

# Probabilistische Online-Wissensgraphkonstruktion aus natürlicher Sprache

---

Clemens Damke

*21. 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

21. Oktober 2017

**Clemens Damke**

*Probabilistische Online-Wissensgraphkonstruktion aus natürlicher Sprache*

Bachelorarbeit, 21. Oktober 2017

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

Betreuer: Dr. Theodor Lettmann

**Universität Paderborn**

*Intelligente Systeme*

Institut für Informatik

Pohlweg 51

33098 Paderborn





# Abstract

In dieser Arbeit wird ein Verfahren zur automatisierten Wissensgraph-Konstruktion aus natürlichsprachlichen textuellen Kommunikationsdaten (z. B. E-Mails) vorgestellt. Das vorgestellte Verfahren transformiert einen Stream von Textnachrichten sukzessive in einen Wissensgraphen. Es arbeitet in zwei Schritten: Im ersten Schritt wird ein Natural Language Processing (NLP) einer eingegebenen Nachricht mit der Stanford CoreNLP Bibliothek durchgeführt. Das im NLP-Schritt extrahierte Wissen wird anschließend mittels der Probabilistic Soft Logic (PSL) in den bisherigen Wissensgraphen eingefügt. Die Struktur des so konstruierten Graphen baut auf J. F. Sowa's Konzeptgraphen auf. Neu in dieser Arbeit ist die Kombination von Konzeptgraphen, CoreNLP und PSL. Der Vorteil dieser Kombination ist die hohe Ausdrucksstärke der konstruierten Wissensgraphen und die Skalierbarkeit des Verfahrens. Das Verfahren wurde implementiert und wird anhand realer Testdaten bewertet.





# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Ziele der Arbeit . . . . .	2
1.3	Aufbau der Arbeit . . . . .	4
<b>2</b>	<b>Verwandte Arbeiten</b>	<b>5</b>
2.1	Ansätze zur Wissensrepräsentation . . . . .	5
2.1.1	Logische Grundlagen . . . . .	5
2.1.2	Entwicklung maschineller Wissensrepräsentation . . . . .	7
2.1.3	Aktuelle Wissensrepräsentationsprojekte . . . . .	9
2.2	NLP-Werkzeuge . . . . .	10
2.3	Wissensgraph-Konstruktionsverfahren . . . . .	11
<b>3</b>	<b>Grundlagen und Hilfsmittel</b>	<b>15</b>
3.1	Wissensmodellierung mit Konzeptgraphen . . . . .	15
3.1.1	Syntax . . . . .	15
3.1.2	Definition . . . . .	17
3.2	Stanford CoreNLP . . . . .	19
3.3	Modellierung von HL-MRFs mit PSL . . . . .	21
3.3.1	Markov Random Fields . . . . .	21
3.3.2	Hinge-Loss MRFs . . . . .	23
3.3.3	Probabilistic Soft Logic . . . . .	26
3.3.4	Inferenzverfahren . . . . .	30
<b>4</b>	<b>Wissensgraphkonstruktion aus Kommunikationsdaten</b>	<b>33</b>
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	Wissensgraph-Konstruktionsphase . . . . .	48
4.3.1	Konstruktion des Konjunktionsgraphen . . . . .	48

4.3.2	Relationsinferenz mit PSL . . . . .	50
4.4	Implementation . . . . .	54
<b>5</b>	<b>Bewertung</b>	<b>55</b>
5.1	Qualitative Bewertung . . . . .	55
5.1.1	Bewertungsansatz . . . . .	56
5.1.2	Testdaten . . . . .	56
5.1.3	Ergebnisse . . . . .	56
5.2	Laufzeituntersuchung . . . . .	60
<b>6</b>	<b>Zusammenfassung</b>	<b>63</b>
6.1	Rückblick . . . . .	63
6.2	Ausblick . . . . .	64
<b>A</b>	<b>Anhang</b>	<b>69</b>
A.1	Verwendete Testnachrichten . . . . .	69
A.2	Verwendete Testanfragen . . . . .	71
A.2.1	Personen . . . . .	71
A.2.2	Ereignisse und Termine . . . . .	72
A.2.3	Personenbezogene Daten . . . . .	72
A.2.4	Positive und negative Aussagen . . . . .	72
A.3	Laufzeitergebnisse . . . . .	73
	<b>Literatur</b>	<b>77</b>





# 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 Arten 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 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 insbesondere:

- **Entity Resolution:** Hinzukommende Entitäten, die bereits im Graphen enthalten sind, müssen als Duplikate erkannt werden. Dies ist i. d. R. nicht trivial, da dieselbe Entität durch viele verschiedene, oftmals vom Kontext abhängige Token repräsentiert werden kann; z. B. *Bob* vs. *Robert* oder *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 Wissensgraph-Konstruktionsverfahren, 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 Wissensgraph-Konstruktion 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 Ontologie für die Domäne des Wissensgraphen, die mächtig genug ist, um möglichst diverse natürlichsprachlich beschriebene Informationen abzubilden. Im Folgenden wird diese Ontologie Wissensgraphonotologie genannt.

2. **Sprachverarbeitung:** Konzeption eines Verfahrens, welches die natürlichsprachlichen Inhalte der Nachrichten in eine für die Wissensgraph-Konstruktion geeignete Form bringt.
3. **Grapherweiterung:** Konzeption 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 Verfahrens zur Sprachverarbeitung und Grapherweiterung wird ein Wissensgraph konstruiert, der die spezifizierte Ontologie benutzt. Der Zweck dieses Wissensgraphen ist, dass er mittels eines Anfragesystems ausgelesen werden kann und so zur Beantwortung von Fragen über den Inhalt der eingegebenen Nachrichten benutzt werden kann. Die Aufgabenstellung dieser Arbeit beschränkt sich allerdings auf die Konstruktion, der Aufbau eines Anfragesystems wird nicht beschrieben.

Das gemäß Abbildung 1.1 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 Wissensgraph-Konstruktion aus Kommunikationsdaten prinzipiell ablaufen kann.

## 1.3 Aufbau der Arbeit

Die drei Komponenten Ontologie, Sprachverarbeitung und Grapherweiterung bilden die Basis des zu entwickelnden Systems und gliedern daher auch diese Arbeit. In den Kapiteln 2 bis 4 werden diese Komponenten schrittweise genauer betrachtet, was am Ende in einem konkreten Verfahren zur Wissensgraph-Konstruktion 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, eine Klasse von graphischen Modellen, welche zur Konstruktion von Wissensgraphen benutzt werden kann.

**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 wird stichprobenartig ein kleines Nachrichten-Set eingefügt. Der resultierende Wissensgraph wird anschließend manuell untersucht.

**Kapitel 6: Zusammenfassung** Abschließend werden die Ergebnisse zusammengefasst und ein Ausblick auf Möglichkeiten der Weiterentwicklung des Systems gegeben.



## Verwandte Arbeiten

Das in 1.2 beschriebene Ziel der automatisierten Wissensgraph-Konstruktion wird 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 Wissensgraph-Konstruktion 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.

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

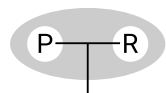
Im Gegensatz zur Prädikatenlogik gibt es keine eigene Syntax für *UND* und *ODER*; diese Operatoren müssen durch die Kombination von Negation und Implikation abgebildet werden. Zudem gibt es ausschließlich den Allquantor; eine existenzquantisierte Aussage muss durch Negation der negierten allquantisierten 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.  $\alpha$ -Graphen, umfasst alle notwendigen syntaktischen Elemente, um ein Kalkül mit der Ausdrucksstärke der Aussagenlogik zu erhalten. Die  $\beta$ -Graphen bilden die zweite Stufe und erweitern die Syntax der  $\alpha$ -Graphen, sodass die Ausdrucksstärke der Prädikatenlogik erster Ordnung erreicht wird. Sowohl für  $\alpha$ -, als auch für  $\beta$ -Graphen, ist die Vollständigkeit und Korrektheit bewiesen. Die dritte Stufe ( $\gamma$ -Graphen) wurde von Pierce nie vollendet; sie deckt in etwa die Ausdrucksstärke 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 [Beg]. 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.

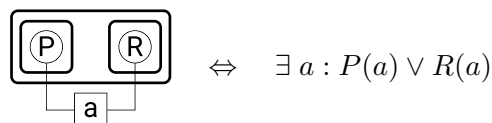
**Universelle Problemlöser** Einer der ersten maschinellen Problemlöser war der von Newell und Simon 1955 entwickelte *Logic Theorist* (LT) [NS56]. LT war in der Lage logische Aussagen zu beweisen, indem er systematisch Ersetzungsaxiome auf eine gegebene Aussage angewandt hat, bis die gesuchte Lösung 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 im Kontext der Informatik erstmals unter dem Begriff *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. Mit welcher Ontologie die Daten im Graph repräsentiert werden und wie sie in diese Repräsentation übersetzt werden, ist nicht öffentlich bekannt. Obwohl bislang nur wenig über die Technologie veröffentlicht wurde, lässt sich beobachten, dass das öffentliche Interesse an der Thematik seit Googles Ankündigung im Mai 2012 deutlich gestiegen ist (siehe Abb. 2.1).



**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 eine Kernaufgabe dieser Arbeit. Hierfür existiert bereits eine Vielzahl von *Natural Language Processing* (NLP) Werkzeugen. Trotz dieser Vielfalt lassen sich grundlegende Verarbeitungsschritte festmachen, die in den meisten Werkzeugen verwendet werden.

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

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

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

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

3. **Part-of-speech Tagging:** Abbildung von Token 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

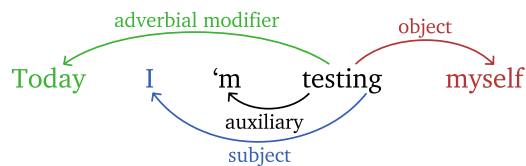
4. **Named Entity Recognition:** Klassifizierung von Token in Kategorien wie z. B. Person, Ort oder Zeitpunkt.

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

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

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

6. **Dependency Parsing:** Eine auf den Part-of-speech-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 Werkzeug 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 Wissensgraph-Konstruktionsverfahren

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 Wissensgraph-Konstruktionsverfahren 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 Wissensgraph-Konstruktion 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 Vektor  $e_i \in \mathbb{R}^H$ , welcher  $H$  Eigenschaften bzw. Features von  $i$  repräsentiert, 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 Abhängigkeitsgraph darf dabei nicht mit dem Wissensgraphen verwechselt werden: Der Abhängigkeitsgraph beschreibt statistische Abhängigkeiten zwischen Relationen, während der Wissensgraph 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+15b].

Ein konkretes Kalkül, welches sich zur Spezifikation von HL-MRFs eignet, ist die *Probabilistic Soft Logic* (PSL) [Br10][Bac+15b]. 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 seiner 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 Wissensgraph-Konstruktionsverfahrens. Der Vorteil dieses Ansatzes gegenüber Latent Feature oder Graph Feature Modellen ist, dass sich domänenspezifische Expertensysteme, z. B. Geoinformationssysteme, 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 Wissensgraph-Konstruktion 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 Hilfsmittel notwendig. Sie werden daher in den folgenden Abschnitten näher beschrieben.

## 3.1 Wissensmodellierung mit Konzeptgraphen

John F. Sowa's Konzeptgraphen [Sow76] 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, die ein Kalkül bildet, dessen 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 existenzquantisiert 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 Relationsknotens 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.

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

**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.

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

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

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

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

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.

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

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

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

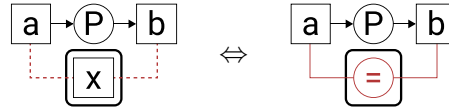
$$\Leftrightarrow \exists a, b : P(a, b) \vee R(b, a)$$

**Koreferenzkanten (coreference links)** Entspricht der Äquivalenzrelation.

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

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

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 Definition

Im vorigen Abschnitt wurde die Konzeptgraphsyntax und deren Semantik beschrieben. Bislang ist allerdings unklar, welche Kombinationen dieser Syntaxelemente erlaubt sind. So wie die Syntaxelemente prädikatenlogischer Ausdrücke nicht beliebig kombiniert werden können, unterliegen auch Konzeptgraphen gewissen Einschränkungen. In diesem Abschnitt werden die Eigenschaften beschrieben, die ein Konzeptgraph erfüllen muss. Ein Konzeptgraph  $G = (V, E)$  ist ein gerichteter Graph mit den folgenden Eigenschaften:

$$V = \{v : \text{concept}(v)\} \cup \{v : \text{relation}(v)\} \cup \{v : \text{context}(v)\}$$

$$E = \{e : \text{coref}(e)\} \cup \{e : \text{arg}(e)\} \cup \{e : \text{nest}(e)\}$$

$\text{concept}(v) : \Leftrightarrow v \in V$  ist ein Konzeptknoten

$\text{relation}(v) : \Leftrightarrow v \in V$  ist ein Relationsknoten

$\text{context}(v) : \Leftrightarrow v \in V$  ist ein Kontext, es gilt  $\text{context}(\top)$

$\text{neg}(v) : \Leftrightarrow \text{context}(v) \wedge v$  ist ein Negationskontext

$\text{coref}(v_1, v_2) : \Leftrightarrow \text{concept}(v_1) \wedge \text{concept}(v_2) \wedge (v_1, v_2) \in E$

$\text{arg}(r, v) : \Leftrightarrow \text{relation}(r) \wedge \text{concept}(v) \wedge ((r, v) \in E \vee (v, r) \in E)$

$\text{nest}(c, v) : \Leftrightarrow \text{context}(c) \wedge (c, v) \in E$

$$E \supseteq \{(\top, v) : v \in V \wedge \neg \exists c \in V \setminus \{\top\} : \text{nest}(c, v)\} \quad (3.7)$$

$$(V, \{(c, v) : \text{nest}(c, v) \wedge v \neq \top\}) \text{ ist zyklensfrei} \quad (3.8)$$

$$\neg \exists c_1, c_2, a : \text{nest}(c_1, a) \wedge \text{nest}(c_2, a) \quad (3.9)$$

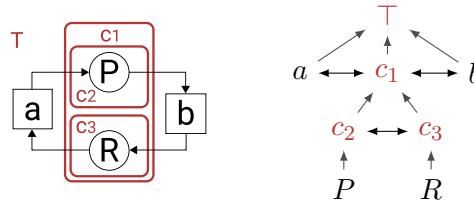
Zentral in dieser Definition sind die Kriterien (3.7), (3.8) und (3.9). Sie fordern, dass der Teilgraph  $T = (V, \{(c, v) : \text{nest}(c, v) \wedge v \neq \top\})$  ein gerichteter Baum mit Wurzel  $\top$  ist. Dabei ist  $\top$  der sog. globale Kontext, der, in Anlehnung an Peirce, auch *Sheet of Assertion* genannt wird.  $\top$  enthält alle Knoten, die keinen explizit dargestellten Elternkontext haben. Der *Elternkontext* eines Knotens  $v$  ist dabei der Kontext  $c$ , für den  $\text{nest}(c, v)$  gilt. Die sog. *umgebenden Kontexte* eines Knotens  $v$  sind alle Kontexte

$c$ , für die ein Pfad  $c \rightarrow v$  in  $T$  existiert.

Zusätzlich zu den obigen Kriterien, muss jeder Konzeptgraph  $G$  die Eigenschaft dominierender Knoten  $dom(G)$  erfüllen. Um diese Eigenschaft zu beschreiben, wird zuerst die Quasiordnung  $\leq$  eingeführt:

$$\begin{aligned} a \leq b &:\Leftrightarrow nest(b, a) \\ &\vee (\exists c \in V : nest(c, a) \wedge nest(c, b)) \\ &\vee (\exists x \in V : a \leq x \wedge x \leq b) \end{aligned} \quad (3.10)$$

Es gilt  $a \leq b$ , wenn  $b$  ein Kontext ist, der  $a$  enthält, oder wenn  $a$  und  $b$  denselben Elternkontext haben, oder wenn aufgrund der Ordnungs-Transitivität  $a \leq x \leq b$  gilt. Das größte Element gemäß  $\leq$  ist also immer  $\top$ . Zusammenfassend beschreibt  $a \leq b$ , dass der Elternkontext von  $b$  umgebender Kontext von  $a$  ist. In Abbildung 3.1 wird dieser Zusammenhang veranschaulicht.

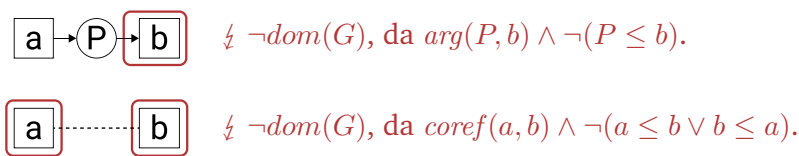


**Abb. 3.1.** Zusammenhang zwischen Kontexten und der  $\leq$ -Ordnung. Die Existenz eines Pfades von  $x$  nach  $y$  im obigen baumartigen Graphen, entspricht  $x \leq y$ .

Auf Basis von  $\leq$  lässt sich nun  $dom$  definieren:

$$\begin{aligned} dom(G) &:\Leftrightarrow \forall r, v \in V : arg(r, v) \rightarrow r \leq v \\ &\wedge \forall v_1, v_2 \in V : coref(v_1, v_2) \rightarrow (v_1 \leq v_2 \vee v_2 \leq v_1) \end{aligned} \quad (3.11)$$

Eine Intuition für diese Anforderung an Konzeptgraphen findet sich in der Semantik, die die durch  $dom$  verbotenen Kanten hätten: Sie würden die Existenz eines Atoms auszudrücken, welches durch nicht existente Variablen parametrisiert ist; dies ist nicht sinnvoll. Eine detailliertere 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. 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** Diese Annotatoren liefern, wie erwartet, eine Liste von Token bzw. eine Liste der Lemmata der Token. Es werden neben Englisch zahlreiche weitere Sprachen und ein Großteil des Unicode Zeichensatzes unterstützt. Der Tokenizer verwendet zum Finden der Token intern einen deterministischen endlichen Automaten.

**POS-Tagging** Dieser Annotator 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 dasselbe Konzept bzw. dieselbe 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 (NN) basiert. Das regelbasierte System liefert die schlechtesten Ergebnisse, das NN-System die besten.

**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 [Mur12, Kapitel 19] 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 ist ein MRF ein ungerichteter Graph  $G$ , in dem die Knoten Zufallsvariablen sind:

$$X := \text{Zufallsvektor } (X_1, \dots, X_n)$$

$$G := (\{X_1, \dots, X_n\}, E)$$

$G$  ist ein MRF, gdw. die Zufallsvariablen  $X_1, \dots, X_n$  die folgenden sog. Markov-Eigenschaften erfüllen:

#### 1. Globale Markov-Eigenschaft:

$$\text{sep}_{X_A, X_B}(X_S) \Rightarrow X_A \perp X_B \mid X_S$$

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

#### 2. Lokale Markov-Eigenschaft:

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

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:

$$\{X_i, X_j\} \notin E \Rightarrow X_i \perp X_j \mid X \setminus \{X_i, X_j\}$$

Aus der lokalen Markov-Eigenschaft folgt, dass jedes nicht adjazente Variablenpaar  $(X_i, X_j)$  bedingt unabhängig voneinander ist, sofern alle anderen Variablen gegeben sind.

**Faktorisierung** Eine häufig benutzte Methode, um die Verteilung  $P(X = x)$  eines MRFs  $G$  zu beschreiben ist die sog. Cliques-Faktorisierung:

$\mathcal{C} := \{c_1, \dots, c_m\}$  = Menge der maximalen Cliques in  $G$

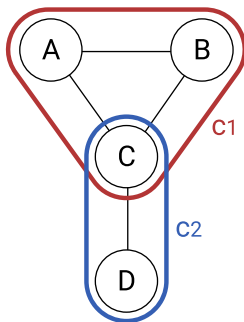
$X_c :=$  Vektor der Zufallsvariablen in der Clique  $c \in \mathcal{C}$

$\Phi_c(x_c) :=$  Cliquespotential  $\in \mathbb{R}_0^+$  der Werte  $x_c$  von  $X_c$

$$P(X = x) = \frac{1}{Z} \prod_{i=1}^m \Phi_{c_i}(x_{c_i}), \text{ mit Normalisier.konst. } Z := \sum_{x \in \mathcal{X}} \prod_{i=1}^m \Phi_{c_i}(x_{c_i}) \quad (3.12)$$

Es lässt sich zeigen, dass für jedes MRF mit einer Verteilung  $P'$  mit positiver Dichte eine Potentialfunktion  $\Phi$  existiert, sodass  $P' = P$ .

**Beispiel** Die obigen Definitionen sind bislang noch recht abstrakt. Ein exemplarisches praktisches Einsatzgebiet von MRFs ist das Lösen von SAT-Problemen [Bac+15a]. Gegeben sei die SAT-Instanz  $(\neg A \vee B \vee C) \wedge (\neg C \vee \neg D)$ . Ein Cliques-faktorisiertes MRF kann benutzt werden, um dieses Problem zu modellieren und eine erfüllende Belegung zu finden.



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

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

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

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

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

Eine Clique repräsentiert in diesem MRF eine Disjunktionsklausel und das Cliquespotential 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.

In diesem sehr einfachen Beispiel konnte jede Klausel eindeutig auf eine Clique abgebildet werden. Dies ist nicht immer der Fall; fügt man dem obigen Beispiel die Klausel  $(\neg B)$  hinzu, muss  $\Phi_{c1}(a, b, c) = (1 - b) \min\{(1 - a) + b + c, 1\}$  gesetzt werden. Falls es also eine Klausel gibt, die alle Variablen der SAT-Instanz enthält, und es somit nur eine Clique gibt, sind alle Zufallsvariablen voneinander abhängig.

**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 Lösungsverfahren existieren daher nicht. Durch Einschränken der Struktur von  $G$  und  $\Phi$ , oder durch das Erlauben von Approximationen, lassen sich MRF-Inferenzen jedoch deutlich effizienter durchführen. Wenn z. B.  $G$  ein Baum ist, es also nur 2-Cliquen gibt, kann mit dem *Belief Propagation* Algorithmus eine exakte Lösung in polynomieller Zeit gefunden werden (vgl. 2-SAT [2sa]).

### 3.3.2 Hinge-Loss MRFs

Eine Unterart von MRFs, sind die sog. Hinge-Loss MRFs [Bac+13]. 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  $\Phi$  nicht mehr frei wählbar ist. Ähnlich zum Beispiel aus 3.3.1, repräsentieren Zufallsvariablen in HL-MRFs aussagenlogische Variablen. Zufallsvariablen sind adjazent, gdw. sie in einer gemeinsamen Disjunktionsklausel vorkommen.
2. Für den Zufallsvektor  $X$  muss  $Bild(X) = [0, 1]^n$  gelten. Diese Einschränkung besteht, da jede HL-MRF-Zufallsvariable als die Wahrscheinlichkeit, dass eine aussagenlogische Variable wahr ist, interpretiert wird.
3. Die Verteilung  $P$  wird etwas anders faktorisiert:

$$P(X = x) := \frac{1}{Z} \prod_{i=1}^m e^{w_i \Phi_i(x)} \propto \exp \left( \sum_{i=1}^m w_i \Phi_i(x) \right) = \exp \left( w^\top \Phi(x) \right) \quad (3.13)$$

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

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

**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.

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

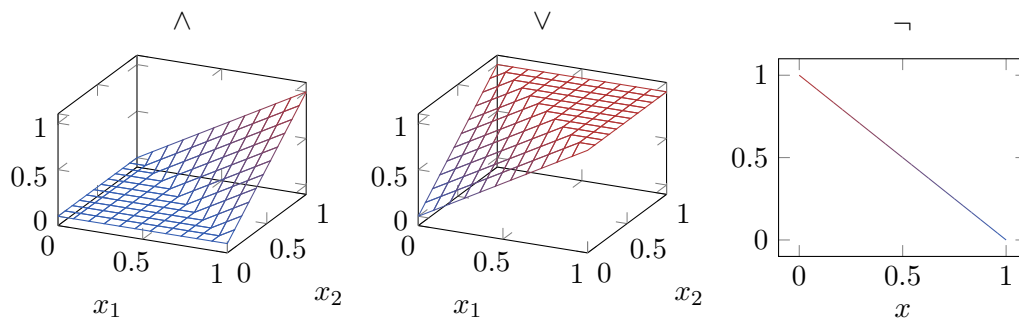
Statt jeder Clique ein Potential zuzuordnen, wird in HL-MRFs jeder Klausel ein Potential zugeordnet; dies wird getan, um die Potentiale einfacher zu halten. Zwei Klauseln mit denselben Variablen werden also durch zwei Potentiale, statt, wie in Cliquen-faktorierten MRFs, durch eines, beschrieben.

**Lukasiewicz 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.

$$x_1 \wedge x_2 := \max\{x_1 + x_2 - 1, 0\} \quad (3.14)$$

$$x_1 \vee x_2 := \min\{x_1 + x_2, 1\} \quad (3.15)$$

$$\neg x := 1 - x \quad (3.16)$$



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

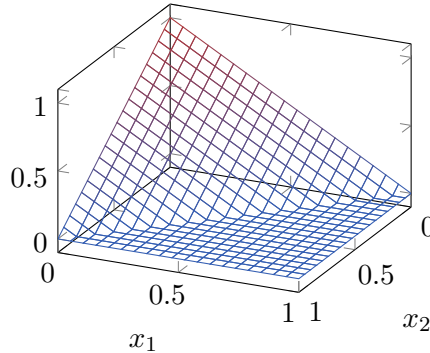
Mittels der Łukasiewicz Logik können Disjunktionsklauseln  $C_j \in C$  für eine gegebene Variablenbelegung  $x$  nun Wahrheitswerte  $\in [0, 1]$  zugeordnet werden. Dieser Wahr-

heitswert wird als Klauselpotential  $\Phi_j(x)$  verwendet. Das HL-MRF-Inferenzproblem für KNF-Formeln in Łukasiewicz Logik ist demnach

$$\begin{aligned}
& \arg \max_{x \in [0,1]^n} \sum_{C_j \in C} w_j \Phi_j(x) \\
&= \arg \max_{x \in [0,1]^n} \sum_{C_j \in C} w_j \left( \left( \bigvee_{i \in I_j^+} x_i \right) \vee \left( \bigvee_{i \in I_j^-} \neg x_i \right) \right) \\
&= \arg \max_{x \in [0,1]^n} \sum_{C_j \in C} w_j \min \left\{ \left( \sum_{i \in I_j^+} x_i \right) + \left( \sum_{i \in I_j^-} (1 - x_i) \right), 1 \right\} \quad (3.17)
\end{aligned}$$

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 dieser Constraints gesucht wird. Das Inferenzproblem lässt sich also als das Finden des folgenden Minimums beschreiben:

$$\begin{aligned}
& \arg \min_{x \in [0,1]^n} \sum_{C_j \in C} w_j \max\{\ell_j(x), 0\} \\
&= \arg \min_{x \in [0,1]^n} \sum_{C_j \in C} w_j \max \left\{ 1 - \left( \sum_{i \in I_j^+} x_i \right) - \left( \sum_{i \in I_j^-} (1 - x_i) \right), 0 \right\} \quad (3.18)
\end{aligned}$$



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

Wie Abbildung 3.4 für den zwei-elementigen Klauselfall veranschaulicht, handelt es sich bei  $\ell$  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 (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. Sie 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 Wahrheitswerte noch unbekannt sind und daher durch das PSL-Programm inferiert werden sollen.

**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:

$$\begin{aligned} 0.4 : \neg Friends(A, B) & \quad (r_1) \\ 0.5 : Friends(A, B) \wedge Friends(B, C) \wedge A \neq C \rightarrow Friends(A, C) & \quad (r_2) \\ 1 : Interest(A, X) \wedge Interest(B, X) \rightarrow Friends(A, B) & \quad (r_3) \\ \infty : Friends(A, B) \rightarrow Friends(B, A) & \quad (r_4) \\ \infty : \neg Friends(A, A) & \quad (r_5) \end{aligned}$$

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  $\infty$  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.  $Friends(A, B) = Friends(B, A)$ .
- ( $r_5$ ) 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  $\mathbb{O}$  sollen inferiert werden. Abbildung 3.5 zeigt ein mögliches MAX-SAT Ergebnis 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.



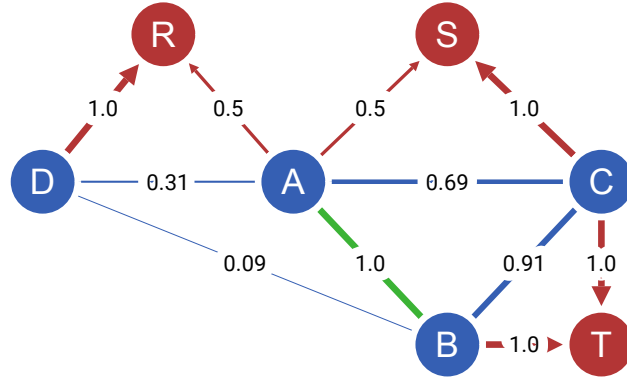


Abb. 3.5. Graph der inferierten *Friends-Relation* und der gegebenen *Interest-Relation*.

### PSL → 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  $\mathcal{A} = \mathbb{C} \dot{\cup} \mathbb{O}$ . 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  $\mathbb{O}$  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  $X \cup Y$ , wobei mit  $\cup$  die Konkatenation der Vektoren gemeint ist.
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:

$$\begin{aligned} \arg \max_{y \in \mathcal{Y}} P(Y = y \mid X = x) &= \arg \max_{y \in \mathcal{Y}} P(X \cup Y = x \cup y) \\ &= \arg \max_{y \in \mathcal{Y}} w^\top \Phi(y, x) \end{aligned} \quad (3.19)$$

Um ein HL-MRF zu erhalten, muss nun  $\Phi$  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 Klauselmeng  $C$  benutzt. Aus  $C$  wiederum lässt sich  $\Phi$  nun mittels der Łukasiewicz Logik eindeutig ableiten.

Das Gewicht jedes Potentials  $\Phi_i$  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. In [Boy+11, Kapitel 10] wird beschrieben, wie sich ADMM mit verteilten Datensets und verteilten Programmiermodellen wie z. B. *MapReduce* [DG08] oder *Pregel* [Mal+10] 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.

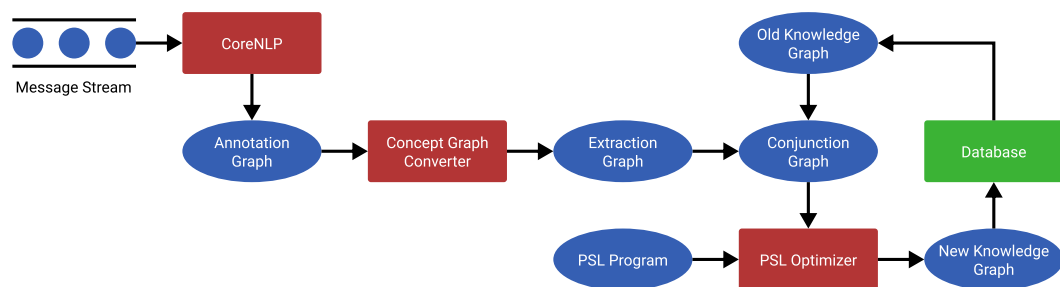
Gegeben sei also eine Bewertung der Atome aus  $\mathcal{A}$ . Werden nun die Wahrheitswerte bislang offener Atome aus  $\mathbb{O}$  bekannt, müssen ausschließlich die  $m$  höchstbewerteten Atome aus  $\mathbb{C}$  neu inferiert werden, die Wahrheitswerte aller anderen Atome werden fixiert. Je höher 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.

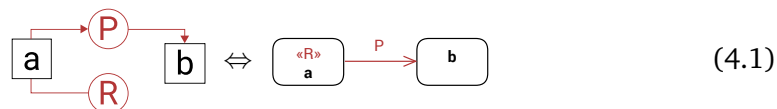
## 4.1 Wissensgraphontologie

Bevor Wissensgraphen konstruiert werden können, muss spezifiziert werden, wie diese strukturell aufgebaut sein 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 die in dieser Arbeit verwendete Ontologie aufgebaut ist.

### 4.1.1 Verwendete Prädikate

Im ersten Schritt wird geklärt, welche Prädikate in den 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 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 Arguments gemäß der  $\leq$ -Ordnung, 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. Alle Konzeptgraph-Relationen von  $x$ , beschreiben Eigenschaften von  $prop(x)$ . Es ist unrealistisch  $prop(x)$  mittels Relationen 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.

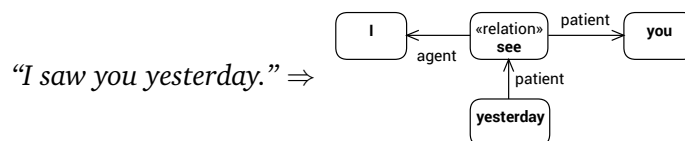
$$\text{"the good book"} \Rightarrow \boxed{\text{good}} \xrightarrow{\text{inst}} \boxed{\text{book}}$$

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**<sup>2</sup>: 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**<sup>2</sup>: 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 im Wissensgraphen Modalität repräsentiert wird. 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.

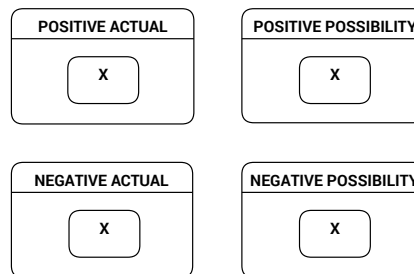


Um derartige Möglichkeiten und Notwendigkeiten direkt zu repräsentieren, reichen die vorgestellten Konzeptgraphen mit Negationskontexten nicht aus. Sowa hat dieses Problem ebenfalls erkannt und daher weitere Kontexttypen eingeführt. Die Beschreibung dieser zusätzlichen Kontexttypen ist allerdings oftmals 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 | 3. Positiver Möglichkeitskontext |
| 2. Negativer Aktualkontext | 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.

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

**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.

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

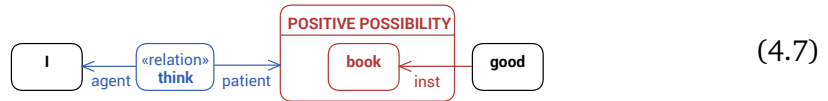
Die Aussage “ $P(x)$  gilt notwendigerweise”, wird also als “Es gibt keine Welt  $w$  in der  $P_w(x)$  nicht gilt” aufgefasst. Statt von Welten, wird in modalen Konzeptgraphen allerdings von Kontexten gesprochen.

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

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

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.

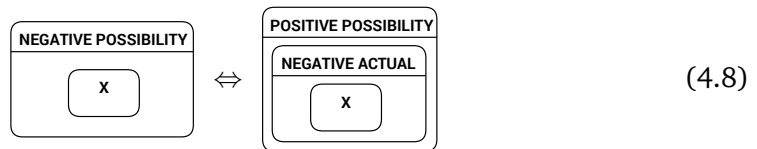
Zur Bestimmung von des Wahrheitswertes von  $\text{actual}(c)$  ist i. d. R. domänenspezifisches Wissen notwendig. Dieses domänenspezifische Wissen, kann durch sog. Kontextrelationen eingebracht werden. Da Möglichkeitskontexte zugleich Konzepte sind, lassen sie sich zu anderen Konzepten in Relation setzen. 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 dies ablaufen kann, ist bislang allerdings noch unklar (siehe 6.2).

**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.

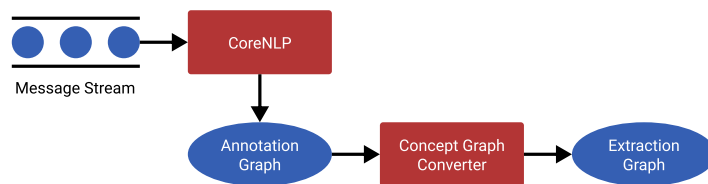


Abb. 4.3. NLP-Phase der Konstruktionspipeline

### 4.2.1 Nachrichtenformat

Vor der Beschreibung der Pipeline-Stufen, muss geklärt werden, wie die eingegebenen Nachrichten aufgebaut sind. Da diese Arbeit lediglich die grundlegende Architektur eines Wissensgraph-Konstruktionsverfahrens 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 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 [Tim] (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:

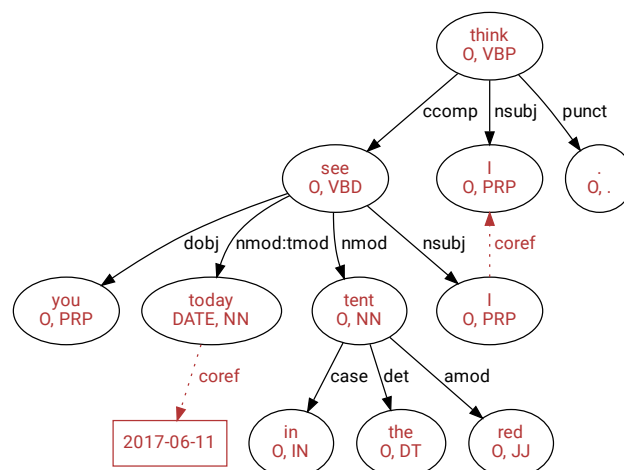
1. **Lemmata:** Die Vollformen der Token werden durch ihre Lemmata ersetzt. Dieser Änderung liegt die Annahme zugrunde, dass die syntaktische Repräsentation einer Wortbeugung i. d. R. nicht relevant ist. Durch die Lemmatisierung wird es einfacher ähnliche Begriffe aufgrund ihrer textuellen Ähnlichkeit 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 hinzugefügten 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 Konzeptgraphen mit 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 Extraktionsgraphen zu repräsentieren. Das Ziel der Extraktionsgraph-Konstruktion 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.

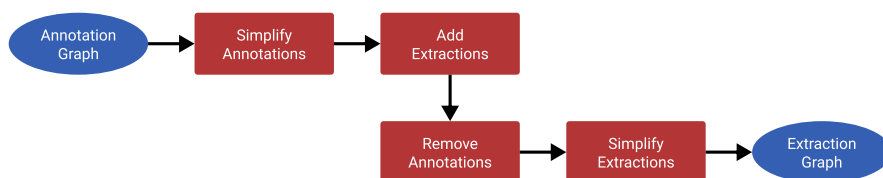


Abb. 4.5. Annotationsgraph → Extraktionsgraph Transformationspipeline

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, dessen Label die Konkate-nation der Lemmata der zusammengefassten Token ist. Die übrigbleibende Menge von Token-Knoten und Komposita bildet nun die Menge der Konzeptknoten des Wissensgraphen.

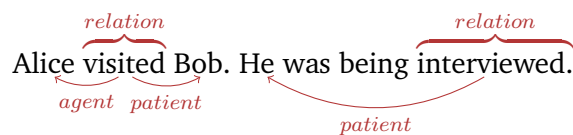
## 2. Phase: Einfügen der Extraktionen

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.

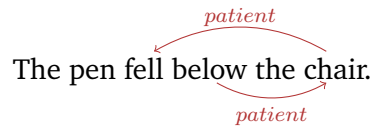
- **Prädikate:** Ein recht offensichtlicher Ansatz ist das Nutzen von Subjekt- und Objekt-Beziehungen. Direkte Objekte eines Prädikates sind i. d. R. auch dessen Patiens. Bei Subjekten ist die Diathese des Prädikates zu beachten. In Aktiv-Konstruktionen entspricht das Subjekt für gewöhnlich dem Agens. 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.

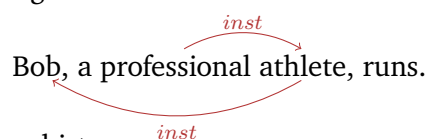
- **NER-Klassen:** Manche Konzepte besitzen NER-Klassen (z. B. Person, Ort, Organisation etc.), die bereits *inst*-Relationen abbilden. Es werden daher Konzeptknoten für alle vorkommenden Klassen hinzugefügt. Diese werden durch *inst*-Kanten mit ihren Instanzen verbunden. Die eingefügten Klassenknoten 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:

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

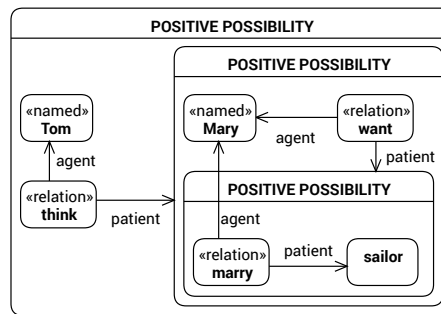
- **Attribute und Appositionen:** Ähnlich wie Kopulae drücken diese zwei grammatikalischen Beziehungen ein *inst*-ähnliches Verhältnis aus.



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* und *xcomp* 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 umgebenden Kontext (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 die 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

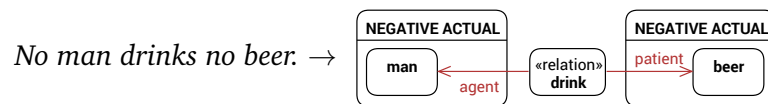
hätte, wird stattdessen ein stark vereinfachtes Modell verwendet, welches für viele, jedoch nicht für alle, Aussagen korrekte Ergebnisse liefert.

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.

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

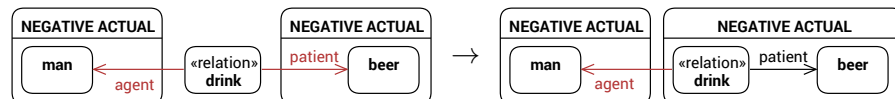
Wie das obige Beispiel zeigt, ist diese Regel nicht unbedingt korrekt. Die Interpretation gemäß der Regel ist allerdings meist zutreffend, da alternative Interpretationen im Englischen i. d. R. nur durch Betonung bestimmter Wörter ausgedrückt werden können. Das Einfügen negativer Kontexte läuft gemäß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

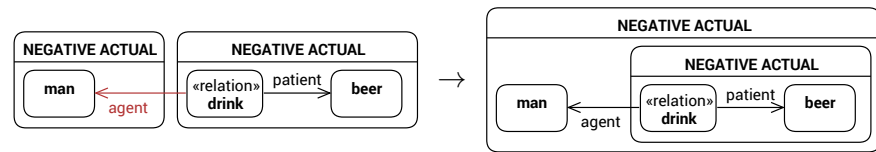


- 2.2 Es kann nun *agent*- oder *patient*-Kanten geben, die in einen der hinzugefügten Negationskontexte hineinlaufen, was gemäß der vorigen Überlegungen **unzulässig** ist. Für jede dieser Kanten von  $x$  nach  $y$  werden nun Korrekturen vorgenommen, sodass hinterher  $x \leq y$  gilt und somit keine 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 umgebende Kontext von  $x$ , der nicht  $x$  und  $y$  umgibt, in den Elternkontext 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.

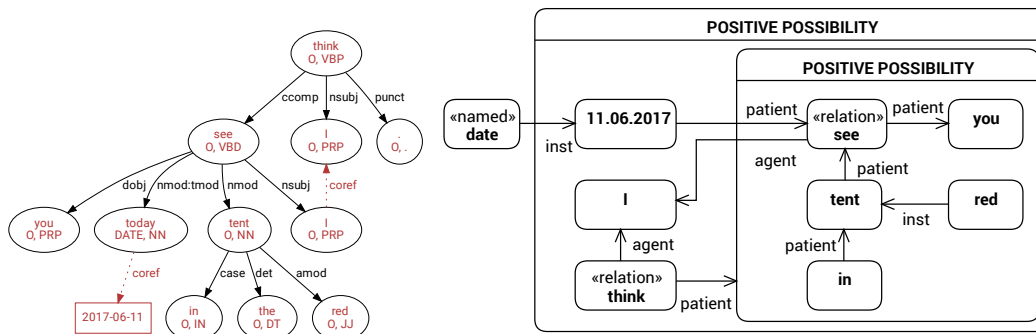
### 4. Phase: Vereinfachen der Extraktionen

**Koreferenzkanten** Nach der dritten Phase liegt zwar prinzipiell eine Konzeptgraph-Struktur 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 derselben 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 umgebenden Kontext 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

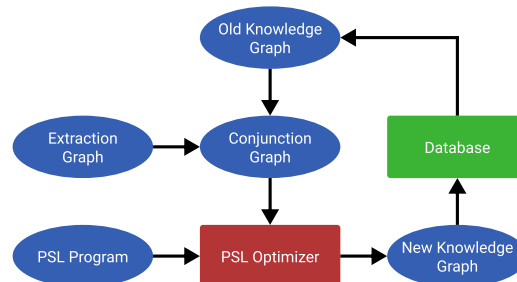
greedy durchgeführt, bis keine weiteren mehr möglich sind. Das Ergebnis ist der Extraktionsgraph.



**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 Wissensgraph-Konstruktionsphase

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.



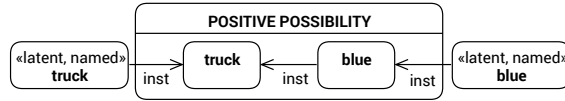
**Abb. 4.8.** Wissensgraph-Konstruktionsphase der Konstruktionspipeline

### 4.3.1 Konstruktion des Konjunktionsgraphen

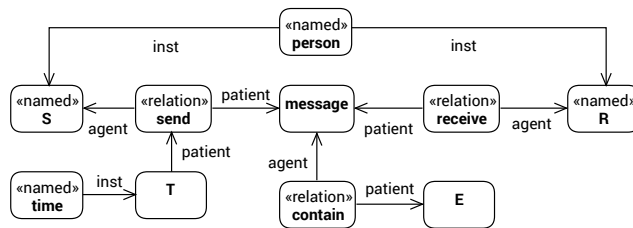
Der erste Schritt der Wissensgraph-Konstruktion ist das Bilden des sog. Konjunktionsgraphen. Der Konjunktionsgraph ist ein Konzeptgraph, der alle Informationen der einzufügenden Nachricht und des bestehenden Wissensgraphen enthält, allerdings noch keine Schlussfolgerungen aus den hinzugekommenen Informationen 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 Konjunktionsgraphen vorhanden sein. Alle Konzepte des Extraktionsgraphen, die sich innerhalb des Wurzel-Möglichkeitskontextes befinden, werden daher dupliziert und als globale la-

tente 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<sub>latent</sub>* : **UUID**: Markiert latente Konzeptknoten.
3. *label* : **UUID** × **String**: Aus der Ontologie übernommen.
4. *named* : **UUID**: Aus der Ontologie übernommen.
5. *inst* : **UUID** × **UUID**: Aus der Ontologie übernommen.
6. *relation* : **UUID**: Aus der Ontologie übernommen.
7. *agent* : **UUID** × **UUID**: Aus der Ontologie übernommen.
8. *patient* : **UUID** × **UUID**: Aus der Ontologie übernommen.
9. *similar* : **String** × **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** × **UUID**: Ordnet Kontexten ihre direkten Kindknoten zu.
14. *nest<sub>indirect</sub>* : **UUID** × **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 Atommenge  $\mathbb{O}$ , noch in der geschlossenen Atommenge  $\mathbb{C}$  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 6.2).

**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 Regelmengende 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  $\text{inst}(A, B)$ -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<sub>indirect</sub>*-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<sub>indirect</sub>*-Relationen in der Eingabe vorhanden sind, müssen letztere mit  $r_8$  und  $r_9$  inferiert werden.

- **Geschlossene Prädikate:**

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

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

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

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 Regelmenge notwendig. Um das Grundprinzip zu veranschaulichen wird in dieser Arbeit eine sehr einfache domänenspezifische Regelmenge verwendet. Abhängig vom betrachteten Datenset, sind für die Inferenz mit realen Kommunikationsdaten i. d. R. deutlich umfangreichere Regelmengen 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:

$$1 : context(ctx) \wedge patient(contain, ctx) \wedge agent(contain, msg) \quad (r_{13}) \\ \wedge inst("concept\_contain", contain) \rightarrow message(ctx, msg)$$

$$1 : message(ctx, msg) \wedge patient(send, msg) \wedge agent(send, sender) \quad (r_{14}) \\ \wedge inst("concept\_send", send) \rightarrow sender(ctx, sender)$$

$$1 : message(ctx, msg) \wedge patient(receive, msg) \wedge agent(receive, receiver) \quad (r_{15}) \\ \wedge inst("concept\_receive", receive) \rightarrow receiver(ctx, receiver)$$

$$1 : nest_{indirect}(ctx, I) \wedge inst("concept\_I", I) \quad (r_{16}) \\ \wedge sender(ctx, sender) \rightarrow inst(sender, I)$$

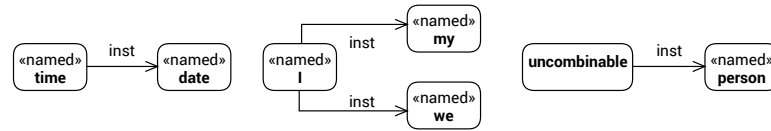
$$1 : nest_{indirect}(ctx, you) \wedge inst("concept\_you", you) \quad (r_{17}) \\ \wedge receiver(ctx, receiver) \rightarrow inst(receiver, you)$$

$$1 : inst("concept\_uncombinable", x) \wedge inst(x, p_1) \wedge inst(x, p_2) \quad (r_{18}) \\ \wedge inst(p_1, y) \wedge inst(p_2, y) \wedge p_1 \neq p_2 \wedge \neg inst(p_2, p_1) \rightarrow inst(p_1, p_2)$$



$r_{13}$ ,  $r_{14}$  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], NELL [Car+10] oder YAGO [Yag] 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.

**Wissensgraph-Konstruktion** Die Wissensgraph-Konstruktion lässt sich nun als Inferenz mit dem PSL-Programm  $R = \{r_1, \dots, r_{18}\}$  verstehen. Die geschlossenen Grundatome  $\mathbb{C}$  der Eingabe sind die Relationen im Konjunktionsgraphen, die keine offenen Grundatome in einer vorherigen PSL-Inferenz waren. Wären zuvor inferierte Atome Teil von  $\mathbb{C}$ , könnten diese während der Inferenz nämlich nicht mehr aktualisiert werden, was z. B. beim Einfügen von zusätzlicher Evidenz für eine bereits vorhandene Relation wichtig ist. Als offene Grundatommenge  $\mathbb{O}$  wird die Menge aller möglicher

Relationen verwendet, die nicht in  $\mathbb{C}$  liegen.  $\mathbb{O}$  kann und muss dabei nicht vollständig in den Speicher geladen werden. Stattdessen wird eine sog. Lazy-Inference verwendet, d. h. alle Atome in  $\mathbb{O}$  werden erst dann instanziiert, wenn sie das erste Mal in einer Grundregel auftauchen. Als Inferenzverfahren wird das in 3.3.4 vorgestellte ADMM-Verfahren benutzt. BOCI konnte nicht benutzt werden, da es in seiner aktuellen Form erfordert, dass keine neuen Grundatome zwischen zwei Inferenzen hinzukommen; beim Einfügen einer Nachricht gilt dies offensichtlich nicht. Jede eingefügte Nachricht erfordert also eine vollständige ADMM-Inferenz.

Nach dem Ausführen von ADMM wird im letzten Schritt die Menge der inferierten Atome  $\mathbb{O}$  als Menge von Konzeptgraph-Relationen interpretiert und zusammen mit  $\mathbb{C}$  als neuer Wissensgraph verwendet. Um die Graphgröße zu reduzieren werden dabei alle Atome mit einer Wahrscheinlichkeit  $< 0.5$  verworfen.

## 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 geeignete Alternative gibt und eine Reimplementation von PSL den Rahmen dieser Arbeit gesprengt hätte.

Als JVM-Sprache wurde Clojure gewählt, ein moderner Lisp-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

Die Bewertung des im vorigen Kapitel vorgestellten Verfahrens erfolgt in zwei Schritten. In 5.1 wird im ersten Schritt die Qualität der konstruierten Wissensgraphen untersucht. Anschließend wird in 5.2 die Tauglichkeit des Verfahrens im Kontext von Nachrichtenstreams betrachtet.

## 5.1 Qualitative Bewertung

Das vorgestellte Verfahren macht zahlreiche Annahmen über die Struktur und Komplexität der Eingabedaten, zudem wird lediglich ein kleiner Teil aller potentiell in Annotationsgraphen enthaltenen Informationen genutzt. In diesem Kapitel wird betrachtet, wie sich diese Einschränkungen auf die Qualität der resultierenden Wissensgraphen auswirken. Ziel ist dabei nicht, eine umfassende empirische Auswertung durchzuführen, sondern stattdessen die Stärken und Schwächen des Verfahrens anhand von stichprobenartig ausgewählten, realen Beispieldaten aufzuzeigen. Es gibt im Wesentlichen zwei Gründe für diese Entscheidung:

1. Das entwickelte Verfahren ist primär prototypisch zu verstehen. Das Ziel war nicht, ein bestimmtes Maß an Qualität zu erreichen, sondern vielmehr die Praktikabilität des Ansatzes zu demonstrieren.
2. Es existieren keine Testdatensets, die für eine empirische Analyse des entwickelten Verfahrens geeignet sind. Es existieren zwar Testdaten für die verwendeten NLP-Tasks (NER, POS-Tagging, Coreference Resolution etc.), da die Ergebnisse hierfür allerdings direkt aus CoreNLP stammen, würde mit den verfügbaren Testdaten lediglich die Performance von CoreNLP gemessen. Benötigt wären Testdaten, die das Evaluieren der Konzeptgraphstruktur erlauben, d. h. solche, die Informationen über die semantischen Abhängigkeiten und Negation von Konzepten enthalten. Fakten-Benchmarks wie *FactBench* [Fac] wären grundsätzlich für die Performance-Bewertung geeignet, hierfür wäre allerdings ein Set von Kommunikationsdaten notwendig, welches Informationen über die im Benchmark geprüften Fakten enthält; ein solches Datenset wurde bei der Recherche im Rahmen dieser Arbeit nicht gefunden.

### 5.1.1 Bewertungsansatz

Da für die Bewertung nicht auf Referenzergebnisse zurückgegriffen werden kann, müssen die Ergebnisse manuell ausgewertet werden. Das manuelle Auswerten einer repräsentativen Stichprobe von Testnachrichten hätte den zeitlichen Rahmen dieser Arbeit gesprengt. Daher wird stattdessen mit einer kleinen Menge von Testnachrichten gearbeitet. Um das Ergebnis dieser Nachrichten zu bewerten, wird der aus ihnen konstruierte Wissensgraph im ersten Schritt in eine Neo4j-Graphdatenbank [Neo] eingefügt. Anschließend wird daran eine Menge von Testanfragen (siehe A.2) gestellt und die Ergebnisse manuell mit den in den Nachrichtentexten enthaltenen Informationen abgeglichen. Auf diesem Wege kann zwar die Wissensgraphqualität nicht akkurat beurteilt werden, es ist allerdings möglich tendenzielle Stärken und Schwächen des Verfahrens zu erkennen.

### 5.1.2 Testdaten

Die verwendeten Textnachrichten stammen aus dem Enron E-Mail Datenset [Coh15]. Da das zufällige Auswählen von E-Mails aus diesem Datenset i. d. R. in unzusammenhängenden Nachrichtensets resultiert, wurden die Enron Daten nicht direkt benutzt, sondern stattdessen der Enron Threads Corpus [JG13][Enr]. Hierbei handelt es sich um ein Datenset, welches die E-Mail-Rohdaten in zusammengehörige Kommunikationsthreads gruppiert. Die Nachrichten innerhalb eines Threads sind gut als Testdaten geeignet, da darin oftmals dieselben Personen und Themen auftauchen und somit die Möglichkeit besteht, Beziehungen zwischen diesen zu finden.

Für die Testnachrichten wurde zufällig ein Testthread aus der Menge aller Threads ausgewählt, der die Schlüsselwörter *“yesterday”* oder *“tomorrow”* enthält. Ziel dabei war es, Nachrichten, in denen es um Termine oder Ereignisse geht, zu erhalten, da diese Nachrichten häufig auch ohne umfangreiches domänenspezifisches Wissen verstanden werden können. Die Wahl eines zufälligen Threads, ohne diese Einschränkung, resultierte meist in Nachrichten, die nur mit umfangreichem domänenspezifischem Wissen aus der Energie-Branche verständlich sind. Die konkret verwendeten Testnachrichten sind in A.1 zu finden.

### 5.1.3 Ergebnisse

In diesem Abschnitt wird der aus den Testdaten konstruierte Wissensgraph<sup>1</sup> untersucht. Hierfür werden vier Testanfragen benutzt. Die Ergebnisse dieser Anfragen

---

<sup>1</sup>Es befindet sich ein Neo4j-Datenbank-Export des Wissensgraphen unter <https://github.com/Cortys/bachelor-thesis/blob/master/thesis/data/evaluation/testKG.cql>, mit diesem können die Ergebnisse nachvollzogen werden.

werden auf Korrektheit geprüft und benutzt, um Verbesserungspotential des Verfahrens aufzuzeigen. Dabei wird zwischen zwei Arten von Fehlern unterschieden:

(A) Fehlerhafte Informationen

(B) Fehlende Informationen

**Personen** Die erste Anfrage (siehe A.2.1) testet, welche Personen in den Testnachrichten erkannt wurden. Da dieselbe Person in verschiedenen Kontexten verschiedene Namen haben kann, ist die Ausgabe der Anfrage eine Liste von Personen, die jeweils durch eine Liste von Namen beschrieben werden.

	people
1	Dr. Zimin Lu, TX Zimin, Zimin, Zimin Lu, address, Zimin Lu Enron Corp
2	John, John Doe
3	Vince, Vince J Kaminski
4	Duane Seppi, Dr. Seppi, Duane
5	Dr. Lu

In den Testnachrichten werden fünf Personen erkannt. Dabei lassen sich zwei Fehler beobachten:

1. Es kommen nur vier Personen in den Nachrichten vor. Der Bezeichner *Dr. Lu* von Person 5 wurde nicht als koreferent zu den Bezeichnern von Person 1 erkannt. Dies liegt daran, dass für die Erkennung von koreferenten Namen lediglich die CoreNLP Koreferenzinformationen und die Jaro-Distanzen von *named*-Konzepten berücksichtigt werden. Das Ergebnis ließe sich verbessern, indem weitere Informationen, wie z. B. der Kontext, in dem ein Bezeichner auftaucht berücksichtigt werden. (B)
2. Person 1 hat die drei fehlerhaften Bezeichner *TX Zimin*, *address* und *Zimin Lu Enron Corp* erhalten. Diese Fehler stammen aus dem Dependency Parser von CoreNLP, sie lassen sich mit der aktuellen Architektur, in der alle Informationen über den Eingabetext aus den CoreNLP-Annotatoren stammen, also nicht vermeiden. Die Ursache des Fehlers ist die Testnachricht 6 (siehe A.1), in der eine Postanschrift vorkommt; damit kann CoreNLP in der verwendeten Version 3.8.0 noch nicht umgehen. (A)

**Ereignisse und Termine** Die zweite Anfrage (siehe A.2.2) testet, wie gut *relation*-Konzepte erkannt werden. Hierfür wird aufgelistet, was *Duane Seppi* wann getan hat.

	relation	patient	time
1	miss	call	2001-03-12 09:54:24

	relation	patient	time
2	have	meeting	2001-03-13
3	send	message	2001-03-13 09:54:24
4	send	message	2001-03-14 15:15:02
5	send	message	2001-03-15 11:59:59
6	send	message	2001-03-15 21:26:00
7	receive	message	
8	talk	etc. option	
9	talk	time	
10	receive	message	
11	receive	message	
12	send	paper	
13	enjoy	talk	
14	illustrate	technique	
15	read	papers	
16	read	preprint	

Wie zu erwarten, finden sich im Ergebnis für alle gesendeten und empfangenen Nachrichten entsprechende *send*- bzw. *receive*-Einträge. Unter den übrigen Einträgen, wurden den ersten beiden Einträgen Zeitpunkte zugeordnet. Beide stammen aus der ersten Testnachricht und sind im Wesentlichen korrekt, die relativen Zeitangaben *yesterday* und *today* wurden von SUTime korrekt aufgelöst und die PSL-Regel  $r_{16}$  hat die Pronomina, die Agens der *relation*-Konzepte sind, korrekt der Person *Duane Seppi* zugeordnet. Es wurden allerdings auch verschiedene Fehler gemacht:

1. In den Einträgen 8 und 9 spiegelt sich wider, dass Fehler im CoreNLP Abhängigkeitsgraphen eine semantisch inkorrekte Repräsentation des ersten Satzes von Nachricht 3 verursacht haben: *“I would be grateful if I could talk with you some time about the typical terms one sees in Swing, take or pay, virtual storage, etc. options.”* (A)
2. Die Einträge 15 und 16 sind fehlerhaft, da bei der CoreNLP Coreference Resolution das Pronomen *I* des zweiten Satzes von Nachricht 6 fälschlicherweise mit *Duane Seppi* verknüpft wurde. (A)
3. Die Aussage von Satz 2 aus Nachricht 3 wird nicht im Wissensgraphen repräsentiert: *“This is related to some research some colleagues and I are doing applying recent innovations in Monte Carlo valuation of options with early exercise.”* (B)

Der Grund hierfür ist, dass nicht alle Universal Dependencies Abhängigkeitstypen auf ihre semantische Entsprechung in der Wissensgraphontologie abgebildet werden. Der obige Satz wird z. B. deshalb nicht korrekt im Graphen

repräsentiert, da die Abhängigkeitstypen *advcl* und *ac1* bislang nicht berücksichtigt werden.

**Personenbezogene Daten** Die dritte Anfrage (siehe A.2.3) testet, wie gut Faktenwissen, welches in der Eingabe vorhanden ist, erkannt und aus dem Wissensgraphen ausgelesen werden kann. In den verwendeten Testnachrichten teilen sich zwei Personen ihre Telefonnummern mit. Die Anfrage nach allen Nummern und zugehörigen Nummertypen (Telefon-, Fax-, Hausnummer etc.) aller Personen, liefert folgendes Ergebnis:

	name	numbers
1	Duane Seppi	number, 412-268-2298
2	Zimin Lu	phone number, 713-853-6388

Die Telefonnummern wurden korrekt erkannt. Bei der ersten Nummer fehlt jedoch die Information, dass es sich um eine Telefonnummer handelt (B); dies liegt daran, dass in der Nachricht, in der sie vorkommt nicht explizit erwähnt wird, um was für eine Art Nummer es sich handelt. Um dies zu erkennen, müssten Kommunikationsverläufe berücksichtigt werden, statt Nachrichten weitestgehend isoliert zu betrachten.

**Positive und negative Aussagen** In den vorigen Anfragen wurden Kontexte nicht berücksichtigt, da die Testnachrichten größtenteils Tatsachen beschreiben. Die vierte Anfrage (siehe A.2.4) wird daher benutzt, um die Erkennung von Negationskontexten zu testen. Konkret werden hierfür alle Relationen abgefragt, die in, von *John Doe* geschriebenen, Nachrichten erwähnt werden und für die etwas über deren Wahrheit oder Unwahrheit bekannt ist.

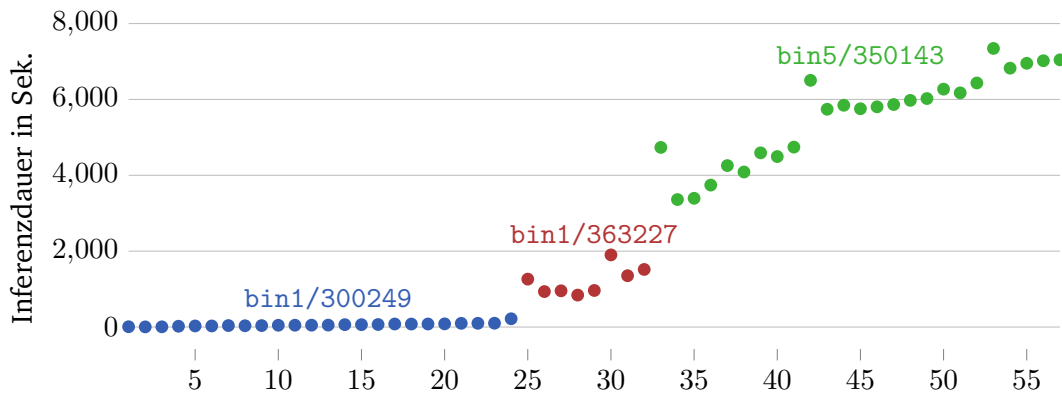
	agents	relation	patients	truth
1	I, John Doe, uncombinable, person	need	example message	POSITIVE
2	I, John Doe, uncombinable, person	have	something	POSITIVE
3	example message	contain	negation	POSITIVE
4	conversation	contain	negation	NEGATIVE
5	I, John Doe, uncombinable, person	write	message	NEGATIVE

Relation 2 wurde dabei semantisch inkorrekt abgebildet (A). Die Negation von “*I have something*” im Satz “*I’m not writing this message because I have something to say*” wurde nicht erkannt. Grund hierfür ist zum einen, dass der Abhängigkeitstyp *advcl* in der NLP-Phase nicht berücksichtigt wird. Außerdem fehlt das Hintergrundwissen über die Bedeutung des Wortes *because*. Um den Satz korrekt zu verstehen, wäre es notwendig die Implikation  $I \text{ have something} \rightarrow \neg(I \text{ write message})$  zu erkennen, mit dem Wissen,

dass *I write message* wahr ist, zu verknüpfen und schließlich  $\neg(I \text{ have something})$  zu folgern. Zu derartige Schlussfolgerungen ist das vorgestellte Verfahren bislang nicht in der Lage.

## 5.2 Laufzeituntersuchung

In diesem Abschnitt wird untersucht, inwiefern sich das beschriebene Verfahren im Kontext von Nachrichtenstreams eignet. Da in der PSL-Phase für jede eingefügte Nachricht momentan eine vollständige ADMM-basierte Inferenz durchgeführt wird, ist zu erwarten, dass mit jeder eingefügten Nachricht die Inferenzdauer steigt. Dies wurde überprüft, indem der Reihe nach die Nachrichten aus drei verschiedenen Enron E-Mail Threads eingefügt wurden; verwendet wurden die zufällig ausgewählten Threads bin1/300249, bin1/363227 und bin5/350143. Nachrichten ohne Inhalt, From- oder To-Header wurden ignoriert. Insgesamt wurden 57 Nachrichten der Reihe nach eingefügt<sup>2</sup>. Wie erwartet steigt die Inferenzdauer mit jeder eingefügten



**Abb. 5.1.** Inferenzdauer beim sequentiellen Einfügen der Testnachrichten (Tabelle in A.3)

Nachricht tendenziell an; es lässt sich jedoch beobachten, dass das Einfügen der ersten Nachricht aus einem Thread eine deutliche Erhöhung der Inferenzdauer verursacht. Eine Erklärung hierfür ist, dass die Stärke des Anstiegs der Inferenzdauer, von der Anzahl neuer Konzepte in einer Nachricht abhängt. Die Inferenzdauer-Anstiege zwischen zwei Threads werden dadurch verursacht, dass die erste Nachricht eines neuen Threads i. d. R. viele neue Konzepte einführt. Diese Erklärung wird durch die Korrelation des Anstiegs der Inferenzdauer  $\Delta t_i = t_i - t_{i-1}$  zwischen zwei Nachrichten  $m_{i-1}, m_i$  und der Anzahl neuer Konzepte  $c_i$  in der Nachricht  $m_i$  unterstützt. Die Korrelation  $Kor(\Delta t, c)$  über die gesamte Messreihe beträgt lediglich 0.187, da insbesondere im ersten Thread konstante Faktoren die Laufzeit dominieren. Betrachtet

<sup>2</sup>Die Tests wurden auf einem Desktop-Rechner mit einem AMD Ryzen 7 1800X Prozessor (8 Kerne, 16 Threads, 3.6 GHz) und 32 GB RAM unter Linux mit der Kernel-Version 4.13.5 und OpenJDK 8u162 durchgeführt. Da ein Testdurchlauf knapp 42 Stunden benötigt, wurden aus zeitlichen Gründen lediglich zwei Durchläufe durchgeführt und gemittelt.



man daher nur die Nachrichten des zweiten und dritten Threads  $m_{25}, \dots, m_{57}$  ergibt sich jedoch  $Kor_{25}(\Delta t, c) = 0.616$ .

Die gemessene Inferenzdauer liegt anfangs bei ca. 4 Sek. und steigt auf ca. 2 Std. an. Die momentane Implementation des Verfahrens ist also definitiv nicht für den praktischen Einsatz geeignet. Es existieren jedoch verschiedene Ansätze, mit denen die absoluten und asymptotischen Laufzeiten deutlich verbessert werden können. Eine triviale Optimierung ist es, nicht alle Nachrichten einzeln, sondern mehrere gleichzeitig einzufügen und so, weniger Inferenzen durchzuführen. Wenn erst der Konjunktionsgraph aller Nachrichten gebildet und dann eine einzige Inferenz durchgeführt wird, benötigt das Einfügen der Nachrichten ca. 2 Stunden, statt 42 Stunden. Weitere Ansätze werden in 6.2 beschrieben.



# Zusammenfassung

Abschließend wird nun zusammengefasst, inwiefern das in 1.2 beschriebene Ziel einer erweiterbaren, parallelisierbaren Wissensgraph-Konstruktion aus Kommunikationsdaten erreicht wurde. Hierfür werden in 6.1 die erreichten Teilziele betrachtet. Anschließend wird in 6.2 ein Ausblick auf Möglichkeiten der Weiterentwicklung des Systems gegeben.

## 6.1 Rückblick

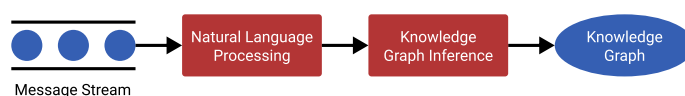


Abb. 6.1. Grober Aufbau des Konstruktionsverfahrens

In Kapitel 4 wurden Lösungen für die drei wesentlichen Teilprobleme Ontologie, Sprachverarbeitung und Grapherweiterung beschrieben. Die drei Teillösungen bilden zusammen ein Konstruktionsverfahren, welches dem zu Anfang beschriebenen Aufbau entspricht.

**Ontologie** Die Wissensgraphontologie basiert auf Konzeptgraphen. Im Gegensatz zu den Fakten-Tripel basierten semantischen Netzen, die lediglich beschreiben, welche Beziehungen zwischen Konzepten existieren, kann mittels Konzeptgraphen auch beschrieben werden, welche Beziehungen nicht, oder nur möglicherweise existieren. Die konstruierten Wissensgraphen haben also eine deutlich höhere Ausdrucksstärke, als z. B. NELL oder der Google Knowledge Graph. Diese Ausdrucksstärke ist wichtig, um die Komplexität natürlichsprachlicher Aussagen abbilden zu können.

Damit den konstruierten Konzeptgraphen eine Semantik zugeordnet werden kann, muss die Semantik der verwendeten Relationen beschrieben werden. Es wurde daher ein möglichst minimales Set von Relationen definiert (siehe 4.1.1), welches zur Beschreibung einer Vielzahl natürlichsprachlicher Aussagen geeignet ist.

Eine der in 1.2 beschriebenen Anforderungen an das Konstruktionsverfahren ist die Erweiterbarkeit, d. h. die prinzipielle Möglichkeit, nicht natürlichsprachliche Eingaben einzufügen. Konkret bedeutet dies, dass z. B. ein Bild in denselben Wissensgraph eingefügt werden kann, in den auch Textnachrichten eingefügt werden. Da die gewählten Konzeptgraph-Relationen nicht speziell für natürliche Sprache ausgelegt sind, ist dies prinzipiell möglich; es gibt keine Relationen, die bestimmte

grammatikalische Konstrukte oder Wortarten abbilden. Wie die Integration anderer Eingabearten im Detail aussehen könnte, war jedoch nicht Teil der Aufgabenstellung und wurde daher nicht näher untersucht.

**Sprachverarbeitung** Für die Transformation einer Nachricht in einen Konzeptgraphen wurde das Stanford CoreNLP Framework mit einer Menge von Transformationsregeln (siehe 4.2.3) kombiniert, welche die von CoreNLP gefundenen Strukturen auf äquivalente Konzeptgraph-Strukturen abbilden.

**Grapherweiterung** Das Einfügen des aus einer Nachricht extrahierten Konzeptgraphen in einen bestehenden Wissensgraphen erfolgt mittels PSL. Im Rahmen dieser Arbeit wurde PSL erstmals im Kontext einer Konzeptgraph-basierten Wissensgraph-Konstruktion eingesetzt. Es wurde ein Set von PSL-Regeln (siehe 4.3.2) vorgestellt, welches den Einsatz in diesem Kontext exemplarisch aufzeigt.

Neben der Erweiterbarkeit, war auch die Parallelisierbarkeit eines der in 1.2 beschriebenen Ziele. Durch die Wahl von PSL für die Grapherweiterung wurde diese Anforderung prinzipiell erfüllt. Das für die Inferenz verwendete ADMM-Verfahren lässt sich gut verteilt implementieren. In [Boy+11, Kapitel 10] wird beschrieben, wie sich ADMM mit verteilten Programmiermodellen, wie z. B. MapReduce oder Pregel, implementieren lässt. Aufbauend darauf, wird in [LDS13] eine Hadoop MapReduce-basierte Implementation vorgestellt. PSL ist prinzipiell also für den Einsatz in Cluster-Umgebungen geeignet. Die exemplarische verteilte PSL-Implementation *foxPSL* [Mag+15] demonstriert dies. In der im Rahmen dieser Arbeit erstellten Implementation wurde die PSL-Referenzimplementation verwendet, da diese am besten dokumentiert ist und die Geschwindigkeit für die verwendeten kleinen Datensets ausreichend ist.

Das Ziel ein erweiterbares, parallelisierbares Wissensgraph-Konstruktionsverfahren für Kommunikationsdaten zu konzipieren wurde somit prinzipiell erreicht. Nichts desto trotz gibt es noch zahlreiche Möglichkeiten das vorgestellte Verfahren weiterzuentwickeln.

## 6.2 Ausblick

Im Folgenden wird ein Überblick über Möglichkeiten der Weiterentwicklung und offen gebliebene Fragestellungen gegeben.

**Verbessern der Testmethode** Ein Problem bei der Konzeption des Verfahrens war der Umstand, dass es keine Kommunikationsdaten mit Referenzergebnissen gibt, anhand derer die Auswirkungen von Änderungen am Verfahren hätten bewertet werden können. Die Qualität vieler Entscheidungen, wie z. B. der Wahl der Regelgewichte,

konnte daher nur auf Basis manuell durchgeführter, stichprobenartiger Tests bewertet werden.

Um dieses Problem zu lösen, könnte ein Kommunikationsdaten-Testset entwickelt werden, mit dem sich die Performance eines Wissensgraph-Konstruktionsverfahrens repräsentativ empirisch messen lässt. Ein solches Datenset ist wichtig, um ein systematisches Weiterentwickeln des Verfahrens zu ermöglichen. Das in dieser Arbeit verwendete Enron E-Mail Datenset ist potentiell ein guter Ausgangspunkt hierfür. Zu klären ist, welche Art von Informationen die Referenzergebnisse enthalten und wie eine umfangreiche Menge solcher Referenzergebnisse konstruiert werden kann.

**NLP-Phase** Im vorgestellten Verfahren wird lediglich ein Teil der von CoreNLP extrahierten Informationen in Konzeptgraph-Strukturen übersetzt. Durch die Integration bislang ungenutzter Abhängigkeitstypen ließe sich die Qualität der Ergebnisse, wie in 5.1.3 Anfrage 2 gezeigt, deutlich verbessern.

Außerdem sollte die verwendete Heuristik zur Übersetzung von natürlichsprachlicher Negation in negative Kontexte verbessert werden. In 5.1.3 Anfrage 4 wird dies deutlich. Ein Ansatz hierfür ist z. B. das Nutzen des CoreNLP Natural Logic Annotators [MM07], welcher u. a. Quantoren, Negationen und die zugehörigen Scopes, d. h. die modifizierten Token, erkennt.

**PSL-Phase** Die verwendeten domänenspezifischen Regeln und das domänenspezifische Vorwissen sind bislang sehr rudimentär. In künftigen Arbeiten könnten vorhandene Wissensbasen, wie z. B. NELL [Car+10] oder YAGO [Yag], in die PSL-Inferenz integriert werden.

Neben der Nutzung domänenspezifischen Wissens, besteht außerdem Verbesserungspotential bei der Nutzung von Kontexten in der Inferenz. Bislang werden Kontexte nur benutzt, um die Richtung von *inst*-Relationen zu ermitteln. Das Inferieren des *actual*-Attributs von Möglichkeitskontexten (siehe 4.1.2) zur Bestimmung der Glaubwürdigkeit von Aussagen wäre z. B. eine sinnvolle Erweiterung des Verfahrens.

**Inferenz** Wie Abschnitt 5.2 gezeigt hat, besteht noch Verbesserungsbedarf bei der Geschwindigkeit der PSL-Inferenz, bevor das Verfahren in der Praxis einsetzbar ist. Um dies zu erreichen, ist die Kombination verschiedener Ansätze denkbar:

1. Nutzen einer Cluster-fähigen ADMM-Implementation, statt der PSL-Referenzimplementation. Hierfür könnte z. B. die zuvor erwähnte Hadoop MapReduce Implementation [LDS13] oder foxPSL [Mag+15] verwendet werden.
2. Weiterentwicklung von BOCI, sodass wachsende Eingabemengen unterstützt werden. Bereits im ursprünglichen Paper [Puj+15, Abschnitt 6] wird dies als offene Fragestellung für folgende Arbeiten genannt. Mittels des Inferenz-Budgets

könnte dann die Inferenzdauer nahezu beliebig verringert werden, sofern dafür eine entsprechende Verschlechterung der Inferenzqualität hingenommen wird.

3. Effizienterer Umgang mit transitiven Relationen. Bei der Inferenz werden *inst*-Atome aufgrund der Transitivität von *inst* instanziiert. Während der Inferenz sind diese Atome wichtig, sie werden momentan allerdings auch Teil des Wissensgraphen. Das Entfernen dieser redundanten *inst*-Atome, würde die Kantenanzahl des in 5.2 konstruierten Graphen um mehr als 75% verringern. Während der Inferenz würden die transitiven *inst*-Relationen dann nur temporär instanziiert. Inwiefern durch eine derartige Erweiterung der PSL-Inferenz neben der Graph-Größe auch die Inferenzdauer verringert werden kann, ist zu untersuchen.

Wie soeben gezeigt, gibt es viele Möglichkeiten das vorgestellten Wissensgraph-Konstruktionsverfahren weiterzuentwickeln. Diese Arbeit ist daher primär als ein Ausgangspunkt zu verstehen, der zeigt, dass die Kombination von Konzeptgraphen, NLP und PSL prinzipiell für die automatisierte Konstruktion von Wissensgraphen geeignet ist. Aufbauend auf dieser Grundidee könnte in künftigen Arbeiten ein in der Praxis einsetzbares Verfahren entwickelt werden.







## A.1 Verwendete Testnachrichten

Die Testnachrichten 1–7 stammen aus dem Thread bin1/363227 des Enron Threads Corpus [Enr]. Die Testnachricht 8 wurde nachträglich hinzugefügt, da in den vorherigen Nachrichten keine Negation vorkam. Da das implementierte Wissensgraph-Konstruktionsverfahren nicht speziell für E-Mails ausgelegt ist, wird nicht zwischen Absender und Absender-Adresse unterschieden. Die Absender- und Empfänger-Adressen in den Rohdaten wurden daher durch die Namen der Absender bzw. Empfänger ersetzt. Abgesehen von dieser Änderung, wurden die Inhalte und Absendezeitpunkte der Mails unverändert übernommen.

```
1  [; Message 1:
2  {:from "Duane Seppi", :to "Vince J Kaminski", :date "2001-03-13T09:54:24"
3  :content
4  "Vince,
5
6  Sorry that I missed your call yesterday. I have a meeting from 2-3
7  today
8  (Tuesday), but otherwise any time in the afternoon works for me. Let me
9  know what is convenient for you. Thanks for your help.
10
11  Duane"}
12  ; Message 2:
13  {:from "Vince J Kaminski", :to "Duane Seppi", :date "2001-03-14T08:44"
14  :content
15  "Duane,
16
17  I shall be traveling for the rest of the week but my colleague
18  Dr. Zimin Lu will call you to talk about different
19  structures.
20
21  Vince"}
22  ; Message 3:
23  {:from "Duane Seppi", :to "Zimin Lu", :date "2001-03-14T15:15:02"
24  :content
25  "Dr. Lu,
26
27  I would be grateful if I could talk with you some time about the typical
28  terms one sees in Swing, take or pay, virtual storage, etc. options.
29  This
30  is related to some research some colleagues and I are doing applying
```

```

31  recent
32  innovations in Monte Carlo valuation of options with early exercise. We
33  would like to illustrate our techniques on some examples which look
34  realistic. When would be convenient for you?
35
36  I look forward to talking with you.
37  Duane Seppi"}
38  ; Message 4:
39  {:from "Zimin Lu", :to "Duane Seppi", :date "2001-03-14T15:57"
40   :content
41   "Dr. Seppi,
42
43   My phone number is 713-853-6388. I will call you
44   tomorrow afternoon around 1:30 pm Houston time.
45
46   Zimin"}
47  ; Message 5:
48  {:from "Duane Seppi", :to "Zimin Lu", :date "2001-03-15T11:59:59"
49   :content
50   "Great. I'll be in my office. My number is 412-268-2298.
51
52   Duane"}
53  ; Message 6:
54  {:from "Zimin Lu", :to "Duane Seppi", :date "2001-03-15T13:58"
55   :content
56   "Dr. Seppi,
57
58   Nice talking to you about swing contracts and your recent work on
59   American-Monte
60   Carlo approach. I am interested to read your preprint papers.
61
62   My address is
63
64   Zimin Lu
65   Enron Corp, EB 1967
66   1400 Smith Street
67   Houston, TX 77002-7361
68
69
70   Zimin"}
71  ; Message 7:
72  {:from "Duane Seppi", :to "Zimin Lu", :date "2001-03-15T21:26:00"
73   :content
74   "Zimin,
75
76   I also enjoyed our talk. Thanks very much. I'll send you the paper as soon
77   as its done.
78
79   Duane"}
80  ; Message 8:
81  {:from "John Doe", :to "Vince J Kaminski", :date "2017-07-07T07:07:07"

```

```

82     :content
83     "Hello!
84
85     Your conversation with Dr. Seppi unfortunately doesn't contain any negation.
86     I'm not writing this message because I have something to say,
87     I just need an example message that contains negation.
88
89     John" }]

```

## A.2 Verwendete Testanfragen

Zur Evaluation des aus den Testnachrichten konstruierten Wissensgraphen, wurden drei Testanfragen verwendet, die jeweils verschiedene Aspekte des Graphen betrachten. Die Anfragen sind in der Sprache Cypher des Neo4j-Graphdatenbanksystems [Neo] geschrieben. Die Tests wurden mit Neo4j 3.2.6 und dem Graph Algorithms Plugin 3.2.2.1 [Gra] durchgeführt.

### A.2.1 Personen

```

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 people
12 return people

```

Diese Anfrage arbeitet im Wesentlichen in zwei Schritten. Im ersten Schritt wird der Teilgraph aller Konzepte gebildet, die Instanz von *person* und nicht Instanz von *I* oder *you* sind. Im zweiten Schritt wird dann die Menge aller über *inst*-Kanten schwach zusammenhängender Komponenten des Teilgraphen ermittelt. Die *label* der Konzepte innerhalb einer Zusammenhangskomponente sind die Namen bzw. Bezeichner, die dieselbe Person in verschiedenen Nachrichten hat.

Der Grund für diesen Aufbau ist, dass eine Person als ein Konzept mit gewissen charakterisierenden Eigenschaften verstanden werden kann. Die Existenz von  $inst(A, B)$  zwischen zwei Personen-Konzepten  $A, B$  bedeutet, dass  $B$  die charakterisierenden Eigenschaften von  $A$  besitzt und somit beide Konzepte auf die selbe Person verweisen. Für die Modellierung dieser Idee werden keine Koreferenzkanten verwendet, da zwei Konzepte, die auf dieselbe Person verweisen, dennoch verschieden sein können;  $B$

könnte z. B. ein Konzept sein, welches die Person *A* zu einem bestimmten Zeitpunkt repräsentiert.

## A.2.2 Ereignisse und Termine

```
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
10 order by time
```

Diese Anfrage ermittelt die Liste der *label* aller *relation*-Konzepte und ihrem Patiens, deren Agens die Person ist, die durch *Duane Seppi* bezeichnet wird. Sofern bekannt, wird den Relationskonzepten der Zeitpunkt zugeordnet, deren Patiens sie sind.

## A.2.3 Personenbezogene Daten

```
1 match ({id:"concept_number"})-[:inst]->(n)<-[:patient]-(p),
2     (p)<-[:inst]-({id:"concept_person"})-[:inst]->(person)-[:inst*0..1]->(p),
3     (number)-[:inst*0..1]->(n)
4 where exists(person.named) and person.id starts with "concept_"
5 with person.label as name, n, collect(distinct number.label) as numbers
6 return name, numbers
7 order by name
```

Diese Anfrage ermittelt zuerst alle Konzepte, die Instanz von *number* sind und zugleich Patiens einer Person sind; die Patiens-Relation zwischen einer Person und einem anderen Konzept wird benutzt, um eine Zugehörigkeit auszudrücken. Das Ergebnis ist also die Menge aller Nummern und den Personen, denen sie gehören. Um zu bestimmen, um was für eine Art von Nummer es sich dabei jeweils handelt, wird anschließend für jede Numme *B* die Menge aller Konzepte *A* bestimmt, für die  $inst(A, B)$  gilt.

## A.2.4 Positive und negative Aussagen

```
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(child) 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 "POSITIVE"
17     when size(negs) = 1 and none(c in negs where c.cnt > 1) then "NEGATIVE"
18     else "CONDITIONAL"
19 end as truth, agents, r, patients
20 return agents, r.label as relation, patients, truth

```

Diese Anfrage ermittelt zuerst alle Relationen, die von einem Kontext umgeben sind, der den Inhalt einer Nachricht beschreibt, die von *John Doe* gesendet wurde. Anschließend wird für jede Relation geprüft, ob sie negative umgebende Kontexte hat. Da in der NLP-Phase bereits doppelte Verneinungen vereinfacht werden, müssen diese nicht beachtet werden. Es sind daher drei Fälle zu beachten:

1. **POSITIVE:** Die Relation hat keine negativen umgebenden Kontexte und ist somit selbst positiv.
2. **NEGATIVE:** Die Relation hat einen negativen umgebenden Kontext, deren einziges Kind sie ist. In diesem Fall ist die Relation negiert.
3. **CONDITIONAL:** Die Relation hat mehrere negative umgebende Kontexte oder ist nicht einziges Kind ihres negativen umgebenden Kontextes. In diesem Fall ist die Existenz der Relation abhängig von der Existenz anderer Konzepte. Prinzipiell ließen sich diese Abhängigkeiten abfragen, der Einfachheit halber, wird dies hier jedoch nicht getan.

Diese Anfrage ist ein gutes Beispiel dafür, dass mit einer Graph-Anfragesprache, in diesem Fall Cypher, Kontexte nur recht umständlich berücksichtigt werden können. Das Entwickeln eines speziell für Konzeptgraphen ausgelegten Anfragesystems ist daher empfehlenswert, um die Syntax komplexerer Anfragen verständlich zu halten.

## A.3 Laufzeitergebnisse

Es folgt eine Tabelle mit den gemessenen Laufzeitergebnissen. Für jede Nachricht  $m_i$  wird die Einfügedauer  $t_i$  in Sekunden, die Anzahl der Grundatome  $a_i$ , die den Extraktionsgraphen von  $m_i$  repräsentieren, die Anzahl neuer Konzepte  $c_i$  (d. h. Konzepte mit einem *label*, welches in den vorigen Nachrichten noch nicht vorgekommen ist),  $\Delta t_i = t_i - t_{i-1}$  und die Korrelation  $Kor(\Delta t_i, c_i)$  mit  $\Delta t_i = (\Delta t_i, \dots, \Delta t_{57})$  und

$c_i = (c_i, \dots, c_{57})$  angegeben.

Die Nachrichten  $m_1, \dots, m_{24}$  stammen aus bin1/300249,  $m_{25}, \dots, m_{32}$  sind die für die Qualitätsuntersuchung verwendeten Nachrichten aus bin1/363227 (siehe A.1),  $m_{33}, \dots, m_{57}$  stammen aus bin5/350143.

$i$	$t_i$	$a_i$	$c_i$	$\Delta t_i$	$Kor(\Delta t_i, c_i)$
1	8	1372	127	8	0.186993360716651
2	4	164	11	-4	0.337595800367055
3	6	182	11	2	0.339476142713783
4	21	1038	61	15	0.341353748312038
5	29	444	17	8	0.442686652061901
6	29	92	3	0	0.4526780563415
7	39	1372	0	10	0.452141625020761
8	34	164	0	-5	0.451406966643334
9	38	182	0	4	0.450430174262627
10	47	1038	0	9	0.449536511103807
11	48	444	0	1	0.448675105114249
12	49	164	0	1	0.447651557171274
13	51	182	0	2	0.446577552781388
14	63	1038	0	12	0.445465196999229
15	63	164	0	0	0.444460572650011
16	67	182	0	4	0.443199579580348
17	80	1038	0	13	0.441939233446981
18	79	115	5	-1	0.440771686338521
19	80	164	0	1	0.440220146480954
20	84	182	0	4	0.438673742334391
21	98	1038	0	14	0.437090271348435
22	99	164	0	1	0.43560882618536
23	101	182	0	2	0.433760714061483
24	218	967	55	117	0.43180896906676
25	1263	331	16	1045	0.616039218666301
26	935	218	8	-328	0.601020622020601
27	955	509	29	20	0.610452474230775
28	842	160	8	-113	0.716209058896984
29	964	136	4	122	0.724363859913607
30	1902	278	15	938	0.724203380492174
31	1353	228	6	-549	0.713963532845561
32	1518	260	10	165	0.730825592098706
33	4735	854	44	3217	0.73815473490607
34	3361	129	4	-1374	0.110170200305455
35	3394	153	6	33	0.186051808256595

36	3742	116	3	348	0.208493429862398
37	4256	130	3	514	0.208526480093024
38	4087	134	5	-169	0.209903356442854
39	4592	261	2	505	0.244215484494565
40	4494	227	5	-98	0.260418039686248
41	4743	265	7	249	0.290769277967329
42	6503	168	5	1760	0.298036322241203
43	5741	115	1	-762	0.145789609070499
44	5846	89	1	105	0.078183800488567
45	5754	122	0	-92	0.08084280577226
46	5805	196	1	51	0.033715824934859
47	5865	360	8	60	0.023386700113926
48	5975	197	6	110	0.071379950242953
49	6023	194	3	48	0.092617414807616
50	6271	193	1	248	0.110303696745175
51	6173	120	0	-98	0.130570607467591
52	6433	68	0	260	0.05624070978447
53	7344	139	5	911	0.131118070280125
54	6824	143	5	-520	-0.989641349021773
55	6952	76	1	128	
56	7020	94	1	68	
57	7043	132	1	23	





# 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: AUA Press, 2013, S. 32–41 (zitiert auf den Seiten 13, 23).
- [Bac+15a] Stephen Bach, Bert Huang und Lise Getoor. „Unifying local consistency and MAX SAT relaxations for scalable inference with rounding guarantees“. In: Artificial Intelligence and Statistics. 2015, S. 46–55 (zitiert auf Seite 22).
- [Bac+15b] 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 13).
- [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 13, 30, 64).
- [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 8).
- [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: AUA Press, 2010, S. 73–82 (zitiert auf Seite 13).
- [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 9, 53, 65).
- [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 20).
- [Dau03] Frithjof Dau. The Logic System of Concept Graphs with Negation. Springer Berlin Heidelberg, 2003 (zitiert auf den Seiten 15, 18).

- [DG08] Jeffrey Dean und Sanjay Ghemawat. „MapReduce: simplified data processing on large clusters“. In: Communications of the ACM 51.1 (2008), S. 107–113 (zitiert auf Seite 30).
- [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 19).
- [Fre79] Gottlob Frege. Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. L. Nebert, 1879 (zitiert auf Seite 5).
- [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 8).
- [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 7).
- [JG13] Emily Jamison und Iryna Gurevych. „Headerless, Quoteless, but not Hopeless? Using Pairwise Email Classification to Disentangle Email Threads.“ In: RANLP. 2013, S. 327–335 (zitiert auf Seite 56).
- [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 12).
- [LDS13] Peter Lubell-Doughtie und Jon Sondag. „Practical distributed classification using the alternating direction method of multipliers algorithm“. In: Big Data, 2013 IEEE International Conference on. IEEE. 2013, S. 773–776 (zitiert auf den Seiten 64, 65).
- [Leh92] F. Lehmann. Semantic Networks in Artificial Intelligence. New York, NY, USA: Elsevier Science Inc., 1992, S. 6 (zitiert auf Seite 8).
- [Mag+15] Sara Magliacane, Philip Stutz, Paul Groth und Abraham Bernstein. „foxPSL: A Fast, Optimized and eXtended PSL implementation“. In: International Journal of Approximate Reasoning 67 (2015), S. 111–121 (zitiert auf den Seiten 64, 65).
- [Mal+10] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik et al. „Pregel: a system for large-scale graph processing“. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM. 2010, S. 135–146 (zitiert auf Seite 30).
- [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 11).

- [MM07] Bill MacCartney und Christopher D Manning. „Natural logic for textual inference“. In: Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing. Association for Computational Linguistics. 2007, S. 193–200 (zitiert auf Seite 65).
- [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 9).
- [Mur12] Kevin P Murphy. Machine learning: a probabilistic perspective. MIT press, 2012 (zitiert auf Seite 21).
- [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 7).
- [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 12).
- [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 12).
- [Nic13] Maximilian Nickel. „Tensor Factorization for Relational Learning“. Diss. Ludwig-Maximilians-Universität München, 2013 (zitiert auf Seite 12).
- [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 20).
- [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 7).
- [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 6).
- [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 13, 30, 31, 65).
- [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 8).
- [San90] Beatrice Santorini. „Part-of-speech tagging guidelines for the Penn Treebank Project (3rd revision)“. In: (1990) (zitiert auf Seite 19).
- [Slu87] Hans Sluga. „Frege against the Booleans.“ In: Notre Dame Journal of Formal Logic 28.1 (Jan. 1987), S. 80–98 (zitiert auf Seite 6).

- [Sow11] John F. Sowa. „Peirce's tutorial on existential graphs“. In: *Semiotica* 2011.186 (Jan. 2011) (zitiert auf Seite 6).
- [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 den Seiten 8, 15).
- [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 19).
- [WR10] Alfred North Whitehead und Bertrand Russell. *Principia mathematica*. 1910 (zitiert auf Seite 7).

## Webseiten

- [2sa] 2-satisfiability. Wikipedia. 28. Aug. 2017. URL: <https://en.wikipedia.org/wiki/2-satisfiability> (besucht am 15. Okt. 2017) (zitiert auf Seite 23).
- [Beg] Begriffsschrift. Wikipedia. 23. Sep. 2017. URL: <https://de.wikipedia.org/wiki/Begriffsschrift> (besucht am 15. Okt. 2017) (zitiert auf Seite 7).
- [BH] Dan Brickley und Ivan Herman. Semantic Web Interest Group. W3C. URL: <https://www.w3.org/2001/sw/interest/> (besucht am 10. Sep. 2017) (zitiert auf Seite 9).
- [Coh15] W. W. Cohen. Enron Email Dataset. 8. Mai 2015. URL: <https://www.cs.cmu.edu/~wcohen/enron/> (besucht am 11. Okt. 2017) (zitiert auf Seite 56).
- [Cor] Stanford CoreNLP. Stanford University. URL: <https://stanfordnlp.github.io/CoreNLP> (besucht am 7. Sep. 2017) (zitiert auf den Seiten 11, 19, 54).
- [Enr] Email Disentanglement. TU Darmstadt. URL: <https://www.ukp.tu-darmstadt.de/data/text-similarity/email-disentanglement/> (besucht am 11. Okt. 2017) (zitiert auf den Seiten 56, 69).
- [Fac] FactBench. URL: <https://github.com/SmartDataAnalytics/FactBench> (besucht am 18. Okt. 2017) (zitiert auf Seite 55).
- [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 10).
- [Gra] Neo4j Graph Algorithms. URL: <https://neo4j-contrib.github.io/neo4j-graph-algorithms/> (besucht am 12. Okt. 2017) (zitiert auf Seite 71).
- [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 8).
- [Neo] Neo4j. Neo4j, Inc. URL: <https://neo4j.com/> (besucht am 12. Okt. 2017) (zitiert auf den Seiten 56, 71).

- [Ope] Apache OpenNLP. URL: <http://opennlp.apache.org/> (besucht am 7. Sep. 2017) (zitiert auf den Seiten 11, 19).
- [Psl] PSL Referenzimplementation. LINQS. URL: <https://github.com/linqs/psl> (besucht am 7. Sep. 2017) (zitiert auf Seite 54).
- [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 9).
- [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 9).
- [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 den Seiten 20, 40).
- [Udv] Universal Dependency Relations. Stanford University. URL: <http://universaldependencies.org/u/dep/> (besucht am 15. Sep. 2017) (zitiert auf Seite 20).
- [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 11).
- [Wor] WordNet. A lexical database for English. Princeton University. URL: <http://wordnet.princeton.edu> (besucht am 10. Sep. 2017) (zitiert auf den Seiten 9, 53).
- [Yag] YAGO: A High-Quality Knowledge Base. Max Planck Institute for Informatics. 2017. URL: <https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/> (besucht am 17. Okt. 2017) (zitiert auf den Seiten 53, 65).



# Abbildungsverzeichnis

1.1	Grober Aufbau des Konstruktionsverfahrens . . . . .	3
2.1	Popularität des Begriffs “knowledge graph” (Quelle: Google Trends [Goo])	10
3.1	Zusammenhang zwischen Kontexten und der $\leq$ -Ordnung. Die Existenz eines Pfades von $x$ nach $y$ im obigen baumartigen Graphen, entspricht $x \leq y$ . . . . .	18
3.2	Beispiele für fehlerhafte Konzeptgraphen ohne dominierende Knoten. .	18
3.3	Visualisierung der Łukasiewicz Operatoren $\wedge, \vee$ und $\neg$ . . . . .	24
3.4	Visualisierung der Loss-Funktion $\ell_j(x_1, x_2)$ für $C_j \cong X_1 \vee X_2$ . . . . .	25
3.5	Graph der inferierten <i>Friends-Relation</i> und der gegebenen <i>Interest-Relation</i> . . . . .	29
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	Wissensgraph-Konstruktionsphase der Konstruktionspipeline . . . . .	48
5.1	Inferenzdauer beim sequentiellen Einfügen der Testnachrichten (Tabelle in A.3) . . . . .	60
6.1	Grober Aufbau des Konstruktionsverfahrens . . . . .	63









# 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, 21. Oktober 2017*

---

Clemens Damke