \quad (r_3)$$

$$\infty : \text{Friends}(A, B) \rightarrow \text{Friends}(B, A) \quad (r_4)$$

$$\infty : \neg \text{Friends}(A, A) \quad (r_5)$$

In dieser Regelmenge sind fünf Annahmen über das Verhalten der *Friends*-Relation kodiert:

- (r_1) bildet ab, dass zwei Personen a priori nicht miteinander befreundet sind. Eine solche PSL-Regel, die, in Ermangelung weiteren Wissens, das Nichtvorhandensein einer Relation postuliert, wird *Prior* genannt.
- (r_2) bildet die Transitivität der *Friends*-Relation ab. Zwei Personen mit einem gemeinsamen Freund, sind evtl. befreundet. Das Infix-Prädikat \neq ist üblicherweise vordefiniert.
- (r_3) bildet ab, dass Personen mit einem gemeinsamen Interesse eine höhere Wahrscheinlichkeit haben befreundet zu sein.
- (r_4) bildet die Symmetrie der *Friends*-Relation ab. Im Gegensatz zu den anderen Regeln, ist r_4 ein sog. *Constraint*, d. h. sie hat ein Gewicht $w_4 = \infty$. Das Nichterfüllen von r_4 hat somit zur Folge, dass die gewichtete Distance to Satisfaction ebenfalls den Wert ∞ annimmt und durch die Erfüllung anderer Regeln nicht ausgeglichen werden kann. Es ist also garantiert, dass jede MAX-SAT Lösung r_4 einhält; d. h. $\text{Friends}(A, B) = \text{Friends}(B, A)$.
- (r_5) ist ein weiterer Constraint, der die Irreflexivität von *Friends* erzwingt.

Für das PSL-Programm $R = \{r_1, \dots, r_5\}$ sei nun folgender Input \mathcal{A} gegeben:

$$\mathbb{C} = \left\{ \begin{array}{ll} \text{Interest}(\text{"Alice"}, \text{"Reading"}) = 0.5, & \text{Interest}(\text{"Dave"}, \text{"Reading"}) = 1, \\ \text{Interest}(\text{"Alice"}, \text{"Skiing"}) = 0.5, & \text{Interest}(\text{"Charlie"}, \text{"Skiing"}) = 1, \\ \text{Interest}(\text{"Bob"}, \text{"Tennis"}) = 1, & \text{Interest}(\text{"Charlie"}, \text{"Tennis"}) = 1, \\ \text{Friends}(\text{"Alice"}, \text{"Bob"}) = 1 \end{array} \right.$$

$$\mathbb{O} = \{ \text{Friends}(x, y) = ? : x, y \in \{\text{"Alice"}, \text{"Bob"}, \text{"Charlie"}, \text{"Dave"}\} \} \setminus \mathbb{C}$$

Die Interessen der Personen und die Freundschaft zwischen Bob und Alice aus \mathbb{C} werden als bekannt angenommen. Die gesuchten Freundschaftsrelationen aus \mathbb{D} sollen inferiert werden. Abbildung 3.5 zeigt ein mögliches MAX-SAT Ergebnis

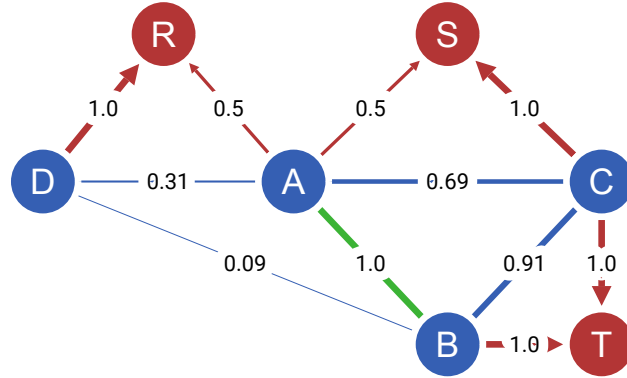


Abb. 3.5. Graph der inferierten *Friends-Relation* und der gegebenen *Interest-Relation*.

der Inferenz. Wegen des Priors r_1 werden Personen generell als nicht befreundet eingeordnet. Personen mit gemeinsamen Interessen werden hingegen, aufgrund von r_3 , als Freunde erkannt. Die durch r_2 modellierte Transitivitätseigenschaft ist ein weiterer verstärkender Faktor. Dies lässt sich in der Clique *Alice-Bob-Charlie* beobachten, die *Friends*-Beziehungen haben sich dort gegenseitig verstärkt.

PSL \rightarrow HL-MRF Übersetzung

Wie soeben gezeigt, ist PSL ein intuitiver Formalismus zur Definition relationaler Modelle. Um Inferenzen durchzuführen, ist allerdings zuerst eine Übersetzung in ein HL-MRF notwendig. Gegeben ist hierbei immer ein PSL-Programm $R = \{r_1, \dots, r_k\}$ und eine Eingabe $\mathcal{A} = \mathbb{C} \cup \mathbb{D}$. Die Übersetzung erfolgt in zwei Schritten:

1. **Zufallsvariablen:** Die Grundatome aus \mathbb{C} werden zu Zufallsvariablen X , deren Belegung x gegeben ist. Die Grundatome aus \mathbb{D} werden zu Zufallsvariablen Y , für die die optimale Belegung $y \in \mathcal{Y}$ aus der Menge aller möglichen Belegungen \mathcal{Y} gesucht wird. Der Zufallsvektor des HL-MRFs ist also $X \cup Y$.
2. **Disjunktionsklauseln:** In der bisherigen HL-MRF Definition wurde stets davon ausgegangen, dass die optimale Belegung *aller* Zufallsvariablen gesucht ist. Im Falle von PSL ist allerdings bereits bekannt, dass $X = x$ gilt. Das Inferenzproblem lautet daher wie folgt:

$$\begin{aligned} \arg \max_{y \in \mathcal{Y}} P(Y = y \mid X = x) &= \arg \max_{y \in \mathcal{Y}} P(X \cup Y = x \cup y) \\ &= \arg \max_{y \in \mathcal{Y}} w \Phi(y, x)^\top \end{aligned} \quad (3.16)$$

Um ein HL-MRF zu erhalten, muss nun Φ und w , d. h. die Menge der Klauseln C und ihre Gewichte, definiert werden. Die PSL-Regeln in R sind zwar Klauseln, können aber nicht direkt als C verwendet werden, da sie potentiell freie PSL-Variablen enthalten, die PSL-Atome also potentiell nicht in \mathcal{A} sind. Es erfolgt daher ein sog. *Grounding* der PSL-Regeln. In jede Regel aus R wird dabei jede mögliche Belegung der PSL-Variablen eingesetzt, sodass die resultierende Regelinstanz nur Atome aus \mathcal{A} enthält. Die Menge dieser Regelinstanzen wird Grundregeln (*ground rules*) genannt und als Klauselmengen C benutzt. Aus C wiederum lässt sich Φ nun mittels der Łukasiewicz Logik eindeutig ableiten. Das Gewicht jedes Potentials Φ_i ist dabei gleich dem Gewicht der PSL-Regel aus der es resultierte.

3.3.4 Inferenzverfahren

In Abschnitt 3.3.2 wurde die Konvexität des HL-MRF-Inferenzproblems bereits diskutiert. Prinzipiell kann daher jedes konvexe Optimierungsverfahren im Kontext von HL-MRFs verwendet werden. In der Praxis hat sich jedoch ein Verfahren, als besonders effizient erwiesen.

ADMM Das sog. *Alternating Direction Method of Multipliers* Verfahren (ADMM) ist eine Variante des *Method of Multipliers* Verfahrens, in der die dualen Variablen partiell aktualisiert werden. Durch diese Variation lässt sich ADMM gut parallelisieren und ist somit geeignet für große Datenmengen und den Einsatz in Cluster-Umgebungen. Das ursprüngliche Paper [Boy+11, Kapitel 10] beschreibt explizit, wie sich ADMM mit verteilten Datensets und verteilten Programmiermodellen wie z. B. *Map-Reduce* oder *Pregel* implementieren lässt.

Die ADMM-Laufzeit ist proportional zur Klauselanzahl $|C|$. Die Klauselanzahl wiederum wächst gemäß $\mathcal{O}(|\mathcal{A}|^r)$, wobei r die maximale Anzahl von Nicht-Grund-Atomen in einer PSL-Regel ist. Insgesamt wächst die Inferenzdauer also polynomiell in Abhängigkeit zur Eingabe \mathcal{A} .

BOCI Um effiziente Updates eines gegebenen Inferenzergebnisses für eine veränderte Eingabe \mathcal{A}' zu ermöglichen, wurde das *Budgeted Online Collective Inference* (BOCI) Verfahren entwickelt. Wenn ein Inferenzergebnis für $\mathcal{A} = \mathbb{C} \dot{\cup} \mathbb{O}$ existiert und anschließend die Wahrheitswerte für einen Teil der bislang offenen Atome aus \mathbb{O} bekannt werden, muss mittels BOCI keine weitere vollständige Inferenz durchgeführt werden.

Die Grundidee dabei ist es Metadaten, die während der letzten ADMM-Inferenz angefallen sind, für eine Bewertung jedes Atoms zu benutzen. Die Bewertungen spiegeln die Volatilität der Wahrheitswerte der Atome, in Abhängigkeit von \mathbb{C} , wider.

Welche Metadaten für die Bewertung verwendet werden sollten, hängt stark von der Semantik der Wahrheitswerte der PSL-Prädikate ab. In [Puj+15, Kapitel 4] werden verschiedene mögliche Bewertungsverfahren beschrieben, für diese Arbeit sind jene allerdings nicht näher relevant.

Gegeben sei also eine Bewertung der Atome aus \mathcal{A} . Werden nun die Wahrheitswerte bislang offener Atome aus \mathbb{O} bekannt, müssen ausschließlich die m höchstbewerteten Atome aus \mathbb{C} neu inferiert werden, die Wahrheitswerte aller anderen Atome werden fixiert. Je höher das sog. Budget m , desto höher ist die Qualität im Vergleich zu einer vollständigen Inferenz. Es wurde empirisch gezeigt, dass die Inferenzqualität mit BOCI gegenüber einer vollständigen Inferenz meist nur unwesentlich schlechter ist, während sich die Inferenzdauer teils um über 60% verringert [Puj+15, Kapitel 5].

Wissensgraphkonstruktion aus Kommunikationsdaten

Auf Basis der vorgestellten Konzeptgraphen, CoreNLP und PSL wird im folgenden Kapitel ein Verfahren für die online Konstruktion eines Wissensgraphen aus natürlichsprachlichen Textnachrichten aufgebaut. Der Fokus liegt dabei primär auf der generellen Architektur des Verfahrens. Das Resultat ist also als Proof-of-Concept zu verstehen, auf dessen Basis praxistaugliche Systeme konzipiert werden können.

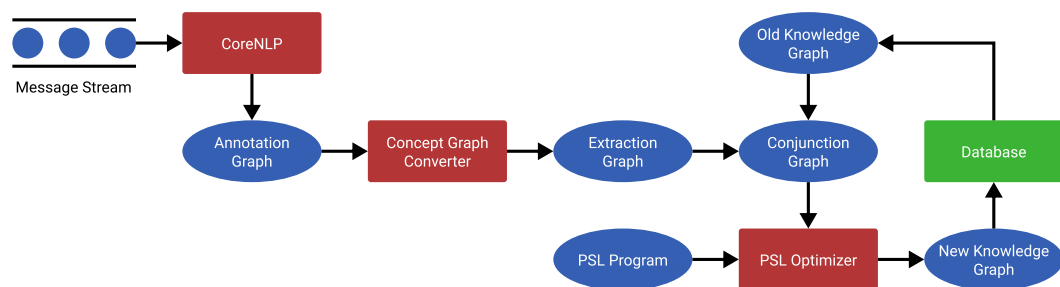


Abb. 4.1. Grobes Architekturdiagramm der Konstruktionspipeline

Das im Folgenden vorgestellte Verfahren folgt einem dreistufigen Pipeline-Modell:

1. Mittels CoreNLP wird eine eintreffende Textnachricht in einen Abhängigkeitsgraphen transformiert.
2. Der resultierende Abhängigkeitsgraph wird in einen sog. Extraktionsgraphen umgewandelt. Hierbei handelt es sich um einen Konzeptgraphen, der den Inhalt der Nachricht formal repräsentiert.
3. Der Extraktionsgraph wird mit dem bestehenden Wissensgraphen verschmolzen. Dies entspricht der Konjunktion der durch die beiden Graphen repräsentierten logischen Ausdrücke. Der resultierende Konjunktionsgraph wird als Eingabe für ein PSL-Programm verwendet, welches auf Basis des hinzugekommenen Wissens neue Beziehungen im Wissensgraphen inferiert.

Die Beschreibung dieser Pipeline erfolgt in vier Abschnitten. Abschnitt 4.1 beschreibt die Ontologie der konstruierten Wissensgraphen. Nachdem beschrieben ist, wie die zu konstruierenden Wissensgraphen strukturell aufgebaut sein sollen, wird schließlich in Abschnitt 4.2 und Abschnitt 4.3 die Transformation von Text zu Extraktionsgraph, bzw. von Extraktionsgraph zu Wissensgraph beschrieben. In Abschnitt 4.4 wird abschließend kurz die technische Umsetzung des zuvor beschriebenen Verfahrens erläutert.

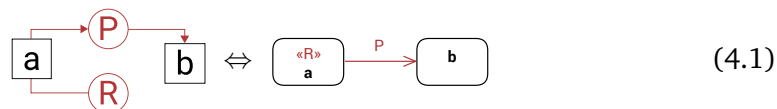
4.1 Wissensgraphontologie

Bevor Wissensgraphen konstruiert werden können, muss spezifiziert sein, wie genau diese aussehen sollen. Um komplexe logische Beziehungen ausdrücken zu können, wird die bereits vorgestellte Konzeptgraph-Struktur verwendet. Dabei bleibt allerdings offen, welche Prädikate vorkommen können und welche Bedeutung sie haben. Außerdem ist unklar, wie mit Konzeptgraphen modale Aussagen ausgedrückt werden. Damit aus einem Konzeptgraphen ein Wissensgraph wird, muss eine Ontologie gegeben sein, welche diese offenen Punkte schließt. Im Folgenden wird beschrieben, wie genau die in dieser Arbeit verwendete Ontologie aufgebaut ist.

4.1.1 Verwendete Prädikate

Im ersten Schritt wird geklärt, welche Prädikate in den in dieser Arbeit konstruierten Wissensgraphen vorkommen können. Es ist nicht sinnvoll beliebige Prädikate zuzulassen, da für eine effiziente maschinelle Weiterverarbeitung der Wissensgraphen, eine Semantik für alle vorkommenden Prädikate definiert sein muss. Für die Wissensgraphen in dieser Arbeit wird daher eine Menge von sechs Prädikaten verwendet.

Syntax Bevor diese Prädikate näher erläutert werden, wird allerdings die Konzeptgraphsyntax leicht modifiziert. Da alle verwendeten Prädikate entweder unär oder binär sein werden, kann eine kompaktere Notation benutzt werden:



Relationsknoten werden nicht mehr dargestellt. Bei unären Relationen werden die Relationsknotenbezeichner stattdessen mit in die Konzeptknoten geschrieben. Binäre Relationen werden durch eine bezeichnete Kante zwischen dem ersten und zweiten Argument repräsentiert. Der nun nicht mehr explizit dargestellte Relationsknoten befindet sich dabei implizit im Kontext des kleinsten Argumentes gemäß \leq -Relation, sodass alle dominierenden Knoten erhalten bleiben.

Wissensmodell Um eine semantisch konsistente Menge von Wissensgraph-Prädikaten zu definieren, ist es sinnvoll zuvor ein Modell zu definieren, welches den abstrakten Begriff des *Konzeptes* formalisiert. Konzepte sind das zentrale Syntaxelement in Konzeptgraphen; ähnlich wie Variablen in der Prädikatenlogik, haben sie jedoch keine inhärente Bedeutung.

Im Folgenden wird ein Konzept x durch eine Eigenschaftsmenge $prop(x) = \{p_1, \dots, p_n\}$ spezifiziert. Prädikate die für x gelten bzw. nicht gelten, beschreiben Eigenschaften von $prop(x)$. Es ist unrealistisch $prop(x)$ mittels Prädikaten vollständig zu definieren,

da dies gleichbedeutend mit einer exakten formalen Beschreibung jedes beliebigen Konzeptes x wäre. Das Ziel ist daher stattdessen, die Eigenschaftsmengen von Konzepten in Relation zueinander zu beschreiben. Der Wissensgraph ist also ein Formalismus zur Beschreibung von Eigenschaften der Eigenschaftsmengen von Konzepten.

Prädikate Abhängig davon, welche Eigenschaften erfasst werden sollen, können die zu verwendenden Prädikate stark variieren. Die für diese Arbeit verwendete Prädikatsmenge hat nicht das Ziel der Vollständigkeit, sondern versucht vielmehr einige wesentliche Eigenschaften natürlichsprachlicher Aussagen zu erfassen. Gemäß dieser Prämisse wurden sechs Wissensgraph-Prädikate definiert.

1. $label \subseteq \mathbf{concept} \times \mathbf{string}$: Das *label*-Prädikat wird benutzt, um den Bezeichner eines Konzeptes zu repräsentieren. Jedes Konzept hat höchstens einen Bezeichner. Da Konzepte aus natürlichsprachlichen Texten extrahiert werden, kann i. d. R. allen Konzepten ein solcher Bezeichner zugeordnet werden. In der bisher verwendeten Konzeptgraphsyntax lassen sich derartige Konstanten allerdings nicht ausdrücken. Die Syntax wird daher erneut leicht modifiziert. Das *label* eines Konzeptes wird nun, statt des Variablenbezeichners, in den zugehörigen Konzeptknoten geschrieben.

$$\boxed{\text{book}} \Leftrightarrow \exists x : label(x, \text{"book"}) \quad (4.2)$$

2. $named \subseteq \mathbf{concept}$: Die Bedeutung des durch *label* beschriebenen Bezeichners kann stark variieren. Für die meisten Konzepte ist das *label* ein Verweis auf ein natürlichsprachliches Wort, dessen Bedeutung übernommen wird. Manche Konzepte, wie z. B. Personen, Städte oder Firmen, lassen sich darüber hinaus jedoch durch ihren Bezeichner identifizieren; diese Konzepte erhalten das Attribut *named*.
3. $inst \subseteq \mathbf{concept}^2$: Die *inst*-Beziehung zwischen Konzepten entspricht grob der IS-A-Beziehung semantischer Netze. Das Vorhandensein von $inst(x, y)$ impliziert, dass $prop(x) \subseteq prop(y)$ gilt; *inst* ist also reflexiv und transitiv.

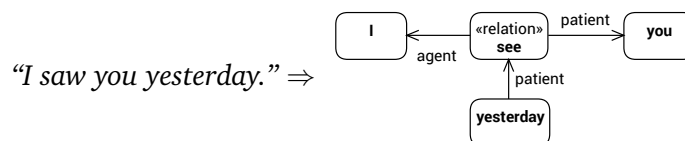
$$\text{"the good book"} \Rightarrow \boxed{\text{good}} \xrightarrow{\text{inst}} \boxed{\text{book}}$$

4. $relation \subseteq \mathbf{concept}$: Da die Prädikatsmenge eingeschränkt ist, können keine eigenen Prädikate für Worte eingeführt werden, die Konzepte in Relation zueinander setzen (z. B. $X \text{ likes } Y$). Stattdessen werden sog. Relationskonzepte verwendet, d. h. Konzepte, die andere Konzepte zueinander in Beziehung setzen. Relationskonzepte r werden durch das unäre Prädikat $relation(r) \Leftrightarrow 'relation \in prop(r)$ gekennzeichnet. Um die durch r zueinander in Beziehung

gesetzten Konzepte zu beschreiben werden die Prädikate *agent* und *patient* verwendet.

5. *agent* \subseteq **concept**²: Das *agent*-Prädikat zwischen einem Konzept x und dem sog. Agens y wird benutzt, um eine kausale Abhängigkeit des x von y zu beschreiben. $agent(x, y) \Leftrightarrow ('cause, y) \in prop(x)$ sagt also aus, dass x aufgrund von y existiert.
6. *patient* \subseteq **concept**²: Der Patiens y eines Konzeptes x ist ein Konzept, dessen Eigenschaften durch x verändert werden. Es wird dabei davon ausgegangen, dass x eine Eigenschaft $p_x \in prop(x)$ hat, die beschreibt, wie x andere Konzepte verändert. $patient(x, y)$ impliziert, dass $prop(y)$ die durch p_x geforderten Eigenschaften hat. p_x lässt sich für natürlichsprachliche Konzepte offensichtlich nicht sinnvoll formal definieren. Dies ist allerdings auch nicht notwendig, wenn p_x als mittels *label* gegebenes Domänenwissen betrachtet wird, welches erst bei der Interpretation der Daten eingebracht wird.

Die Prädikate *agent* und *patient* sind grob an die linguistische Idee des Proto-Agens und Proto-Patiens angelehnt [Dow91]. Zusammen mit den anderen Prädikaten haben sie sich als hinreichend mächtig erwiesen, um eine Vielzahl natürlichsprachlicher Aussagen abzubilden.



4.1.2 Modale Kontexte

Nachdem nun die verwendeten Prädikate beschrieben wurden, wird im zweiten Schritt geklärt, wie Wissensgraphen Modalität repräsentieren. Beispiele für modale Aussagen sind:

"I *think* that *the book is good*."
 "I *don't want* to *see you*."

Die beiden Teilaussagen "*the book is good*" und "*I see you*" sind offensichtlich nicht unbedingt wahr, sondern beschreiben Möglichkeiten, die in Abhängigkeit von *think* bzw. *don't want* wahr oder falsch werden könnten. Ob die Aussagen wahr werden, hängt davon ab, inwiefern das Denken oder Wollen einer Person mit der Realität — auch aktuelle Welt genannt — übereinstimmt. Dies ist je nach Person stark verschieden; manche tendieren dazu die allgemein als wahr akzeptierten Aussagen zu denken bzw. entsprechend ihrer Wünsche zu handeln, andere nicht.

Um derartige Möglichkeiten und Notwendigkeiten direkt zu repräsentieren, reichen die vorgestellten Konzeptgraphen mit Negationskontexten nicht aus. Sowa hat dieses Problem ebenfalls erkannt und daher weitere Kontexttypen eingeführt. Die Beschreibung dieser zusätzlichen Kontexttypen ist allerdings oftmals zu unpräzise für eine eindeutige Übersetzung in die Prädikatenlogik. Aufbauend auf Sowas Ideen wird in dieser Arbeit daher eine Variante modaler Kontexte eingeführt, die die Übersetzbarkeit in die Prädikatenlogik erster Ordnung bewahrt. Da die zuvor vorgestellten Konzeptgraphen bereits vollständig und korrekt sind, handelt es sich bei diesen modalen Kontexten also lediglich um eine Kurzschreibweise. Insgesamt gibt es durch die Erweiterung um modale Kontexte nun vier Kontexttypen:

1. Positiver Aktualkontext
2. Negativer Aktualkontext
3. Positiver Möglichkeitskontext
4. Negativer Möglichkeitskontext

Die modale Notwendigkeit hat keine eigenen Kontexttypen erhalten, um konsistent zur Quantisierung in Konzeptgraphen zu bleiben. So, wie der Allquantor in Konzeptgraphen durch Negation der negierten existenzquantisierten Aussage ausgedrückt wird, wird die Notwendigkeit durch Negation der negativen Möglichkeit ausgedrückt.

Da es nun vier Kontexttypen gibt, muss die Konzeptgraphsyntax leicht angepasst werden, um zwischen den verschiedenen Typen differenzieren zu können:

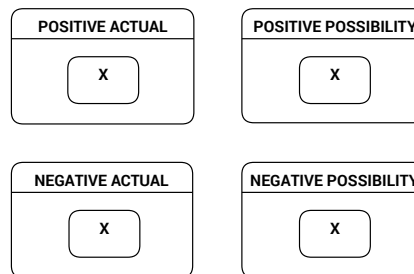


Abb. 4.2. Konzeptgraphsyntax für modale Kontexte

Positiver Aktualkontext Dieser Kontexttyp hat keinen Einfluss auf die Bedeutung der enthaltenen Knoten. Er entspricht in etwa der Klammerung in der Prädikatenlogik. Das *Sheet of Assertion* \top ist ein solcher Kontext, abgesehen davon tauchen positive Aktualkontexte allerdings nicht auf; sie werden hier lediglich der Vollständigkeit halber erwähnt.

$$\begin{array}{|c|} \hline \text{POSITIVE ACTUAL} \\ \hline \text{X} \\ \hline \end{array} \Leftrightarrow \text{X} \Leftrightarrow \exists x : \text{label}(x, \text{"X"}) \quad (4.3)$$

Negativer Aktualkontext Entspricht dem bisherigen Negationskontext.

$$\begin{array}{|c|} \hline \text{NEGATIVE ACTUAL} \\ \hline \boxed{x} \\ \hline \end{array} \Leftrightarrow \neg \exists x : \text{label}(x, \text{"X"}) \quad (4.4)$$

Positiver Möglichkeitskontext Dieser Kontexttyp bildet die modale Möglichkeit ab. Um die Übersetzbarkeit in die Prädikatenlogik beizubehalten, wird hierfür keine fundamental neue Struktur eingeführt. Es wird stattdessen die Tatsache ausgenutzt, dass sich die modalen Operatoren gemäß der Mögliche-Welten-Interpretation analog zu den prädikatenlogischen Quantoren verhalten.

$$\begin{aligned} \Box P(x) &\Leftrightarrow \neg \Diamond \neg P(x) \\ \forall w : \text{world}(w) \rightarrow P_w(x) &\Leftrightarrow \neg \exists w : \text{world}(w) \wedge \neg P_w(x) \end{aligned} \quad (4.5)$$

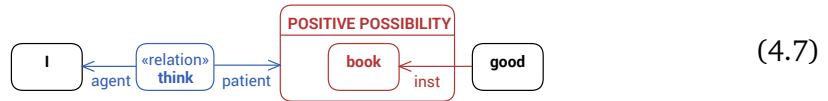
Die Aussage “ $P(x)$ gilt notwendigerweise”, wird also als “Es gibt keine Welt w in der $P_w(x)$ nicht gilt” aufgefasst. Statt von Welten, wird in modalen Konzeptgraphen allerdings von Kontexten gesprochen.

$$\begin{array}{|c|} \hline \text{POSITIVE POSSIBILITY} \\ \hline \boxed{x} \\ \hline \end{array} \Leftrightarrow \begin{array}{|c|} \hline \text{NEGATIVE ACTUAL} \\ \hline \begin{array}{|c|} \hline \text{NEGATIVE ACTUAL} \\ \hline \boxed{x} \\ \hline \end{array} \quad \begin{array}{|c|} \hline \text{«actual»} \\ \hline - \\ \hline \end{array} \end{array} \quad - \quad \begin{array}{|c|} \hline \text{NEGATIVE ACTUAL} \\ \hline \begin{array}{|c|} \hline \text{NEGATIVE ACTUAL} \\ \hline \begin{array}{|c|} \hline \text{«actual»} \\ \hline - \\ \hline \end{array} \\ \hline \boxed{x} \\ \hline \end{array} \end{array} \quad (4.6)$$

$$\Leftrightarrow \exists c : \text{context}(c) \wedge (\text{actual}(c) \leftrightarrow \exists x : \text{label}(x, \text{"X"}))$$

Anders als Aktualkontexte, tauchen Möglichkeitskontexte explizit in der prädikatenlogischen Repräsentation auf, sie lassen sich also nicht nur als Kontext sondern zugleich auch als Konzept auffassen. Der in einem Möglichkeitskontext c enthaltene Teilgraph G ist dabei genau dann wahr, wenn der Kontext die aktuelle Welt beschreibt ($\Leftrightarrow \text{actual}(c)$). Solange $\text{actual}(c)$ unbekannt ist, kann also keine Aussage über die Wahrheit von G gemacht werden. Wenn $\text{actual}(c)$ jedoch wahr bzw. falsch ist, ist c äquivalent zu einem positiven bzw. negativen Aktualkontext.

Zur Bestimmung von des Wahrheitswertes von $\text{actual}(c)$ ist i. d. R. domänenspezifisches Wissen notwendig. Dieses domänenspezifische Wissen, kann durch sog. Kontextrelationen eingebracht werden. Da Möglichkeitskontexte zugleich Konzepte sind, lassen sie sich zu anderen Konzepten in Relation setzen. Dies ist z. B. dann nützlich, wenn man die Aussage “*I think that the book is good.*” repräsentieren möchte.



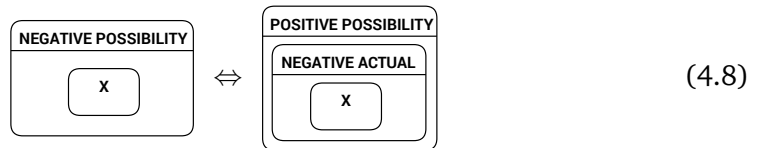
$$\Leftrightarrow \exists i, g, t, c : \text{label}(i, \text{"I"}) \wedge \text{label}(g, \text{"good"})$$

$$\wedge \text{label}(t, \text{"think"}) \wedge \text{relation}(t) \wedge \text{agent}(t, i) \wedge \text{patient}(t, c)$$

$$\wedge \text{context}(c) \wedge (\text{actual}(c) \leftrightarrow \exists b : \text{label}(b, \text{"book"}) \wedge \text{inst}(g, b))$$

Durch die Verknüpfung $\text{patient}(t, c)$ eines Konzeptes mit einem Kontext, kann Wissen über das Konzept benutzt werden, um Schlüsse über die Aktualität des Kontextes zu ziehen. Wie genau dies ablaufen kann, ist bislang allerdings noch unklar. In Abschnitt 4.3 wird die *actual*-Inferenz näher untersucht.

Negativer Möglichkeitskontext Lediglich eine Kurzschreibweise für einen positiven Möglichkeitskontext der einen negativen Aktualkontext umgibt.



4.2 NLP-Phase

In diesem Abschnitt werden die ersten beiden Stufen der Konstruktionspipeline beschrieben. Die erste Stufe ist im Wesentlichen lediglich ein Wrapper um verschiedene CoreNLP-Annotatoren. In der zweiten Stufe werden dann die CoreNLP-Annotationen in eine Konzeptgraphinstanz der in 4.1 beschriebenen Ontologie transformiert.

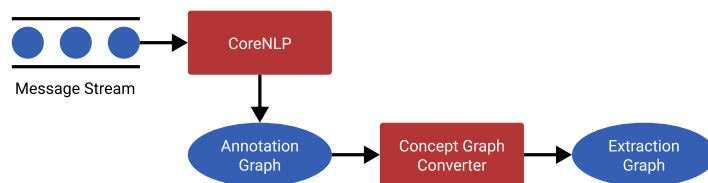


Abb. 4.3. NLP-Phase der Konstruktionspipeline

4.2.1 Nachrichtenformat

Vor der Beschreibung der Pipeline-Stufen, muss geklärt werden, wie genau die eingegebenen Nachrichten aufgebaut sind. Da diese Arbeit lediglich die grundlegende Architektur eines Wissensgraphkonstruktionsverfahrens beschreibt, wurde eine vereinfachte Struktur gewählt, die die wesentlichen Eigenschaften von Nachrichten

enthält. Eine Nachricht wird daher im Folgenden als ein Tupel mit den folgenden Komponenten verstanden:

1. **Sender:** Als Sender wird, der Einfachheit halber, der Name des Absenders verwendet. Im Kontext von E-Mail Nachrichten müsste zwar zwischen dem Absendernamen und der Absenderadresse differenziert werden, diese Unterscheidung macht das Verfahren allerdings lediglich komplizierter, ohne dabei die Kernaspekte zu bereichern.
2. **Empfänger:** Auch hier wird lediglich der Empfängername benutzt. Nachrichten mit mehreren Empfängern werden in dieser Arbeit nicht betrachtet.
3. **Sendezeit:** Gemeint ist hiermit ein Datum oder Zeitpunkt im TIMEX3-Format (z. B. YYYY-MM-DD), welches den Absendezeitpunkt der Nachricht beschreibt. TIMEX3 unterstützt höchstens sekundengenaue Angaben und keine Zeitzonen. Für die Zwecke dieser Arbeit sind diese Einschränkungen allerdings nicht problematisch.
4. **Inhalt:** Der Nachrichteninhalt wird als natürlichsprachliche Zeichenkette verstanden. Als Sprache wird dabei Englisch angenommen, da CoreNLP dann die besten Ergebnisse liefert. Zudem werden eingebundene Multimedia-Inhalte und Anhänge nicht berücksichtigt.

Zwei sehr einfache exemplarische Nachrichten sind:

(m_1) 2017-06-11: Bob \rightarrow Alice: *“I think I saw you in the red tent today.”*

(m_2) 2017-06-12: Alice \rightarrow Bob: *“Oh... I don’t remember seeing you yesterday.”*

4.2.2 CoreNLP Annotation

In Abschnitt 3.2 wurden die für diese Arbeit wesentlichen Annotatoren bereits vorgestellt: Tokenization, Lemmatization, POS-Tagging, NER, Coreference Resolution und Dependency Parsing. Um mit den Ergebnissen all dieser Annotatoren zu arbeiten, werden jene in einer gemeinsamen Datenstruktur zusammengefasst, dem sog. *Annotationsgraphen*. Hierbei handelt es sich um den Abhängigkeitsgraphen des CoreNLP *Enhanced++ Dependency Parsers*, in den die Ergebnisse der anderen Annotatoren eingefügt wurden:

1. **Lemmata:** Die Vollformen der Token werden durch ihre Lemmata ersetzt. Dieser Änderung liegt die Annahme zugrunde, dass die syntaktische Repräsentation einer Wortbeugung i. d. R. nicht relevant ist. Durch die Lemmatisierung wird es einfacher ähnliche Begriffe aufgrund ihrer textuellen Ähnlichkeit zu identifizieren.

2. **POS-Tags:** Durch das Verwerfen der Token-Vollformen können wertvolle grammatikalische Informationen verloren gehen. Daher wird jedem Token-Knoten sein POS-Tag hinzugefügt, welcher die Wortart und Beugung jedes lemmatisierten Tokens einheitlich repräsentiert.
3. **NER-Klassen:** Alle Token-Knoten, die der NER-Annotator klassifizieren konnte, erhalten ihre Klasse. Für alle von SUTime erkannten Zeitangaben, wird zudem ein neuer Knoten eingefügt. Alle Token, die an der Spezifikation einer Zeitangabe beteiligt sind (z. B. *yesterday* und *morning*) werden mittels einer Koreferenzkante mit dem Zeitknoten verknüpft.
4. **Koreferenzklassen:** Jede gefundene Token-Koreferenzklasse $C = \{t_1, \dots, t_n\}$ wird als eine Menge von Koreferenzkanten in den Annotationsgraphen eingefügt. Als Kantenmenge wird allerdings nicht $C \times C$ verwendet, da diese unnötig groß ist und zudem nicht alle verfügbaren Informationen repräsentiert. Die CoreNLP Coreference Resolution ordnet jeder Klasse nämlich eine sog. *Representative Mention* $rep(C) \in C$ zu. $rep(C)$ ist das Token einer Klasse, welches diese am besten repräsentiert.

Tom is calling his brother. $\Rightarrow C = \{\text{Tom, his}\}, rep(C) = \text{Tom}$

Um die *rep*-Information zu erhalten, werden nur Koreferenzkanten von den anderen Klassentoken $C \setminus \{rep(C)\}$ zu $rep(C)$ hinzugefügt.

Abbildung 4.4 zeigt den Annotationsgraphen der Beispielnachricht m_2 . In den Knoten sind die Token-Lemmata, die NER-Klasse und der POS-Tag dargestellt. Informationen, die nicht aus dem Abhängigkeitsgraphen stammen wurden farblich hervorgehoben.

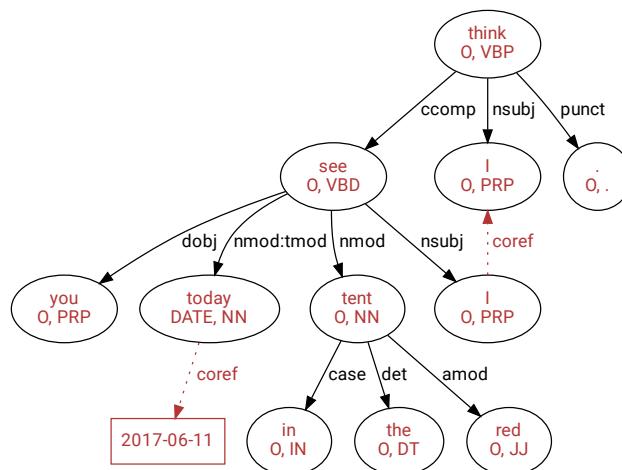


Abb. 4.4. Annotationsgraph der Nachricht “I think I saw you in the red tent today.” vom 2017-06-11.

4.2.3 Konzeptgraphtransformation

Aus dem Annotationsgraphen einer Nachricht muss im nächsten Schritt der sog. *Extraktionsgraph* erstellt werden. Hierbei handelt es sich um einen Konzeptgraphinstanzen der in 4.1 beschriebenen Ontologie, der den Inhalt der Nachricht beschreibt. Da natürliche Sprachen eine höhere Ausdrucksstärke als die Prädikatenlogik erster Ordnung und somit auch als Konzeptgraphen haben, ist es generell unmöglich jede beliebige Aussage im Extraktionsgraph zu repräsentieren. Das Ziel der Extraktionsgraphkonstruktion ist daher lediglich, einige wesentliche repräsentierbare semantische Strukturen zu identifizieren.

Hierfür wird eine deterministische regelbasierte Transformationspipeline benutzt. Diese basiert auf der Annahme, dass bestimmte Muster von POS-Tags und grammatikalischen Abhängigkeiten Rückschlüsse auf die semantischen Rollen in einer Aussage zulassen. Es wird also wenig bis gar kein Wissen über Wortbedeutungen o. ä. verwendet. Die Transformationspipeline, die im Rahmen dieser Arbeit entwickelt wurde, lässt sich in vier Phasen gliedern.

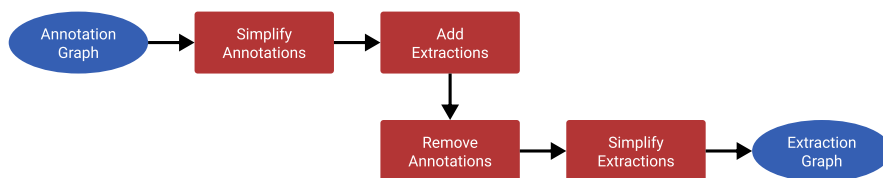


Abb. 4.5. Annotationsgraph → Extraktionsgraph Transformationspipeline

Die einzelnen Transformationsschritte in jeder Phase sind primär exemplarisch zu verstehen. Um die Extraktionsqualität zu verbessern, können und sollten weitere Schritte hinzugefügt werden.

1. Phase: Vereinfachen der Annotationen

In dieser Phase werden im ersten Schritt alle Daten aus dem Annotationsgraphen entfernt, die kein potentieller Bestandteil eines Konzeptes sind (z. B. Token für Satz- und Sonderzeichen, sowie Interjektionen, wie "Oh"). Die entfernten Token können zwar die Konnotation anderer Token beeinflussen, da deren Übersetzung in Konzeptgraphkonstrukte allerdings ein nicht triviales Problem ist, werden sie hier der Einfachheit halber ignoriert. Anschließend werden Token, die Bestandteile eines Kompositums sind, zu einem Knoten zusammengefasst. Die übrigbleibende Menge von Token-Knoten und Komposita bildet nun die Menge der Konzeptknoten des Wissensgraphen.

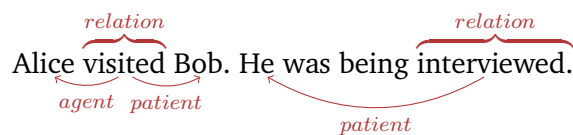
2. Phase: Einfügen der Extraktionen

Diese Phase ist für das Finden der Konzeptgraphstruktur verantwortlich. In ihr werden Konzeptgraph-Relationen zwischen den Konzept- bzw. Token-Knoten aus der ersten Phase und Kontexte in den Annotationsgraphen eingefügt.

Einfügen von *label*- und *named*-Prädikaten Das Einfügen der Bezeichner ist trivial. Als *label* wird für alle Konzepte ihr Lemma verwendet. Alle Konzepte mit einem POS-Tag, der einen Eigennamen impliziert (NNP und NNPS), erhalten das *named*-Attribut.

Einfügen von *relation*-, *agent*- und *patient*-Prädikaten Agens- und Patiens-Relationen bilden einen Großteil der semantischen Beziehungen im Wissensgraphen ab. Sie werden auf drei Wegen aus den Annotationen extrahiert.

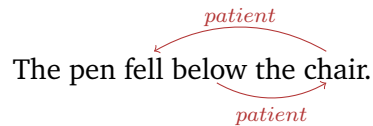
- **Prädikate:** Ein recht offensichtlicher Ansatz ist das Nutzen von Subjekt- und Objekt-Beziehungen. Direkte Objekte eines Prädikates sind i. d. R. auch dessen Patiens. Bei Subjekten ist die Diathese des Prädikates zu beachten. In Aktiv-Konstruktionen entspricht das Subjekt für gewöhnlich dem Agens. Bei passiven Prädikaten hingegen, ist das Subjekt der Patiens. Prädikate können dabei meist als *relation* aufgefasst werden.



Eine Ausnahme der vorigen Regeln sind Kopulae, da sie keine konkrete Eigenbedeutung haben und somit auch keinen Patiens haben können, den sie verändern. Es handelt sich stattdessen um Bindewörter, die ihrem Subjekt die Eigenschaften ihres Objekts zuschreiben. Da CoreNLP Kopulae erkennt, lässt sich diese Ausnahme leicht umsetzen. Im Englischen ist die Erkennung relativ trivial, da nur das Verb *to be* Kopula ist. Der Umgang mit Kopulae wird im Kontext der *inst*-Relation betrachtet.

Konkret werden also folgende Transformationen durchgeführt: $nsubj(x, y) \rightarrow agent(x, y)$, $nsubjpass(x, y) \vee dobj(x, y) \rightarrow patient(x, y)$ und $tag(x, VB^*) \rightarrow relation(x)$. Dabei darf x keine Kopula sein.

- **Präpositionen:** Neben Prädikaten werden Präpositionen häufig zum Verknüpfen von Konzepten benutzt. Diese haben im Gegensatz zu Prädikaten jedoch im Allgemeinen keinen Agens. Stattdessen lässt sich das auf eine Präposition p folgende Konzept x als Patiens von p auffassen. Wird nun eine solche Präpositionalkonstruktion in Beziehung zu einem anderen Konzept y gesetzt, wird y von x verändert, y ist dann also Patiens von x .



Konkret wird die folgende Transformation durchgeführt:

$$nmod(x, y) \vee nummod(x, y) \vee case(x, y) \rightarrow patient(y, x)$$

- **Adverbien:** $advmod(x, y) \rightarrow patient(y, x)$, da die Veränderung eines Verbs durch ein Adverb ebenfalls als eine Patiens-Beziehung aufgefasst werden kann.

Einfügen von *inst*-Prädikaten Für die Bestimmung von *inst*-Relationen werden drei Verfahren benutzt.

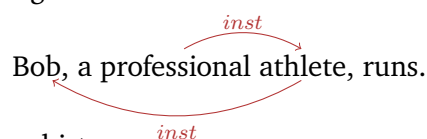
- **NER-Klassen:** Manche Konzepte besitzen NER-Klassen (z. B. Person, Ort, Organisation etc.), die bereits *inst*-Relationen abbilden. Es werden daher Konzeptknoten für alle vorkommenden Klassen hinzugefügt. Diese werden durch *inst*-Kanten mit ihren Instanzen verbunden. Die eingefügten Klassenknoten repräsentieren die abstrakten Konzepte der verschiedenen Klassen; sie werden also durch ihr *label* eindeutig identifiziert und erhalten daher das *named*-Attribut.
- **Kopulae:** Ein Prädikat, welches Kopula ist, wurde für das Identifizieren von *agent*- und *patient*-Relationen zuvor explizit ausgeschlossen. Dies liegt daran, dass Kopulae ihre Subjekte und Objekte i. d. R. in ein *inst*-ähnliches Verhältnis setzen.



Konkret wird folgende Transformation durchgeführt:

$$(\exists z : cop(x, z)) \wedge nsubj(x, y) \rightarrow inst(x, y)$$

- **Attribute und Appositionen:** Ähnlich wie Kopulae drücken diese zwei grammatikalischen Beziehungen ein *inst*-ähnliches Verhältnis aus.



Die Transformationsregel ist:

$$amod(x, y) \vee appos(x, y) \rightarrow inst(y, x)$$

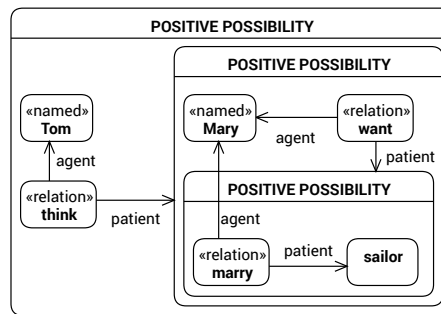


Abb. 4.6. Clause-basierte Kontextschachtelung der Aussage “Tom thinks that Mary wants to marry a sailor.”

Einfügen von Kontexten Nachdem nun beschrieben wurde, wie die Konzeptgraph-Relationen ermittelt werden, muss noch geklärt werden, wie die Kontexthierarchie aufgebaut wird. Das gewählte Verfahren lässt sich in zwei Schritte gliedern. Im ersten Schritt werden ausschließlich positive Möglichkeitskontexte eingefügt. Im zweiten Schritt werden dann die negativen Aktualkontexte hinzugefügt.

1. **Möglichkeitskontexte:** Um Möglichkeitskontexte einfügen zu können, muss zuerst definiert werden, was als Möglichkeit verstanden wird. Sowas Ansatz hierfür ist es, Teilsätze (im Englischen *clause* genannt) als solche zu interpretieren, da sie jeweils genau einen Sachverhalt ausdrücken, der möglicherweise wahr ist. Teilsätze können prinzipiell beliebig geschachtelt werden und hängen i. d. R. von einem einleitenden Konzept ab; sie lassen sich also gut durch entsprechend geschachtelte Möglichkeitskontexte repräsentieren. In dieser Arbeit werden die Möglichkeitskontexte nach dem selben Prinzip eingefügt. Teilsätze lassen sich im Annotationsgraphen über die Abhängigkeiten *csubj*, *ccomp*, *xcomp* und *advcl* leicht identifizieren. Da der Abhängigkeitsgraph ein DAG ist, können Konzepte in mehreren Teilsätzen bzw. Kontexten auftauchen; in diesem Fall muss das Konzept in den kleinsten gemeinsamen Elternkontext (gemäß \leq -Relation) eingefügt werden.

Da die Gesamtheit einer Nachricht m nicht unbedingt wahr sein muss, wird zudem ein Wurzel-Möglichkeitskontext eingefügt, der alle Konzepte, die innerhalb von m erwähnt wurden, enthält. Konzeptknoten, die für NER-Klassen eingefügt wurden, fallen nicht darunter, da die Existenz der NER-Klassen, unabhängig vom Inhalt einer Nachricht, als wahr vorausgesetzt wird.

2. **Negative Aktualkontexte:** Das Formalisieren von Negation in natürlichen Sprachen ist nicht immer eindeutig möglich. Dies liegt daran, dass die Bedeutung einer Negation variieren kann, so ist doppelte Verneinung im Englischen z. B. manchmal ein umgangssprachliches stilistisches Mittel zur Betonung der Negation (z. B. “We don’t have no time.”). Da eine umfassende Behandlung aller möglichen Interpretationen der Negation den Rahmen dieser Arbeit gesprengt

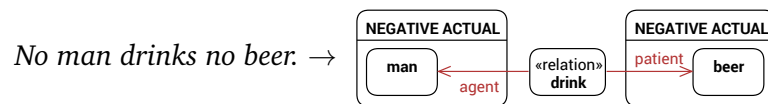
hätte, wird stattdessen ein stark vereinfachtes Modell verwendet, welches für viele, jedoch nicht für alle, Aussagen korrekte Ergebnisse liefert.

Natürlichsprachliche Negation wird im Folgenden immer als äquivalent zur logischen Negation verstanden. Um diese Negationen durch Kontexte zu repräsentieren, muss definiert werden, welche Konzepte durch das Vorkommen einer natürlichsprachlichen Negation betroffen sind. Als allgemeine Regel wird benutzt, dass für *agent* und *patient*-Beziehungen von x nach y die Eigenschaft $x \leq y$ (siehe 3.1.2) gilt; d. h. die Nicht-Existenz des Agens oder Patiens eines Prädikates, impliziert dessen Nicht-Existenz.

$$\text{No man helped.} \Rightarrow \begin{cases} \exists \text{helped} \neg \exists \text{man} : \text{agent}(\text{helped}, \text{man}) & \text{man betont} \\ \neg \exists \text{man} \exists \text{helped} : \text{agent}(\text{helped}, \text{man}) & \text{helped betont } \checkmark \end{cases}$$

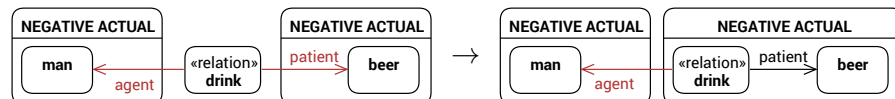
Wie das obige Beispiel zeigt, ist diese Regel nicht unbedingt korrekt. Die Interpretation gemäß der Regel ist allerdings meist zutreffend, da alternative Interpretationen im Englischen i. d. R. nur durch Betonung bestimmter Wörter ausgedrückt werden können. Das Einfügen negativer Kontexte läuft gemäß der vorigen Überlegungen wie folgt ab:

- 2.1 Konzepte, die eine ungerade Anzahl von *neg*-Beziehungen zu Negations-Token haben, werden von einem negativen Aktualkontext umschlossen.

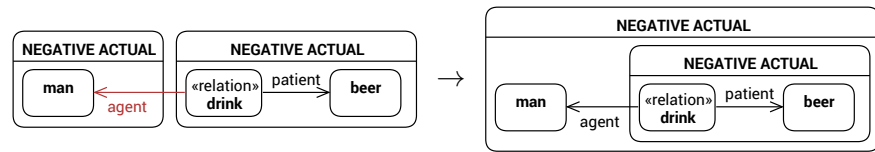


- 2.2 Es kann nun *agent*- oder *patient*-Kanten geben, die in einen der hinzugefügten Negationskontexte hineinlaufen, was gemäß der vorigen Überlegungen **unzulässig** ist. Für jede dieser Kanten von x nach y werden nun Korrekturen vorgenommen, sodass hinterher $x \leq y$ gilt und somit keine Regel-Verletzungen mehr vorliegen. Es werden dabei zuerst *patient*-Kanten, dann *agent*-Kanten, korrigiert.

- 2.2.1 Falls vor der Korrektur $y \leq x$ gilt, wird x in den Kontext von y bewegt.



2.2.2 Andernfalls wird der größte Elternkontext von x , der nicht gemeinsamer Elternkontext von x und y ist, in den Kontext von y verschoben.



3. Phase: Entfernen der Annotationen

Dies ist die trivialste der vier Phasen. Sie wird hier lediglich der Vollständigkeit halber erwähnt. Alle Abhängigkeitsgraph-Kanten, sowie POS-Tags und NER-Klassen-Tags werden entfernt, *coref*-Kanten bleiben erhalten. Das Ergebnis ist nun bereits beinahe ein Konzeptgraph.

4. Phase: Vereinfachen der Extraktionen

Koreferenzkanten Nach der dritten Phase liegt zwar prinzipiell eine Konzeptgraphstruktur vor, es kann allerdings noch Verletzungen der Eigenschaft dominierender Knoten geben. Konkret liegt dies daran, dass die in 4.2.2 eingefügten Koreferenzkanten nach Einfügen der Kontexte in benachbarten Kontexten liegen können. Dies passiert z. B. dann, wenn in zwei Nebensätzen von der selben Person gesprochen wird, diese jedoch im Hauptsatz nicht auftaucht. Die Existenz einer derartigen Koreferenz impliziert, dass das koreferente Konzept auch im Kontext des Hauptsatzes existent sein muss, obwohl es nicht explizit erwähnt wird.

Um dominierende Knoten zu erhalten, können die koreferenten Konzepte zu einem Konzeptknoten zusammengefasst werden. Dieser neue Knoten übernimmt das *label* der *Representative Mention* (siehe 4.2.2) und wird in den kleinsten gemeinsamen Elternkontext aller zusammengefassten Knoten (gemäß der \leq -Relation) eingefügt. Dabei geht zwar die Information verloren, wo und mit welchen Pronomina das Konzept innerhalb der Nachricht referenziert wurde, dies ist allerdings i. d. R. irrelevant für die weitere Verarbeitung. Nach diesem Schritt liegt ein Konzeptgraph vor.

Kontexte Um einen möglichst kompakten Konzeptgraphen zu erhalten, werden im letzten Schritt geschachtelte Kontexte vereinfacht. Dabei werden $2n$ -fach geschachtelte Ketten negativer Aktualkontexte entfernt. Zudem werden positive und negative Möglichkeitskontexte, die lediglich einen negativen Aktualkontext enthalten, zu negativen bzw. positiven Möglichkeitskontexten. Diese Vereinfachungen werden so lange greedy durchgeführt, bis keine weiteren mehr möglich sind. Das Ergebnis ist der Extraktionsgraph.

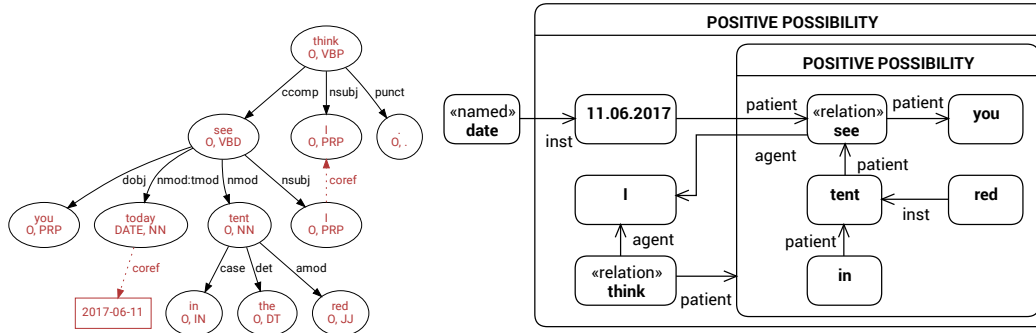


Abb. 4.7. Annotationsgraph und daraus konstruierter Extraktionsgraph der Nachricht “I think I saw you in the red tent today.” vom 2017-06-11.

4.3 Wissensgraphkonstruktionsphase

Nachdem nun geklärt ist, wie die Konstruktion des Extraktionsgraphen einer Nachricht abläuft, wird im Folgenden beschrieben, wie dieser Extraktionsgraph mittels PSL in einen bestehenden Wissensgraphen eingefügt wird.

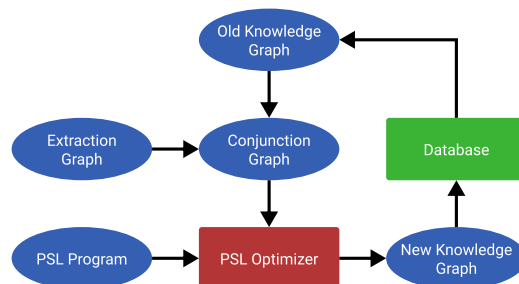


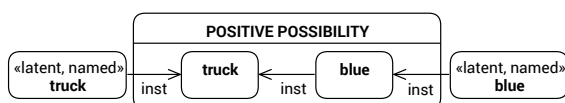
Abb. 4.8. Wissensgraphkonstruktionsphase der Konstruktionspipeline

4.3.1 Konstruktion des Konjunktionsgraphen

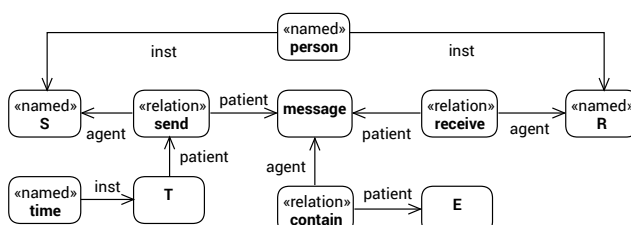
Der erste Schritt der Wissensgraphkonstruktion ist das Bilden des sog. Konjunktionsgraphen. Der Konjunktionsgraph ist ein Konzeptgraph, der alle Informationen der einzufügenden Nachricht und des bestehenden Wissensgraphen enthält, allerdings noch keine Schlussfolgerungen aus den hinzugekommenen Informationen beinhaltet. Der Konjunktionsgraph wird in vier Schritten konstruiert.

1. **Latente Konzepte:** Da mit PSL keine neuen Konzepte, sondern lediglich Relationen zwischen gegebenen Konzepten inferiert werden können, müssen alle potentiell möglichen Konzepte bereits im Konjunktionsgraph vorhanden sein. Alle Konzepte des Extraktionsgraphen, die sich innerhalb des Wurzel-Möglichkeitskontextes befinden, werden daher dupliziert und als globale latente Konzepte in das Sheet of Assertion eingefügt. Zwischen den latenten Konzepten und den zugehörigen Konzepten werden zudem *inst*-Relationen eingefügt. Während der PSL-Inferenz wird dann bestimmt, ob ein latentes

Konzept auch außerhalb des Möglichkeitskontextes aus dem es stammt existent ist und somit zu einem Konzept wird. Da latente Konzepte das durch ihr *label* repräsentierte Konzept repräsentieren, erhalten sie das *named*-Attribut.



2. **Nachrichtenmetadaten:** Damit der Konjunktionsgraph alle Informationen der einzufügenden Nachricht enthält, müssen neben dem Extraktionsgraph, der lediglich den Inhalt der Nachricht beschreibt, auch der Sender *S*, der Empfänger *R* und die Sendezeit *T* eingefügt werden. Dies geschieht durch folgende Erweiterung des Extraktionsgraphen *E*:



Die *patient*(contain, *E*)-Kante hat dabei den Wurzel-Möglichkeitskontext von *E* als Ziel.

3. **Konjunktion:** Der um Metadaten erweiterte Extraktionsgraph kann nun mit dem Wissensgraphen kombiniert werden. Hierfür werden lediglich die Knoten- und Kantenmengen vereinigt. Dies entspricht der logischen Konjunktion der durch beide Graphen repräsentierten Ausdrücke.
4. **Koreferenzen:** Im Konjunktionsgraph befinden sich i. d. R. bekannte koreferente Konzepte. Um den Graphen zu vereinfachen und die Inferenz zu vereinfachen, sollten diese Konzepte zusammengefasst werden. Konkret handelt es sich hierbei um die NER-Klassen-Konzeptknoten (Person, Ort, Datum etc.), die latenten Konzepte und um die Knoten, die für *S* und *R* eingefügt wurden. Diese drei Arten von Knoten können bereits durch vorherige Nachrichten Teil des Wissensgraphen geworden sein und müssen beim Hinzukommen einer neuen Nachricht nicht erneut eingefügt werden.

4.3.2 Relationsinferenz mit PSL

Der Konjunktionsgraph wird nun als Eingabe für ein PSL-Programm verwendet, welches neue Relationen zwischen den Konzepten inferieren soll. In diesem Abschnitt wird beschrieben, wie ein solches PSL-Programm aussehen kann.

PSL-Prädikate Im ersten Schritt müssen die verwendeten PSL-Prädikate definiert werden. Hier bietet es sich an, die sechs Konzeptgraph-Relationen zu übernehmen. Darüber hinaus müssen allerdings noch Prädikate für die Modellierung der Kontextstruktur definiert werden.

1. *concept* : **UUID**: Markiert nicht latente Konzeptknoten.
2. *concept_{latent}* : **UUID**: Markiert latente Konzeptknoten.
3. *label* : **UUID** \times **String**: Aus der Ontologie übernommen.
4. *named* : **UUID**: Aus der Ontologie übernommen.
5. *inst* : **UUID** \times **UUID**: Aus der Ontologie übernommen.
6. *relation* : **UUID**: Aus der Ontologie übernommen.
7. *agent* : **UUID** \times **UUID**: Aus der Ontologie übernommen.
8. *patient* : **UUID** \times **UUID**: Aus der Ontologie übernommen.
9. *similar* : **String** \times **String**: Die Jaro-Ähnlichkeit der gegebenen Strings. Falls zwei Strings A, B eine Distanz < 0.75 haben, gilt jedoch $similar(A, B) = 0$.
10. *context* : **UUID**: Markiert Kontexte.
11. *possibility* : **UUID**: Markiert Möglichkeitskontexte.
12. *negative* : **UUID**: Markiert negative Kontexte.
13. *nest* : **UUID** \times **UUID**: Ordnet Kontexten ihre direkten Kindknoten zu.
14. *nest_{indirect}* : **UUID** \times **UUID**: Ordnet jedem Kontext alle Knoten zu, die von seiner Kontextbox umschlossen sind.

Bei *similar* handelt es sich um ein sog. funktionales Prädikat. Funktionale Prädikate werden intensional durch eine Funktion beschrieben und dürfen weder in der offenen Atommengen \mathbb{O} , noch in der geschlossenen Atommengen \mathbb{C} vorkommen, da weder das Definieren, noch das Inferieren, von Grundatomen eines solchen Prädikates sinnvoll ist. *similar* wird verwendet, um die *label*-Ähnlichkeit von *named*-Konzepten bei der Inferenz berücksichtigen zu können. Es wurde das Jaro-Ähnlichkeitsmaß gewählt, da es für Namen gute Ergebnisse liefert und sich zudem effizient berechnen lässt [Chr06], was aufgrund der zahlreichen Vergleiche, die im Rahmen einer Inferenz durchgeführt werden, wichtig ist. Ähnlichkeitswerte < 0.75 werden auf 0 abgerundet, damit unähnliche *label* bei der Inferenz gar nicht erst betrachtet werden. Der Schwellwert 0.75 wurde manuell gewählt (siehe 5).

Allgemeine PSL-Regeln Basierend auf der Semantik der PSL-Prädikate, lassen sich allgemeine PSL-Regeln definieren, die unabhängig von der Bedeutung einzelner Konzepte wahr sind. Ein Teil dieser Regeln modelliert negative Priors und Transitivität. Der Rest modelliert verschiedene Annahmen über die Struktur der Prädikate. Konkret wird folgende Regelmenge verwendet:

- **Regeln zur *inst*-Inferenz:**

$$\begin{aligned}
 0.2 : & \text{label}(A, L_A) \wedge \text{label}(B, L_B) \wedge \text{similar}(L_A, L_B) & (r_1) \\
 & \wedge \text{nest}(C_A, A) \wedge \text{nest}_{\text{indirect}}(C_A, B) \wedge A \neq B \\
 & \wedge \text{named}(A) \wedge \text{named}(B) \rightarrow \text{inst}(A, B)
 \end{aligned}$$

Die Regel r_1 modelliert die Annahme, dass zwischen zwei *named*-Konzepten A, B mit ähnlichen Labeln u. U. eine *inst*(A, B)-Beziehung besteht; Voraussetzung dafür ist, dass $B \leq A$ gilt.

$$0.1 : \neg \text{inst}(A, B) \quad (r_2)$$

$$\infty : \text{inst}(A, B) \wedge \text{inst}(B, C) \rightarrow \text{inst}(A, C) \quad (r_3)$$

$$\infty : \text{inst}(A, B) \rightarrow \neg \text{inst}(B, A) \quad (r_4)$$

r_2 beschreibt, dass *inst* im Allgemeinen nicht gilt. Die Constraints r_3 und r_4 bilden die Transitivität und Asymmetrie von *inst* ab.

- **Regeln zur *concept*-Inferenz:**

$$0.05 : \text{concept}_{\text{latent}}(A) \wedge \text{inst}(A, B) \rightarrow \text{concept}(A) \quad (r_5)$$

$$0.5 : \text{concept}(A) \wedge \text{concept}_{\text{latent}}(B) \wedge \text{inst}(A, B) \rightarrow \text{concept}(B) \quad (r_6)$$

$$0.1 : \neg \text{concept}(A) \quad (r_7)$$

r_5 modelliert die Annahme, dass ein latentes Konzept mit mehreren Instanzen vermutlich ein tatsächliches Konzept ist. r_6 beschreibt umgekehrt, dass ein latentes Konzept, das Instanz eines tatsächlichen Konzeptes ist, vermutlich ebenfalls ein tatsächliches Konzept ist.

- **Regeln zur *nest_{indirect}*-Inferenz:**

$$\infty : \text{nest}(A, B) \rightarrow \text{nest}_{\text{indirect}}(A, B) \quad (r_8)$$

$$\infty : \text{nest}_{\text{indirect}}(A, B) \wedge \text{nest}(B, C) \rightarrow \text{nest}_{\text{indirect}}(A, C) \quad (r_9)$$

Da nur *nest*-Relationen und keine *nest_{indirect}*-Relationen in der Eingabe vorhanden sind, müssen letztere mit r_8 und r_9 inferiert werden.

- **Geschlossene Prädikate:**

$$\infty : \neg \text{named}(A) \quad (r_{10})$$

$$\infty : \neg \text{relation}(A) \quad (r_{11})$$

$$\infty : \neg \text{nest}(A) \quad (r_{12})$$

Der Zweck dieser Priors ist es, zu verhindern, dass zusätzlich zu den *named*-, *relation*- und *nest*-Atomen des Konjunktionsgraphen weitere derartige Atome inferiert werden. Da bei der PSL-Inferenz keine NLP-Informationen zur Verfügung stehen, können diese Prädikate nämlich im Allgemeinen nicht sinnvoll inferiert werden und werden daher von der Inferenz ausgeschlossen.

Domänenspezifische PSL-Regeln Die allgemeinen Regeln sind nicht ausreichend, um aus einem gegebenen Konjunktionsgraphen relevante Erkenntnisse zu ziehen. Hierfür ist domänenspezifisches Wissen und eine zugehörige Regelmengung notwendig. Um das Grundprinzip zu veranschaulichen wird in dieser Arbeit eine sehr einfache domänenspezifische Regelmengung verwendet. Abhängig vom betrachteten Datensatz, sind die Inferenz mit realen Kommunikationsdaten i. d. R. deutlich umfangreichere Regelmengungen notwendig.

Die verwendeten domänenspezifischen Regeln führen zusätzlich zu den 14 allgemeinen PSL-Prädikaten weitere Prädikate ein, die zur Repräsentation von Zwischenergebnissen benutzt werden. Diese zusätzlichen Prädikate sind nicht unbedingt notwendig, sondern vereinfachen lediglich die Regeldefinition. Die folgenden domänenspezifischen Regeln werden verwendet:

$$1 : \text{context}(ctx) \wedge \text{patient}(\text{contain}, ctx) \wedge \text{agent}(\text{contain}, msg) \quad (r_{13})$$

$$\wedge \text{inst}(\text{"concept_contain"}, \text{contain}) \rightarrow \text{message}(ctx, msg)$$

$$1 : \text{message}(ctx, msg) \wedge \text{patient}(\text{send}, msg) \wedge \text{agent}(\text{send}, sender) \quad (r_{14})$$

$$\wedge \text{inst}(\text{"concept_send"}, \text{send}) \rightarrow \text{sender}(ctx, sender)$$

$$1 : \text{message}(ctx, msg) \wedge \text{patient}(\text{receive}, msg) \wedge \text{agent}(\text{receive}, receiver) \quad (r_{15})$$

$$\wedge \text{inst}(\text{"concept_receive"}, \text{receive}) \rightarrow \text{receiver}(ctx, receiver)$$

$$1 : \text{nest}_{\text{indirect}}(ctx, I) \wedge \text{inst}(\text{"concept_I"}, I) \quad (r_{16})$$

$$\wedge \text{sender}(ctx, sender) \rightarrow \text{inst}(sender, I)$$

$$1 : \text{nest}_{\text{indirect}}(ctx, you) \wedge \text{inst}(\text{"concept_you"}, you) \quad (r_{17})$$

$$\wedge \text{receiver}(ctx, receiver) \rightarrow \text{inst}(receiver, you)$$

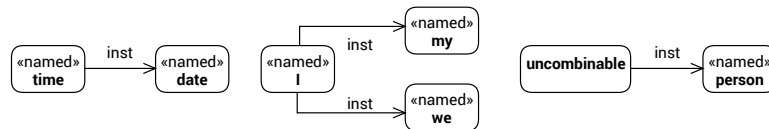
$$1 : \text{inst}(\text{"concept_uncombinable"}, x) \wedge \text{inst}(x, p_1) \wedge \text{inst}(x, p_2) \quad (r_{18})$$

$$\wedge \text{inst}(p_1, y) \wedge \text{inst}(p_2, y) \wedge p_1 \neq p_2 \wedge \neg \text{inst}(p_2, p_1) \rightarrow \text{inst}(p_1, p_2)$$

r_{13} , r_{14} und r_{15} ordnen Nachrichten ihre Sender und Empfänger zu, was aufgrund der einheitlichen Repräsentation der Nachrichtenmetadaten im Konjunktionsgraphen eindeutig möglich ist. r_{16} und r_{17} beschreiben, dass die Konzepte I und you innerhalb einer Nachricht auf den Sender bzw. den Empfänger dieser Nachricht

verweisen. Interessanter ist die Regel r_{18} . Falls ein Konzept y Instanz zweier Konzepte $p_1 \neq p_2$ ist, die wiederum Instanz eines *uncombinable* Konzeptes x sind, gilt entweder $inst(p_1, p_2)$ oder $inst(p_2, p_1)$. Der Zweck dieser Regel wird im Kontext des im Folgenden beschriebenen domänenspezifischen Vorwissens deutlich.

Domänenspezifisches Vorwissen Damit die domänenspezifischen Regeln die gewünschten Ergebnisse liefern, müssen sie um domänenspezifisches Wissen ergänzt werden. Dieses Wissen ist im Wesentlichen der Initialzustand des Wissensgraphen, bevor Nachrichten eingefügt wurden. In dieser Arbeit wird der folgende initiale Wissensgraph verwendet:



Für die Analyse von Nachrichten, die viel domänenspezifisches Wissen enthalten, ist offensichtlich eine wesentlich umfangreichere Wissensbasis erforderlich. Hierfür könnten frei verfügbare semantische Netze, wie z. B. WordNet [Wor] oder NELL [Car+10], verwendet werden.

Um das Grundprinzip des Konstruktionsverfahren zu illustrieren, ist ein kleiner initialer Wissensgraph allerdings vollkommen ausreichend. Es sind im Wesentlichen drei Tatsachen kodiert: Ein Datum ist eine spezielle Form der Zeitangabe, die Konzepte *my* und *we* schließen u. a. das Konzept *I* ein und Personen sind *uncombinable*. Letztere Relation wird verwendet, damit r_{18} für Personen gilt. Falls also ein Konzept Instanz zweier Personen ist, muss eine dieser beiden Personen eine Instanz der anderen Person sein. Anders formuliert bedeutet dies, dass ein Konzept nicht zugleich zwei verschiedene Personen $p_1 \neq p_2$ repräsentieren kann. Die Eigenschaften $prop(p_1), prop(p_2)$ sind also nicht frei kombinierbar.

4.4 Implementation

Das in den vorherigen Abschnitten beschriebene Verfahren wurde im Rahmen dieser Arbeit prototypisch implementiert. Bei der Wahl der hierfür verwandten Technologien wurde darauf Wert gelegt, dass eine möglichst nahtlose Integration der Komponenten möglich ist. Sowohl CoreNLP [Cor] als auch die PSL-Referenzimplementation [Psl] sind JVM-Bibliotheken. Diese Arbeit wurde ebenfalls für die JVM implementiert, insbesondere weil es zu der JVM-basierten PSL-Referenzimplementation keine Alternative gibt und eine Reimplementation von PSL den Rahmen dieser Arbeit gesprengt hätte.

Als JVM-Sprache wurde Clojure gewählt, ein moderner Lisp-1-Dialekt. Andere JVM-Sprachen, wie z. B. Java, Scala oder Groovy, wurden ausgeschlossen, da sie sich während des Entwicklungsprozesses als hinderlich erwiesen haben. Der Hauptgrund hierfür ist, dass CoreNLP bei der Initialisierung der Verarbeitungs-Pipeline diverse Modelle laden muss. Abhängig von der benutzten Hardware dauert dies zwischen ca. 20 Sekunden und mehreren Minuten; diese Wartezeiten sind ein stark verlangsamernder Faktor, wenn sie nach jeder Code-Änderung auftreten. Da Clojure ein Lisp ist, unterstützt es traditionsgemäß *REPL Driven Development*. Statt nach jeder Änderung die Anwendung neu zu starten und die Modelle erneut zu laden, kann so lediglich der geänderte Bytecode in den laufenden Prozess injiziert werden; die geladenen Modelle bleiben dabei im Speicher und die Änderung kann ohne weitere Wartezeit getestet werden. Durch die Wahl von Clojure konnte die Entwicklung deutlich beschleunigt werden.

Alle im folgenden Kapitel vorgestellten Tests wurden mit der entwickelten Referenzimplementation durchgeführt. Die Rechte am Quelltext liegen bei der Firma Atos, er kann daher nicht frei verfügbar gemacht werden.

Bewertung

Hallo Welt.

```

1  call algo.unionFind.stream(
2      "match (:concept {id: 'concept_person'})-[:inst]->(n) " +
3      "optional match (x:concept)-[:inst]->(n) " +
4      "where x.id in ['concept_I', 'concept_you'] " +
5      "with n, count(x) as cx where cx = 0 " +
6      "return id(n) as id",
7      "match (n1)-[:inst]->(n2) return id(n1) as source, id(n2) as target",
8      {graph: "cypher"})
9  yield nodeId, setId
10 match (node) where id(node) = nodeId
11 with setId, collect(distinct node.label) as personNames
12 return personNames

1  match (a:concept)-[:inst*0..1]->()-[:agent]-(r:relation),
2      (r)-[:patient]->()-[:inst*0..1]-(p:concept)
3  where a.label = "Duane Seppi"
4  with r, collect(distinct p.label) as patients
5  optional match (:concept {id:"concept_time"})-[:inst]->(t:concept),
6      (t)-[:inst*0..1]->()-[:patient]->(r)
7  where not exists(t.named) and t.id starts with "concept_"
8  unwind patients as patient
9  return r.label as relation, patient, t.label as time

1  match (sender:concept {label: "John Doe"})-[:inst*0..1]->()-[:agent]-(send),
2      (send:relation)-[:patient]->(msg:concept),
3      (msg)-[:agent]-(contains:relation)-[:patient]->(ctx:context),
4      nestPath = (ctx)-[:nest*]->(r:relation),
5      (a:concept)-[:inst*0..1]->()-[:agent]-(r),
6      (r)-[:patient]->()-[:inst*0..1]-(p:concept)
7  with collect(distinct a.label) as agents,
8      collect(distinct p.label) as patients,
9      r, nestPath
10 unwind nodes(nestPath) as ctx
11 optional match (ctx)-[:nest]->(child)
12 with ctx, count(ctx) as cnt, agents, r, patients
13 with filter(c in collect({neg: ctx.negative, cnt: cnt}) where c.neg) as negs,
14      agents, r, patients
15 with case
16     when size(negs) > 0 then case
17         when any(c in negs where c.cnt > 1) then "CONDITIONAL"
18         else "NEGATIVE"
19     end

```

```
20     else "POSITIVE"
21 end as truth, agents, r, patients
22 return agents, r.label as relation, patients, truth
```

5.1 Testmethode

5.2 Ergebnisse

Anhang

A

Literatur

- [Bac+13] Stephen H. Bach, Bert Huang, Ben London und Lise Getoor. „Hinge-loss Markov Random Fields: Convex Inference for Structured Prediction“. In: Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence. UAI'13. Bellevue, WA: AUAI Press, 2013, S. 32–41 (zitiert auf Seite 15).
- [Bac+15] Stephen H. Bach, Matthias Broecheler, Bert Huang und Lise Getoor. „Hinge-Loss Markov Random Fields and Probabilistic Soft Logic“. In: (2015). arXiv: 1505.04406v2 [cs.LG] (zitiert auf Seite 15).
- [Boy+11] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato und Jonathan Eckstein. „Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers“. In: Foundations and Trends in Machine Learning 3.1 (Jan. 2011), S. 1–122 (zitiert auf den Seiten 15, 31).
- [Bra83] Brachman. „What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks“. In: Computer 16.10 (1983), S. 30–36 (zitiert auf Seite 10).
- [Br10] Matthias Bröcheler, Lilyana Mihalkova und Lise Getoor. „Probabilistic Similarity Logic“. In: Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence. UAI'10. Catalina Island, CA: AUAI Press, 2010, S. 73–82 (zitiert auf Seite 15).
- [Car+10] Andrew Carlson, Justin Betteridge, Bryan Kisiel et al. „Toward an Architecture for Never-ending Language Learning“. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence. AAAI'10. Atlanta, Georgia: AAAI Press, 2010, S. 1306–1313 (zitiert auf den Seiten 11, 53).
- [Chr06] Peter Christen. „A comparison of personal name matching: Techniques and practical issues“. In: Data Mining Workshops, 2006. ICDM Workshops 2006. Sixth IEEE International Conference on. IEEE. 2006, S. 290–294 (zitiert auf Seite 50).
- [CM12] Angel X. Chang und Christopher D. Manning. „Sutime: A library for recognizing and normalizing time expressions.“ In: LREC. Bd. 2012. 2012, S. 3735–3740 (zitiert auf Seite 21).
- [Dau03] Frithjof Dau. The Logic System of Concept Graphs with Negation. Springer Berlin Heidelberg, 2003 (zitiert auf den Seiten 17, 20).
- [Dow91] David Dowty. „Thematic Proto-Roles and Argument Selection“. In: Language 67.3 (1991), S. 547 (zitiert auf Seite 36).

- [Fin+05] Jenny Rose Finkel, Trond Grenager und Christopher Manning. „Incorporating non-local information into information extraction systems by gibbs sampling“. In: Proceedings of the 43rd annual meeting on association for computational linguistics. Association for Computational Linguistics. 2005, S. 363–370 (zitiert auf Seite 21).
- [Fre79] Gottlob Frege. Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. L. Nebert, 1879 (zitiert auf Seite 7).
- [Har+07] Frank van Harmelen, Vladimir Lifschitz und Bruce Porter. Handbook of Knowledge Representation. San Diego, USA: Elsevier Science, 2007, S. 213–237 (zitiert auf Seite 10).
- [HR+83] Frederick Hayes-Roth, Donald A. Waterman und Douglas B. Lenat. Building Expert Systems. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1983, S. 6–7 (zitiert auf Seite 9).
- [Lao+11] Ni Lao, Tom Mitchell und William W. Cohen. „Random Walk Inference and Learning in a Large Scale Knowledge Base“. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. EMNLP ’11. Edinburgh, United Kingdom: Association for Computational Linguistics, 2011, S. 529–539 (zitiert auf Seite 14).
- [Leh92] F. Lehmann. Semantic Networks in Artificial Intelligence. New York, NY, USA: Elsevier Science Inc., 1992, S. 6 (zitiert auf Seite 10).
- [Man+14] Christopher D. Manning, Mihai Surdeanu, John Bauer et al. „The Stanford CoreNLP Natural Language Processing Toolkit“. In: Association for Computational Linguistics (ACL) System Demonstrations. 2014, S. 55–60 (zitiert auf Seite 13).
- [MS03] Catherine C. Marshall und Frank M. Shipman. „Which Semantic Web?“ In: Proceedings of the Fourteenth ACM Conference on Hypertext and Hypermedia. HYPERTEXT ’03. Nottingham, UK: ACM, 2003, S. 57–66 (zitiert auf Seite 11).
- [New+59] Allen Newell, John C. Shaw und Herbert A. Simon. „Report on a general problem solving program“. In: IFIP congress. Bd. 256. Pittsburgh, PA. 1959, S. 64 (zitiert auf Seite 9).
- [Nic+14] Maximilian Nickel, Xueyan Jiang und Volker Tresp. „Reducing the Rank in Relational Factorization Models by Including Observable Patterns“. In: Advances in Neural Information Processing Systems 27. Hrsg. von Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence und K. Q. Weinberger. Curran Associates, Inc., 2014, S. 1179–1187 (zitiert auf Seite 14).
- [Nic+16] Maximilian Nickel, Kevin Murphy, Volker Tresp und Evgeniy Gabrilovich. „A Review of Relational Machine Learning for Knowledge Graphs“. In: Bd. 104. 1. Institute of Electrical und Electronics Engineers (IEEE), Jan. 2016, S. 11–33 (zitiert auf Seite 14).
- [Nic13] Maximilian Nickel. „Tensor Factorization for Relational Learning“. Diss. Ludwig-Maximilians-Universität München, 2013 (zitiert auf Seite 14).
- [Niv04] Joakim Nivre. „Incrementality in deterministic dependency parsing“. In: Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together. Association for Computational Linguistics. 2004, S. 50–57 (zitiert auf Seite 22).

- [NS56] Allen Newell und Herbert Simon. „The logic theory machine—A complex information processing system“. In: IRE Transactions on information theory 2.3 (1956), S. 61–79 (zitiert auf Seite 9).
- [Pei85] C. S. Peirce. „On the Algebra of Logic: A Contribution to the Philosophy of Notation“. In: American Journal of Mathematics 7.2 (Jan. 1885), S. 180 (zitiert auf Seite 8).
- [Puj+15] Jay Pujara, Ben London und Lise Getoor. „Budgeted Online Collective Inference“. In: Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence. UAI’15. Amsterdam, Netherlands: AUAI Press, 2015, S. 712–721 (zitiert auf den Seiten 15, 32).
- [RM92] R. P. Van de Riet und R. A. Meersman. Linguistic Instruments in Knowledge Engineering. New York, NY, USA: Elsevier Science Inc., 1992 (zitiert auf Seite 10).
- [San90] Beatrice Santorini. „Part-of-speech tagging guidelines for the Penn Treebank Project (3rd revision)“. In: (1990) (zitiert auf Seite 21).
- [Slu87] Hans Sluga. „Frege against the Booleans.“ In: Notre Dame Journal of Formal Logic 28.1 (Jan. 1987), S. 80–98 (zitiert auf Seite 8).
- [Sow11] John F. Sowa. „Peirce's tutorial on existential graphs“. In: Semiotica 2011.186 (Jan. 2011) (zitiert auf Seite 8).
- [Sow76] John F. Sowa. „Conceptual Graphs for a Data Base Interface“. In: IBM Journal of Research and Development 20.4 (Juli 1976), S. 336–357 (zitiert auf Seite 10).
- [Tou+03] Kristina Toutanova, Dan Klein, Christopher D Manning und Yoram Singer. „Feature-rich part-of-speech tagging with a cyclic dependency network“. In: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1. Association for Computational Linguistics. 2003, S. 173–180 (zitiert auf Seite 21).
- [WR10] Alfred North Whitehead und Bertrand Russell. Principia mathematica. 1910 (zitiert auf Seite 9).

Webseiten

- [BH] Dan Brickle und Ivan Herman. Semantic Web Interest Group. W3C. URL: <https://www.w3.org/2001/sw/interest/> (besucht am 10. Sep. 2017) (zitiert auf Seite 11).
- [Cor] Stanford CoreNLP. Stanford University. URL: <https://stanfordnlp.github.io/CoreNLP> (besucht am 7. Sep. 2017) (zitiert auf den Seiten 13, 20, 53).
- [Goo] Google Trends "knowledge graph". Google. 2017. URL: <https://trends.google.com/trends/explore?date=2004-01-01%202017-08-31&q=knowledge%20graph> (besucht am 7. Sep. 2017) (zitiert auf Seite 12).
- [McC+16] James P. McCusker, Deborah L. McGuinness, John S. Erickson und Katherine Chastain. What is a Knowledge Graph? 2016. URL: <https://www.authorea.com/users/6341/articles/107281-what-is-a-knowledge-graph> (besucht am 9. Sep. 2017) (zitiert auf Seite 10).

- [Ope] Apache OpenNLP. URL: <http://opennlp.apache.org/> (besucht am 7. Sep. 2017) (zitiert auf den Seiten 13, 20).
- [Psl] PSL Referenzimplementation. LINQS. URL: <https://github.com/linqs/psl> (besucht am 7. Sep. 2017) (zitiert auf Seite 53).
- [Sha06] Victoria Shannon. A 'more revolutionary' Web. 23. Mai 2006. URL: <http://www.nytimes.com/2006/05/23/technology/23iht-web.html> (besucht am 10. Sep. 2017) (zitiert auf Seite 11).
- [Sin12] Amit Singhal. Introducing the Knowledge Graph: things, not strings. Google. 16. Mai 2012. URL: <https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html> (besucht am 7. Sep. 2017) (zitiert auf Seite 11).
- [Tim] Guidelines for Temporal Expression Annotation for English. TimeML Working Group. 14. Aug. 2009. URL: <http://www.timeml.org/tempeval2/tempeval2-trial/guidelines/timex3guidelines-072009.pdf> (besucht am 18. Sep. 2017) (zitiert auf Seite 21).
- [Udv] Universal Dependency Relations. Stanford University. URL: <http://universaldependencies.org/u/dep/> (besucht am 15. Sep. 2017) (zitiert auf Seite 22).
- [Usz] Hans Uszkoreit. Repräsentationen und Prozesse in der Sprachverarbeitung. Einführung in die Computerlinguistik. URL: <http://www.coli.uni-saarland.de/~hansu/Verarbeitung.html> (besucht am 13. Sep. 2017) (zitiert auf Seite 13).
- [Wik] Wikipedia. Begriffsschrift. URL: <https://de.wikipedia.org/wiki/Begriffsschrift> (besucht am 7. Sep. 2017) (zitiert auf Seite 9).
- [Wor] WordNet. A lexical database for English. Princeton University. URL: <http://wordnet.princeton.edu> (besucht am 10. Sep. 2017) (zitiert auf den Seiten 11, 53).

Abbildungsverzeichnis

1.1	Grober Aufbau des Konstruktionsverfahrens	5
2.1	Popularität des Begriffs “knowledge graph” (Quelle: Google Trends [Goo])	12
3.1	Zusammenhang zwischen Kontexten und der \leq -Ordnung. Die Existenz eines Pfades von x nach y im obigen baumartigen Graphen, entspricht $x \leq y$	20
3.2	Beispiele für fehlerhafte Konzeptgraphen ohne dominierende Knoten. .	20
3.3	Visualisierung der Łukasiewicz Operatoren \wedge, \vee und \neg	26
3.4	Visualisierung der Loss-Funktion $\ell_j(x_1, x_2)$ für $C_j \cong X_1 \vee X_2$	27
3.5	Graph der inferierten <i>Friends-Relation</i> und der gegebenen <i>Interest-Relation</i>	30
4.1	Grobes Architekturdiagramm der Konstruktionspipeline	33
4.2	Konzeptgraphsyntax für modale Kontexte	37
4.3	NLP-Phase der Konstruktionspipeline	39
4.4	Annotationsgraph der Nachricht “I think I saw you in the red tent today.” vom 2017-06-11.	41
4.5	Annotationsgraph \rightarrow Extraktionsgraph Transformationspipeline	42
4.6	Clause-basierte Kontextschachtelung der Aussage “Tom thinks that Mary wants to marry a sailor.”	45
4.7	Annotationsgraph und daraus konstruierter Extraktionsgraph der Nachricht “I think I saw you in the red tent today.” vom 2017-06-11.	48
4.8	Wissensgraphkonstruktionsphase der Konstruktionspipeline	48

Tabellenverzeichnis

Erklärung zur Bachelorarbeit

Ich, Clemens Damke (Matrikel-Nr. 7011488), versichere, dass ich die Bachelorarbeit mit dem Thema *Probabilistische Online-Wissensgraphkonstruktion aus natürlicher Sprache* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen der Arbeit, die ich anderen Werken dem Wortlaut oder dem Sinn nach entnommen habe, wurden in jedem Fall unter Angabe der Quellen der Entlehnung kenntlich gemacht. Das Gleiche gilt auch für Tabellen, Skizzen, Zeichnungen, bildliche Darstellungen usw. Die Bachelorarbeit habe ich nicht, auch nicht auszugsweise, für eine andere abgeschlossene Prüfung angefertigt. Auf § 63 Abs. 5 HZG wird hingewiesen.

Paderborn, 10. Oktober 2017

Clemens Damke