Semantic Decomposition and Marker Passing in an Artificial Representation of Meaning

vorgelegt von
Dipl.-Inform.
Johannes Fähndrich
Geb. in Lahr im Schwarzwald

von der Fakultät IV — Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
— Dr.-Ing. —

genehmigte Dissertation

Tag der wissenschaftlichen Aussprache: 19. Feb. 2018

Berlin 2018

Abstract

The research area of Distributed Artificial Intelligence aims at building intelligent agent systems. Multi-Agent Systems have been applied successfully in many domains, from an intermodal planning domain to cascading security thread simulations. But still, agents struggle with the meaning of concepts used in language. Intelligence needs language to form thoughts. Thus, the challenge addressed in this thesis is to provide a computable representation of meaning and evaluate its usefulness. Based on the theory of a mental lexicon and the thesis that meaning is a combination of symbolic and connectionist parts, I investigate the use of the theory of Natural Semantic Metalanguage (NSM) to build an artificial representation of meaning. I show that the use of NSM for creating a semantic graph out of different information sources can be utilized as a basis for Marker Passing algorithms.

The Marker Passing algorithm encodes symbolic meaning to guide the reasoning over the connectionist semantic graph. Through the combination of a semantic graph and symbolic Marker Passing, I can combine connectionist and symbolic approaches to AI research to create my artificial representation of meaning.

To test my approach, I build a semantic distance measure, a word sense disambiguation algorithm and a sentence similarity measure which all go head to head with the state-of-the-art. I apply those approaches to two use cases: A semantic service match marking and a context-dependent heuristics. I evaluate my heuristic by utilizing them in AI problem-solving component which uses AI planning guided by my heuristic.

Zusammenfassung

Die Wissenschaft im Bereich der verteilten künstlichen Intelligenz untersucht unter anderem Multi-Agenten Systeme und deren Anwendung in verschiedenen Bereichen. Solche intelligenten verteilten Systeme finden beispielsweise erfolgreich Einsatz bei der Planung intermodaler Routen oder bei der Simulation von Kaskaden Effekten durch Sicherheitsbedrohungen. Dabei entwickeln Agenten immer mehr Intelligenz zur autonomen Lösunge von neuen Problemen. Agenten kämpfen jedoch noch immer mit der Bedeutung von Konzepten der natürlichen Sprache. Intelligenz benötigt jedoch Sprache um Gedanken zu formen. Deshalb wird in dieser Arbeit die Herausforderung angegangen eine künstliche Repräsentation von Bedeutung zu erschaffen und deren Nutzbarkeit zu evaluieren.

Basierend auf der Theorie eines mentalen Lexikons und darauf, dass Bedeutung aus zwei Teilen besteht (Symbolischer und Konnektivistischer Bedeutung), untersucht diese Arbeit die Verwendung der Natural Semantic Metalanguage (NSM) zum Erstellen einer künstlichen Repräsentation von Bedeutung. Hauptaugenmerk liegt dabei auf der automatischen Erzeugung eines semantischen Graphen, der durch Marker Passing Ansätze genutzt werden kann. Der semantische Graph wird dabei basierend auf der NSM Theorie aus verschiedenen Informationsquellen automatisch erstellt. Der Marker Passing Algorithmus beschreibt dabei den symbolischen Teil unseres Ansatzes. Die symbolische Information der Marker wird dazu verwendet diese geeignet über den semantischen Graphen zu verteilen. Durch die Verteilung der Marker wird eine Art von Schlussfolgerung modelliert. Durch die so entstandene Kombination aus Dekomposition und Marker Passing kann eine Mischung aus symbolische und konnektivistische Bedeutung entstehen.

Die so entstandene künstliche Repräsentation von Bedeutung wird durch mehrere Experimente getestet: Ich verwende sie um ein semantisches Distanzmaß zu bauen, erstellen einen Ansatz zur Auflösung von Mehrdeutigkeit von Worten in natürlicher Sprache und erzeugen einen neuen Ansatz zur Bestimmung von Satzähnlichkeit. Dabei konnte gezeigt werden, dass die so entstandenen Ansätze dem Stand der Technik in nichts nachstehen. Des Weiteren teste ich an zwei Anwendungen ob meine künstliche Repräsentation von Bedeutung wirklich Bedeutung formalisiert: erstens anhand einer Semantischen Service Matching-Komponente und zweitens einer kontextabhängigen und zielorientierten Heuristik. Diese Heuristik wird durch den Einsatz in einem Planungsalgorithmus evaluiert.

Acknowledgements

I want to express my gratitude and recognition to my doctoral supervisor Prof. Dr. Dr. h.c. Sahin Albayrak from whom I have had the opportunity to learn a lot about science, academia and the work I want to do in my life. I want to thank Prof. Dr. Rainer Unland for his support in all those years mentoring me in the Doctoral Consortia. I wish to thank Prof. Dr. Michael Weyrich, whose constructive criticism has kept me on my scientific toes. In addition, I want to thank the team of scientists at the Technische Universität Berlin for their constant support. Especially Sebastian Ahrndt for his enthusiastic criticism and Dr. Frank Trollmann for the many fruitful discussions and the scientific mentoring. Without all of them, this would have been less fun.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

List of Publications

This section lists the publications which are used in this dissertation.

Fähndrich, J., Masuch N., Borchert L., and Albayrak, S. **Learning Mechanisms on OWL-S Service Descriptions for Automated Action Selection**, Presented at the IoA at the Conference on Autonomous Agents and MultiAgent Systems (AAMAS), pp. 41-57, 2017. [102]

Fähndrich, J., Küster, T., Masuch N., and Albayrak, S. **Semantic Service Management and Orchestration for Adaptive and Evolving Processes**, In: International Journal on Advances in Internet Technology, vol. 9, no. 4, pp. 75-88, 2016. [100]

Fähndrich, J., Küster, T., Masuch N., and Albayrak, S. **Semantic Service Management for Enabling Adaptive and Evolving Processes**, Presented at the International Conference on Internet and Web Applications and Services, Valencia, pp. 46-53, 2016. [101] (Best Paper Award)

Fähndrich, J., Weber, S., Ahrndt, S., and Albayrak, S. **Design and Use of a Semantic Similarity Measure for Interoperability Among Agents**, Presented at the German Conference on Multiagent System Technologies (MATES), pp. 41- 57, 2016. [104]

Fähndrich, J., Ahrndt, S., and Albayrak, S. **Self-Explanation through Semantic Annotation and (automated) Ontology Creation: A Survey**. Presented at the International Symposium Advances in Artificial Intelligence and Applications (AAIA), pp. 1-15, 2015 `http://doi.org/10.15439/2015F416` [99]

Fähndrich, J., Ahrndt, S., and Albayrak, S. **Formal Language Decomposition into Semantic Primes.**, In: Advances in Distributed Computing and Artificial Intelligence Journal (ADCAIJ), 3(8), pp. 56, 2014 `http://doi.org/10.14201/ADCAIJ2014385673` [98]

Fähndrich, J., Ahrndt, S., and Albayrak, S. **Are There Semantic Primes in Formal Languages?**, Presented at the International Conference on Distributed Computing and Artificial Intelligence (DCAI), 290 (Chapter 46), pp. 397-405, 2014, `http://doi.org/10.1007/978-3-319-07593-8_46` [97] (Best Paper Award)

Fähndrich, J., Masuch, N., Yildirim, H., and Albayrak, S. **Towards Automated Service Matchmaking and Planning for Multi-Agent Systems with OWL-S Approach and Challenges.**, Presented at the International Conference on Service-Oriented Computing (ICSOC) Workshops (Vol. 8377), pp. 240, 2013, `http://doi.org/10.1007/978-3-319-06859-6_21` [103]

Fähndrich, J. **Best First Search Planning of Service Composition Using Incrementally Refined Context-Dependent Heuristics**, Presented at the German Conference Multiagent System Technologies (MATES), pp. 404, 2013 `http://doi.org/10.1007/978-3-642-40776-5_34` [93]

Fähndrich, J., Ahrndt, S., and Albayrak, S. **Towards Self-Explaining Agents.**, presented at the International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS), 221 (Chapter 18), pp. 147-154, 2013, `http://doi.org/10.1007/978-3-319-00563-8_18` [96]

Fähndrich, J., Ahrndt, S., and Albayrak, S. **Self-Explaining Agents.** Jurnal Teknologi (Science and Engineering), 63(3), pp. 53-64, 2013, `http://doi.org/0.11113/jt.v63.1955` [95]

Fähndrich, J. **Exploring Self-Explanation : The System Side.**, Presented at the German Conference on Multiagent System Technologies (MATES), p. 12, 2012 [94]

List of Supervised Theses

This section lists the supervised student theses which have been basis to some of the sections of this dissertation.

Hannes Kanthak, Ein semantischer Ansatz zum Lösen von Winograd Schemen. Technische Universität Berlin, Bachelor (2017)

Maik Wischow, Konfliktlösung bei der Erweiterung von Wissensgraphen. Technische Universität Berlin, Bachelor (2016)

Tom König, Untersuchung semantischer Distanzmaße auf der Grundlage von Aktivierungsausbreitung über ontologiebasierten Hypergraphen. Technische Universität Berlin, Bachelor (2016)

Nico Tobias Schneider, Quantifizierung semantischer Satzähnlichkeit basierend auf lexikalischer Dekomposition und Marker Passing. Technische Universität Berlin, Bachelor (2016)

Benjamin Brand, Semantic Distance of Service Descriptions through Activation Spreading. Technische Universität Berlin, Diploma (2016)

Florian Marienwald, How Does the Interpretation Of Word Relations Help Word Sense Disambiguation via a Marker Passing Approach? Technische Universität Berlin, Bachelor (2016)

Pascal Lukanek, Ein auf Marker-Passing basierender Word-Sense-Disambiguation Ansatz. Technische Universität Berlin, Bachelor (2016)

Sabine Weber, The roles of Synonyms in a semantic decomposition. Technische Universität Berlin, Bachelor (2015)

Ghadh, Altayyar, An editor for manual semantic decomposition, Technische Universität Berlin, Bachelor (2015)

Zeinab, Sawan, Goal driven Heuristics based on Service Descriptions, Technische Universität Berlin, Bachelor (2015)

Part I.

Introduction

1. Introduction

Artificial Intelligence (AI) helps to solve ever more complex problems. The use of increasingly sophisticated software enables us to automate many tedious tasks, perform better research and grasp a better understanding of the world, e.g., playing GO [395], or fighting cancer [90]. *"Despite all these developments, the promises of strong artificial intelligence set forth in the 1960s have not been fulfilled."* [68, p. 7], meaning that AI is not able to understand natural language [419], construct plans on dynamic domains [135], or do common sense reasoning like humans [356]. These kind of problems are solved by a so called "strong AI" [353][1]. A strong AI is able to learn new problem-solving skills or can get to know new topics, in difference to special purpose AI like most chess AI which is, e.g., unable to drive a car.

One of the reasons for human intelligence might be the ability to think. Having a language to formulate thoughts, meaning and ideas helps us to handle unknown situations with adaptiveness and dynamic behavior. Part of the capacity to think is reasoning, which does not always "obey the rules of classical logic" but gives us my common sense [128]. The foundation for language to think is a representation of meaning. Consequently, research in AI analyzes how methods from Mathematics, Linguistics, Psychology, Philosophy and Computer Science can be used to create machines with the ability to represent meaning.

One approach to AI is the intelligent component "agent" [338]. An agent models human intelligence behavior, like goal orientation or autonomy [417], to act intelligently. Grasping the meaning of things, events or actions allows agents to react appropriately and makes them able to react to change. As a result, agents become more robust in their problem-solving skills. The adaption of an agent becomes easier if the agent can integrate new concepts into its knowledge and use them for future reasoning processes. In humans, we[2] call this understanding of concepts, meaning [234]. One possibility for an agent to reach its goal is to use the help of other agents.

In this work we will firstly have a look at how meaning is described, and on this basis I will develop two mechanisms to integrate a meaning representation into agents: one to formalize semantic information and one to perform reasoning using this information.

During research on meaning, I found the difficulty to measure meaning directly [296], this is why the AI community has established experiments which allow us to collect evidence on the existence of meaning in humans and machines by observing their behavior in specific test situations. This can be seen in the testing of human knowledge, e.g., in schools and universities tests. For an AI these tests are challenges with defined data sets. Here we start out with my hardest experiment (the creation of heuristics for AI planning) and explain why the other experiments questions have to be answered first.

Planning anticipates acting through reasoning [133]. Planning allows humans to approach complex problems and is envisioned as key for artificial agents to become more autonomous

[1]*strong AI* (sometimes called *full AI* or *hard AI*) [213, p. 260] refers to a human level intelligence.

[2]If the reference "we" is used in this thesis, this "we" references the author and the reader in the form of a *pluralis modestiae*.

and proactive [339]. The ability to plan enables agents to solve problems that have not been programmed during design time; creating agents which adapt to changes in (dynamic) environment. One dynamic environment is a Service Oriented Computing (SOA) [89] where distributed services undergo change during runtime.

In complex planning problems heuristics are needed to solve a problem. The creation of **heuristics** (see Section 10.5) during runtime may leads to the encounter of new concepts, which then lead us back to my original question: How can AI make sense of new concepts? For heuristics this means interpreting the new concepts and adding additional information to classical heuristic approaches: A function $H : state \rightarrow \mathbb{R}^+$ is called heuristic [337, p. 92] and estimates the distance from a state to a given goal. I extend this definition of heuristic to: $H : service \times state \times goal \rightarrow \mathbb{R}^+$ making the heuristic more dynamic since now it can adapt to changing goals and services. Here the heuristic determines the usefulness w.r.t. the goal of the given service in the current state (e.g. by comparing the service to a request). We integrate service and goal description into the heuristic because if only a state is the information source for the heuristic, we miss out on information like the description of the service. This leads to the experiment to evaluate a service according to some service needs according to a goal in a planning problem. This experiment is done by implementing a service match maker and testing it in one of my experiments in a **Service Matching** (see Section 10.4) context. Heuristics can be build by simplifying the given problem [304]. Those implications are for example used by Fast-Forward-Planner by removing the delete list of service effects [171]. Because of the open world assumption, this does not work in service planning. Other heuristics guide specific problems which then can not be used in problem independent (sometimes called general purpose) planning [304], because they do not adapt to e.g. changes in the goal or to new services. Thus we need another way of gathering information about the available service and the goal. In this work, this information is the meaning of the concepts used to describe e.g. the service or goal. The service descriptions are e.g. given in sentences, means that we have to compare more than just single concepts. This leads to the need of comparing sentences, using a **Sentence Similarity** measure (see Section 10.3). Such a sentence similarity measure gives use a next experiment on how well my meaning representation can capture the meaning of sentences. Ambiguous word carry different meanings in different contexts. Selecting the right word sense leads us to the next experiment: which word sense is the right one to select in a context of use. This leads us to my next experiment for **Word Sense Disambiguation** (see Section 10.2). To be able to use services of agents programmed by other developers, using other models and with that other ontologies, the agent needs to integrate new concepts into its knowledge. The interpretation of new concepts requires to identify if a new concepts is equivalent to a already known concept (e.g. by using a semantic similarity measure). This leads us to my experiment with a **Semantic Similarity** measure (see Section10.1). The recognition of similarity is a basic task to form more sophisticated and abstract abilities for reasoning [131, p. 197]. All of those subproblems show an aspect of meaning. Each experiment analyzes the capability of the representation of meaning of a different use. With all those experiments we will gather evidence on whether my representation of meaning is a useful one or not.

In Section 1.1, I will motivate why we need an artificial representation of meaning, explain how I derived the creation of an artificial representation of meaning as my main goal and explain how the work, done in this doctoral thesis, fits into the bigger picture of AI research. Afterwards, I work out the problems we need to solve to create such representation Section 1.2. Regarding

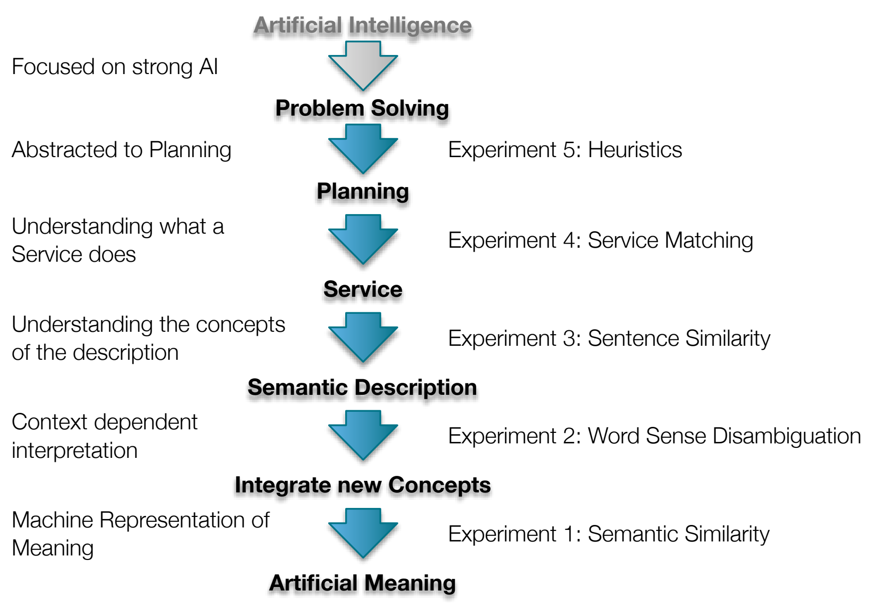

Artificial Intelligence
Focused on strong AI
Problem Solving
Abstracted to Planning
Experiment 5: Heuristics
Planning
Understanding what a Service does
Experiment 4: Service Matching
Service
Understanding the concepts of the description
Experiment 3: Sentence Similarity
Semantic Description
Context dependent interpretation
Experiment 2: Word Sense Disambiguation
Integrate new Concepts
Machine Representation of Meaning
Experiment 1: Semantic Similarity
Artificial Meaning

Figure 1.1.: Hierarchy of arguments towards the need for artificial meaning.

this problem, I formulate hypotheses in my Research Statement in Section 1.3. I then derive research questions which I want to answer in this work in Section 1.4. Finally, I will look at the abstract research approach in Section 1.5.

1.1. Motivation

Successfully creating an artificial representation of meaning will advance us on the task of building machines that help us to solve ever more complex problems. The Figure 1.1 depicts the abstraction of the here hold argument, starting from the research gap shown in the introduction: lack of general purpose problem-solving skills in AI agents. This section has the goal of narrowing my research scope from the general research area of AI to **problem solving**, down to the need of a representation of meaning. To gain insight into how to represent and use meaning (the part of AI we are interested in) we abstract problem solving to **planning** so that we can use computers to help with the problem-solving. The usefulness of planning mechanisms falls and stands with the used heuristic.

Planning can be used to create service compositions. In the research area of service composition, the runtime is shaped by dynamically appearing and disappearing services. One way to create service composition is to combine the development paradigm of Service Oriented Architecture (SOA) [89] with AI planning techniques [133] in Multi-Agent Systems (MAS) [110, 403]. Here we see agents as a service provider and as planning entity. This means that the planning agent try to find a solution to fulfill one of its goals by planning, including the services of other agents. If the agent can reach its goal with its own services, then no interaction with other agents is needed. I focus on problems where an agent needs the services provided by other agents [331], which is motivated in both SOA and MAS as distributed systems. Here the prerequisite is, that agents can integrate new services, e.g., proposed by other agents, into their own problem solution [373, 372]. With that integration, the search space of available services grows, creating more effort with the search for a plan. The question at hand is: *Which services* *are useful for an agent given its current goal?* I answer this question by creating a heuristic, which guides the search for useful combinations of services during the planning process. In the area of service composition, those heuristics cannot be created during design time, as the designers neither know about the available services nor know about the actual goals an agent pursues. Subsequently, these unknown components make it necessary to create heuristics during runtime.

The creation of heuristics (Experiment 5) during runtime may lead to the encounter of new concepts, which then lead us back to my original question: How can AI make sense of new concepts? For heuristics this means interpreting the new concepts and adding additional information to classical heuristic approaches: A function $H : \mathit{state} \rightarrow \mathbb{R}^+$ is called heuristic [337, p. 92] and estimates the distance from a state to a given goal. I extend this definition of heuristic to: $H : \mathit{service} \times \mathit{state} \times \mathit{goal} \rightarrow \mathbb{R}^+$ making the heuristic more dynamic since now it can adapt to changing goals and services. Here the heuristic determines the usefulness w.r.t. the goal of the given service in the current state (e.g., by comparing the service to a request). I integrate service and goal description into the heuristic because if a state is the only information source for the heuristic, we miss out on important information. This leads to the experiment to evaluate a service according to some service needs according to a goal in a planning problem. This experiment is done by implementing a service match maker and testing it in one of my experiments in a service matching (Experiment 4) context.

To build a dynamic heuristic, we need to understand [234] the functionality encapsulated by a **service** to decide if the service is useful for my goal. For that, the service needs to describe its functionality for others to analyze. The SOA community provides semantic **service descriptions** to facilitate the understanding of services [255]. Those descriptions contain ontologies which describe the concepts used, as a formal representation of semantic meaning. If an ontology contains concepts unknown to an agent, the use of the service becomes difficult. Special to the problem of creating a heuristic regarding semantic service description is that we have to handle reasoning with the Open World Assumption [74] and domain specific ontologies.

The difficulty is that the agent needs to integrate new concepts into its beliefs. An example for that could be, an agent providing a service. This service can be atomic and only depend on the agent itself, or the service is composed, meaning it depends on other services to provide its functionality. If the service is composed, it can be composed of services proposed by other agents. Consequently, a service provided by an agent can rely on services of other agents. The use of services of other agents increases the reusability, resilience, and flexibility of those services and creates a loosely coupled SOA like architecture. For my example: If an agent provides the service of booking the cheapest flight available, the service might depend on other airline agents, providing flight schedule and price information. Ensuring that the booked flight is the cheapest available, the agent needs to find all airline agents, and call their services, compare the results and return the cheapest flight. The distributed and dynamic nature of how this is done depends on how well my agent can find new airline services or how well change in the service interface can be integrated.

The precondition for an agent to use the functionality of a service is the ability to **integrate new concepts** into its beliefs. To integrate new concepts into the agent's beliefs, we need to set the new concept into relation with already known concepts. This task is called Ontology Matching [91]. In this work, I try to create an **artificial meaning** representation with the goal of enabling the agent to learn new concepts and use them by reasoning.

Learning new concepts could mean looking up their definitions. This means that we have to compare definitions of words, to find out if the meaning of a concept is known to the agent in form of another concept. Definitions, e.g., in Dictionaries are often given in sentence form. Choosing the right definition, therefore, requires semantically comparing sentences with each other, using a **Semantic Sentence Similarity Measure** (Experiment 3). Such a sentence similarity measure gives us a next experiment on how well my representation of meaning can capture the meaning of sentences.

Used in sentences, words can change their meaning. Ambiguous words carry different meanings in different contexts. Selecting the right word sense leads us to the next experiment: which word sense is the right one to select in a context of use. This leads us to my next experiment for **Word Sense Disambiguation** (Experiment 2).

To be able to use services of agents programmed by other developers, using other models and with that other ontologies, the agent needs to integrate new concepts into its knowledge. The interpretation of new concepts requires the detection if a new concept is equivalent to an already known concept (e.g., by using a semantic similarity measure). This leads us to my experiment with a **Semantic Distance Measure** (Experiment 1). The recognition of similarity is a basic task to form more sophisticated and abstract abilities for reasoning [131, p. 197].

To be able to find relations between concepts, we need a fitting representation of the agent's knowledge describing my representation of meaning (connectionist view) and a reasoning algorithm (symbolic view) to use the representation of meaning. Together those two parts represent my artificial representation of meaning.

All of those sub problems show an aspect of meaning. Each experiment analyzes the capability of the representation of meaning of a different use. With all those experiments we will gather evidence on whether my representation of meaning is a useful one or not. In order to understand which kind of meaning is of interest for us, we look at properties of natural languages and how to correlate to technological developments in computer science.

In linguistics, the science concerned with language, Morris [275] has introduced three components of natural language: *Syntax*, concerning the interpretation of signals, *Semantics*, concerning the meaning and relationship between entities and *Pragmatics*, concerning the interpretation of entities. In computer science, or more precise in formal systems, those components of communications have been incorporated by many theories starting with the syntax of signals [54] leading to Berners-Lee [24] with the vision of a "Semantic Web" and could be extended to a pragmatic reasoning [44]. The third element from linguistics, the pragmatics, can be seen as a context-dependent interpretation of meaning. As noticed by Steel [369] interpretations of statements can become easier if the mutual context is taken into account. Context also might help my agent to integrate new concepts. As syntax is a well-researched scientific area, this work focuses on semantic and contextual information for describing meaning artificially.

For an agent to create pragmatic (context dependent) meaning of concepts, the description of the concepts needs to be context dependent. Bouquet et al. [35] define context as a "local" description of meaning. Since this local knowledge of the domain encodes most of the background knowledge needed by the agent to create a dynamic heuristic this domain knowledge is necessary for the reasoning process [258].

The vision here is to create an ability of agents to self-explain new concepts, which might be a step towards an implicature, which I see as a pragmatic extension of logical implication. Having a formal representation of context-dependent meaning, that is used by artificial reasoners for learning new concepts and comprehending their meaning, can be seen as a step towards AI in agents.

1.2. Problem Statement

To recapitulate the introduction: Agents perform badly in unknown situations [403, p. 3]. I postulate that this is partly because they are unable to integrate new concepts into their knowledge [128, 276]. With that, it seems that modern agents fail to represent the meaning of (new) concepts and bring them in connection with concepts they already know. This task can be broken into two parts: the representation of meaning and the reasoning upon this representation.

Problem Statement: Agents struggle with the meaning of concepts and the reasoning upon it.

The way meaning is represented conditions the reasoning mechanisms an agent can use [273]. Logic-based representations (henceforth I refer to the logic-based representation as *symbolic representation*) have enabled reasoners to make an inference like theorem proving, e.g., with the superposition calculus [83]. Logicians, on the one hand, build upon the idea of symbolic representation where description logics describe languages and the meaning of symbols. Knowledge graphs (henceforth I refer to the knowledge graph-based representation as *connectionist representation*) on the other hand allow inference on structural features, e.g., graph traversal. This neat (symbolic) vs. scruffy (connectionist) discussion is going on for the last 40 years [273].

In AI research a symbolic representation of meaning states that the symbols carry the meaning. In this case, a reasoner has a semantic for the symbols of a language and can use these symbols for inference. One example could be theorem proving in first order calculus [83]. Here examples of such symbols are $\vee$ and $\wedge$, which have a defined semantic and can be used by the reasoner.

Another representation of meaning is connectionist. In connectionist views the meaning of a concept is defined by the concepts, it is in relation with. This kind of meaning representation creates a network of concepts [120] which we can use for reasoning.

The research in both of those factions analyses different things like symbolic reasoning is able to provide consequences form a set of axioms [83] and connectionist using algorithms like PageRank to analyze clusters of topics by connecting concepts [73]. The problem we tackle in this thesis is to combine those two approaches to enable more complex reasoning.

Reasoning is quite abstract and includes multiple tasks. We focus on the reasoning in the form of finding similar concepts, identifying word senses or validating logical correctness, with an influence of connectionist fuzziness in how concepts are seen. The fuzziness is needed because the symbolic reasoning has a fixed semantic for symbols. As soon as another agent uses other symbols for the same concept, connectionist approaches are needed.

The problem with agents today is that without the ability to understand new concepts, it becomes hard to cope with new situations and problems. One example of enabling an agent to react to new situations is to use AI planning to solve previously unsolved problems [134].

1.3. Research Statement

This section formulates the overall thesis of this work and its overarching aim. The goal of this section is to pinpoint a scientific gap, which I will analyze further in the problem analysis in

Section 5. Therefore, I describe my research statement and formulate a hypothesis which I will try to analyze in the remainder of this work.

Given my problem described in Section 1.2 I want my agent to be able to react appropriately in unknown situations. Meaning my agent has to be able to integrate new concepts into its beliefs and reason upon them. By looking at how humans learn new concepts [314, 362, 383], we can formalize the problem by regarding the agents beliefs as a semantic graph, which is a connectionist representation of meaning [273]. Within this work, I will formalize this process and represent the known concepts within a graph structure, henceforth referred to as *semantic graph*. In addition to this semantic graph, I have a second representation of meaning encoded in the reasoning done on the semantic graph. This reasoning is a symbolic view and represents an implicature. New concepts will be included into this graph, leading to the questions (1) where to place them, (2) are there equivalent concepts already present and (3) to which concepts should the new one connect to?

Both representations have their strengths and weaknesses. Symbolic representations can automatically create proofs for facts they have inferred via implications. Connectionist models can represent the fuzzy nature of semantic of natural language. I postulate that the connection of both leads to a better representation of meaning.

To approach the problem described in Section 1.2 I formulated and analyze the following thesis:

Thesis **A combination of symbolic and connectionist representation of meaning can help agents to reason with new concepts.**

As we just described this thesis' objective and the ideas to approach it, now we will ground these ideas and the actual approach in the application domains and the experiments next. The experiments carried out in this thesis can be seen as a collection of evidence that my representation of meaning is sufficient and the agent can use them for reasoning.

By proving evidence for the usefulness of my artificial representation of meaning, I will show evidence of my thesis. From here I can specify questions which I need to be answered to show the usefulness of my artificial representation of meaning. I formulate those research questions in the following section.

1.4. Research Questions

I focus my work on two aspects of meaning, to show the correctness of my thesis: First I look at how to automatically build a connectionist representation of meaning and how it represents meaning in a context-dependent manner. Second, I look at how this representation of meaning can be used with symbolic reasoning to enable an agent to reason on this representation of meaning. With that this thesis will answer the following questions:

Research Question 1 **How can meaning formally be described in a context-dependent manner?**

An answer to this question is provided in Section 5 by introducing a concept for the representation of meaning in agents. This representation is based on the assumption that encoding of information into the knowledge of an agent at design time might not be sufficient to react to new situations. In consequence, we want an agent to be able to update its knowledge during runtime. The representation of meaning thus needs to be able to change with the integration of new concepts. Furthermore, this representation of meaning should be formal enough to enable reasoning in the context of different problems, which leads us to the following question:

Research Question 2 Can the reasoning be improved for the following tasks:

- **Experiment 1** Does my representation of meaning include information to create a state-of-the-art semantic similarity measure?
- **Experiment 2** Does my representation of meaning include information to create a state-of-the-art word sense disambiguation approach?
- **Experiment 3** Does my representation of meaning include information to create a state-of-the-art semantic sentence similarity measure?
- **Experiment 4** Can I improve the performance of service matchers by using the connectionist and symbolic representation of meaning as ontology matching?
- **Experiment 5**

The second research question is concerned with the usefulness of the meaning representation. Having a representation of meaning, I need to show that the information encoded here is of some use in reasoning tasks. Here I select a different task from AI research to compare my approach against the state-of-the-art. Starting with the task of creation of a semantic similarity measure (Experiment 1), I will test this on data sets like the Stanford Rare Word Similarity dataset [244] and SensEval [84]. To compare sentences, we need to be able to compare concepts [104]. Since words in natural language are ambiguous, we need to select the right word sense of the concept used in context (Experiment 2) [282] which is tested on the SensEval Task 3 called “Word-Sense Disambiguation of WordNet Glosses” [267]. Experiment 3 extends the comparison of concepts to the comparison of sentences which we test on the data sets MSRvid, MSRpar, SMTeuroparl like in [308]. To establish a semantic distance between, e.g., two service descriptions, the natural language service description can be compared, like comparing the meaning of sentence [221]. Experiment 4 analyzes the usefulness of semantic similarity for finding appropriate services given a service request, which is tested on the “OWL-S Test Collection” (OWLS-TC) v4[3] which we call S3 test collection. To estimate the usefulness of a service w.r.t. a given goal, e.g., for a heuristic the different meanings of concepts in a goal or in a precondition are of importance (Experiment 5), which we test on the Secure Agent-Based Pervasive Computing (Scallop) domain[4].

In the next section, I will introduce all experiments and their proposed method explicitly. In doing so, I will deduce an overview about the research approach.

1.5. Research Approach

This section describes the overall approach on how I create an artificial representation of meaning and how I want to use it to enable better reasoning for agents.

[3]`http://projects.semwebcentral.org/frs/?group_id=89&release_id=380`, last visited on 27.07.2017

[4]`http://www.dfki.de/scallops/`, last visited on 27.07.2017

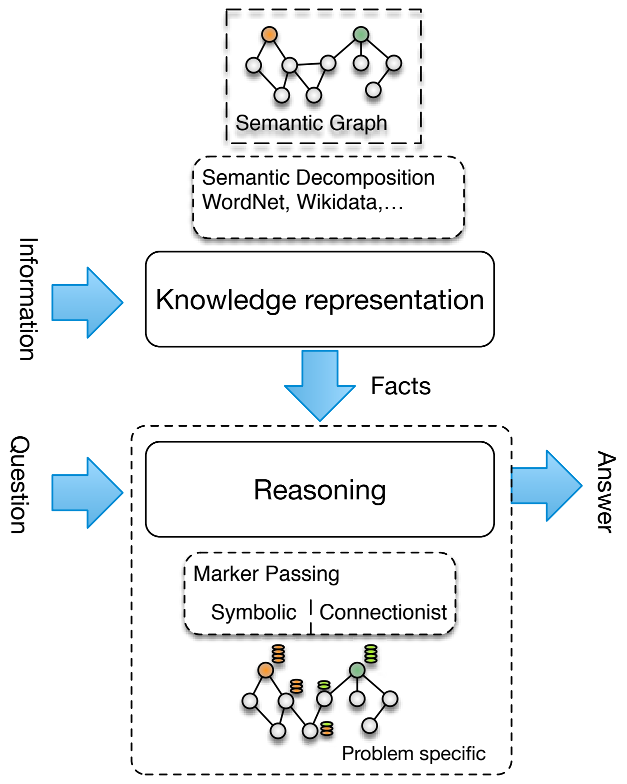


Figure 1.2.: Abstract research approach.

Next, I will describe my research approach by recapitulating my problem statement: An artificial notion of meaning needs to be created for an agent with strong AI to emerge [265]. Without a language and with that the meaning of the words used in this language, an AI is unable to think. Without thought, there is only reacting, no reasoning. AI today can syntactically capture language for many specific problems but never establishes meaning for the words of these languages or can abstract to concepts [44]. Creating an artificial representation of meaning requires the analysis of what meaning is and how it can be formally represented (see research question 1). There are many terms associated with meaning, like semantics, pragmatics, knowledge, understanding or word sense [240]. All of those describe an aspect and bear a multitude of theories explaining what meaning could be. These theories need to be analyzed to develop an artificial notion of meaning best fitted to my current state of knowledge.

We create my artificial representation of meaning in two parts: First the creation of a semantic graph and second the symbolic reasoning using this graph. This abstract approach is depicted in Figure 1.2, which I will describe next.

Knowledge representation

For the first part, we look at different theories of meaning and create an insight into how AI can create something like my understanding of concepts. Since understanding is difficult to measure we use proxy problems to measure if my representation of meaning handles the problem in a way we can understand. These proxy problems are, e.g., guessing a semantic similarity between words via a semantic similarity measure or selecting word senses from one word given a context of use.

So my approach is to create a connectionist knowledge representation as a semantic graph consisting of concepts and their relations, which will serve as a foundation for the representation of meaning [95, 96, 98, 290, 373]. This knowledge representation represented as a semantic graph is based on a lexical decomposition [328]. Here the graph is created by lexical decomposition which breaks each concept semantically down until a set of semantic primes are reached. The primes are taken from the theory of the Natural Semantic Metalanguage [144, 407], which I have analyzed for their usefulness in formal languages [103]. Representing meaning as a graph is a connectionist way of AI, cognition, and linguistic researchers think about meaning [238, 256, 371].

The creation of a notion of artificial thoughts and with that, the representation of meaning is approached by giving an agent the ability to self-explain. This is done by including both connectionist and symbolic meaning: First, I use knowledge sources to create a semantic graph. I build this graph out of different knowledge sources like WordNet, Wiktionary, and BabelNET. Second, I use a Marker Passing algorithm to select relevant concepts out of this graph.

Reasoning

In the second part, I use this knowledge representation to extract facts about the given question and use them to reason upon this knowledge. This is done by an algorithm encoding symbolic information on markers and specifying a set of rules how to move them over the semantic graph so that an interpretation of the resulting marked graph can answer my question about the given problem. I then use this approach to test its applicability in the experiments mentioned in Section 1.4 to answer research question 2. Upon this graph, Marker Passing [48, 99, 165, 167] is used to create the dynamic part of meaning representing thoughts [61][5].

The novelty of this approach is the automatic creation of a semantic graph and the Marker Passing algorithm, where symbolic information is passed along relations from one concept to another, which uses node and edge interpretation to guide its markers. The node and edge interpretation model the symbolic influence of certain concepts.

This doctoral thesis creates a notion of meaning combining the state-of-the-art knowledge of natural meaning with the symbolic and connectionist formalization of meaning for AI. Whether this representation of meaning really formalizes meaning has to be tested. Now we can ask if the here selected experiments really represent problems which need strong AI to be solved. This can be answered by looking at my second experiment which identifies Word Sense Disambiguation (WSD) — the differentiation of meaning of words — as a main problem of language understanding [5]. As an AI-complete problem WSD is a core problem of natural language understanding [179, 419]. Selecting the right word sense in a context of a sentence provides more information then guessing the semantic distance of two words, thus making the creation of a semantic distance measure AI-complete as well. This argument can be extended to sentence similarity and with that to my service matching and heuristic experiment. Therefore, solving such problems lets us evaluate my artificial representation of meaning.

[5]This approach is comparable to the way humans think in that it dynamically reasons about known concepts. Just like the physical brain creates the basis for neural stimuli to activate.

2. Thesis Document

This chapter introduces the structure of the document (*cf.* Section 2.1) and list the contributions (*cf.* Section 2.2) that are provided within this dissertation.

2.1. Thesis Structure

Now that we have a research approach, I can structure scientific endeavor. This section will give an overview of the thesis and explain why the research has been done in this order. Depicted in Figure 2.1 this thesis is structured in four logical parts:

Part 1 – Introduction: The first part introduces the problem and states my research questions, which is placed before this section in **Section 1**. This is done to form a common understanding of the problem and scope of this thesis between reader and author. Easing the navigation of the document, and letting the reader select the parts of interest to him/her, we have the current section which explains the organization of the thesis document in **Section 2**.

Part 2 – Foundation: The foundation establishes the basic terms used in this thesis in **Chapter 3**. The basic terms discussed in Chapter 3 start out with the definition of my fundamental acting entity "Agent" in Section 3.1. Then we look at actions of agents in Section 3.2. I then describe how agents solve problems through "planning" with services in Section 3.3. my analysis continues with the description of the basic building blocks of ontologies a graph in Section 3.4 and the "concepts" in Section 3.5. Next, I describe my representation of connectionist knowledge in form of "Ontologies" in Section 3.6. Since a concept represents meaning, and can be interpreted in different ways, I describe how I will use "semantic" in Section 3.7. Because semantic meaning is independent of context and I want an agent to be able to represent pragmatic (context dependent) meaning I discuss what I will use as context in Section 3.8. Afterwards, I discuss what is meant in this thesis by "meaning" in Section 3.9. In Section 3.10 I introduce the linguistic foundation of my semantic theory called Natural Semantic Metalanguage (NSM).

We will analyze the different state-of-the-art of the different topics relevant to this thesis in **Chapter 4**, starting with semantic theories in Section 4.1. After this I look at formalizations of semantic information in form of different ontologies in Section 4.2. I then have a look at the different semantic description languages in Section 4.3. Afterwards, I will look at the state-of-the-art in the two main parts of my approach: the semantic decomposition in Section 4.4 and Marker Passing in Section 4.5. Because my use case on problem solving is planning, I analyze the related work on AI planning in Section 4.6.

Part 3 – Approach: The approach is separated into four parts: A description of the abstract approach will be provided in **Chapter 5**. This approach foresees that a semantic decom-

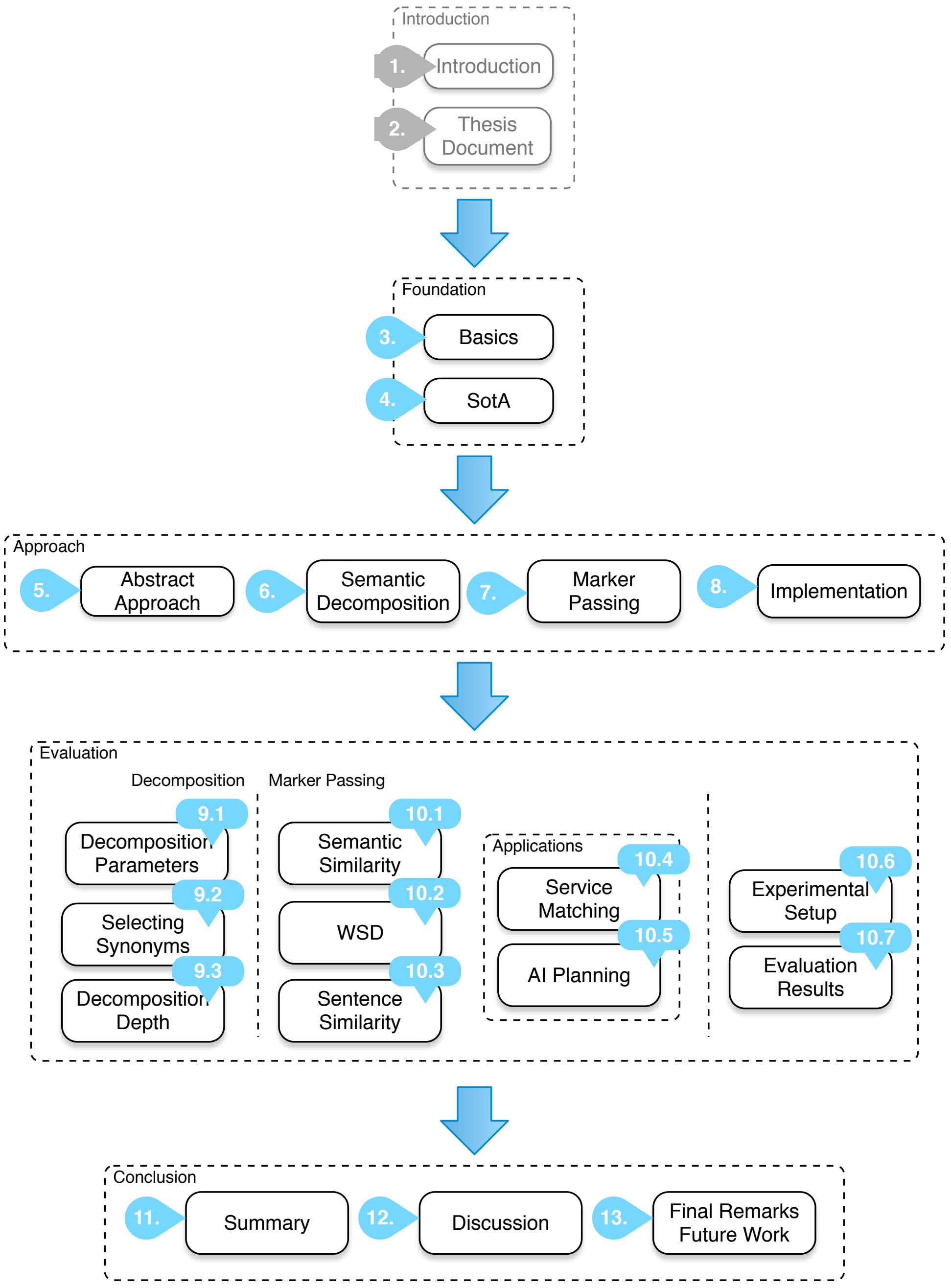


Figure 2.1.: The structure of this thesis.

position creates a graph of concepts which I will describe in detail in **Chapter 6**. Based on this, my approach uses the created semantic graph as basis for a Marker Passing algorithm which is introduced in **Chapter 7**. The approach ends with the description of the implementation of the decomposition and the Marker Passing algorithm in **Chapter 8**

Part 4 – Evaluation: The evaluation consists of six experiments which are described in two parts: **Chapter 9** the experiments regarding the Decomposition and **Chapter 10** regarding the Marker Passing. The first experiments (see Section 9.1 to 9.3) analyzes the meta parameters of my approach. The following experiments use my approach in different scientific challenges: a semantic similarity measure (Section 10.1), Word Sense Disambiguation (Section 10.2) or a Sentence Similarity Measure (Section 10.3). The experiments are followed by two experimental applications of my approach: The first experimental application is a Semantic Service Matchmaking described in Section 10.4, the second experimental application is a dynamic heuristic for service planning described in Section 10.5.

Part 5 – Conclusion: The conclusion summarizes the approach in **Chapter 11**, discusses the experiments with respect to the research questions in **Chapter 12** and provides a critical reviews of my approach and an outlook into possible future work in **Chapter 13**.

2.2. Contributions

We can structure my scientific output by cutting the different parts into contributions. These contributions will be discussed next. The main contribution of creating a theoretic sound representation of meaning can be split up into smaller contributions. To show the novelty of my approach, I have surveyed technical semantic representation and analyzed their viability for my domain. Creating the needed artificial representation of meaning presented hurdles, for instance, the acquisition of new facts had to be automated. The main contributions of this dissertation provide an answer to the research questions formulated in Section 1.4 and are listed in the following:

Contribution 1. *A formal representation of context-dependent meaning consisting of a semantic graph and a Marker Passing algorithm.*

Contribution 2. *A semantic decomposition component that automatically creates a semantic graph.*

Contribution 3. *A Marker Passing component that uses symbolic and connectionist information.*

Contribution 4. *An evaluation of my approach by comparison with the state-of-the-art based on six different problems.*

We base my contributions on the analyses of the applicability of NSM as a pragmatic metamodel for the utilization in service descriptions. Here I will outline an approach, able to create NSM-based context-dependent explanations which can be used by artificial reasoners to search on it. One outcome is a context-dependent description of meaning which is my Contribution 1.

This work will compare different reasoning algorithms and their extension to cope with the newly introduced concepts, eventually introducing a new approach to the creation of a semantic decomposition algorithm in Contribution 2 and a reasoning algorithm which uses the so created semantic graph and builds Contribution 3. These two algorithms combined are compared to the state-of-the-art in six AI challenges forming Contribution 4. Given these contributions, I can integrate this thesis into the bigger picture of AI by comparing it w.r.t. existing contributions.

In conclusion, the following contributions can be drawn from this doctoral thesis: A new meaning representation combining connectionism and symbolism is defined and automated in its creation. I extend the known Marker Passing algorithms and analyze their use for reasoning. This analysis gives insight into the notion of meaning and how it can be formalized to be useful in AI research. I have published parts of this research [93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104].

Part II.

Foundations

3. Basic Terms and Concepts

To build a common ground on the terminology used, we will now look at the basic terms and concepts used in our scientific strive. For that this chapter introduces relevant concepts from the fields of AI agents (Section 3.1), their actions and the relation to services (Section 3.2) and planning (Section 3.3), formal descriptions of graphs (Section 3.4), the definition of concepts (Section 3.5) and ontologies (Section 3.6) as well as semantic representations (Section 3.7), a notion of context (Section 3.8), our notion of meaning (Section 3.9) and finally a short introduction to the Natural Semantic Metalanguage (Section 3.10). We will start out by looking at agents and AI Planning to understand the basic algorithms to compose actions into a plan and with that understand why there are still problems in finding problem solutions through planning. We then look at the formal basis for our approach: Graphs and Ontologies. We use ontologies to encode semantic information. We, having ontologies as a basis, then discuss our understanding of semantic information, meaning, and context. The mixture of semantics and context brings us to an understanding of pragmatics.

We start with the definition of an agent since the fundamentally acting entity in this work is called agent.

3.1. Agents

Picking up the idea of Weiss [403, p. 3], agents are an approach to create more autonomous and intelligent software. The notion of an agent and the design paradigm of Multi-Agent Software Architectures [416] are well discussed in the related work. This section defines what we will use as a definition of an agent, which part of the agent will be further analyzed and how the agents and services are interlinked to enable the transfer of semantic approaches to the agent world.

We concentrate on the deliberation part of the agent which selects a service[1] executed next. This mechanism of an agent selects a service based on the information available to the agent which could be, e.g., the goal the agent wants to achieve. This mechanism is called **planning** or sometimes *means end reasoning*; meaning that the agent reasons about what service to perform next and its effect or its precondition, before executing it. Planning is a hard task for most humans and as the quote leading into this section of Weiss [403, p. 3] signifies: computer are even worse at it. Pinpointing which part of the agent we are concerned with, we need to look at some general definitions of agents and determine in which part of the agent the "reasoning" is happening, being able to improve it.

Hayes-Roth [160] formulates the most fitting abstract definition of an agent. Here an agent acts continuously by:

Sense: Perception of the environment updating the knowledge of the agent.

[1]In the Agent community the services provided by agents are called actions. We use the word service to emphasize the SOA part of an agent, where the action proposed by an agent can be seen as service.

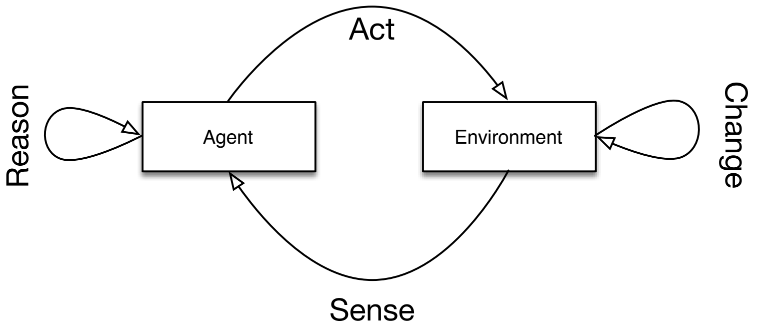


Figure 3.1.: Illustration of an agent in its environment.

Act: Acting as taking effect on the environment by executing services.

Reason: Reason with available information, e.g., the knowledge of the agent.

Figure 3.1 shows an abstract agent and its interplay with the environment. This definition of agent is too broad and needs to be narrowed to grasp the aspects of agents used in this work. Since this work focuses on the reasoning part of the agent, we further focus on architectures of agents concerned with the internal design of an agent, e.g., its service selection behavior. We see the reasoning as the mechanism which describes the inherent ability of the agent to change his mental state. With a certain complexity of the reasoning of an agent, its ability to represent meaning or act intelligently gets hard to prove. Because of this, we test such reasoning mechanisms through observation of the behavior of the agent [384].

We start out by separating natural from artificial agents. Artificial agents are agents which are man made, thus do not occur in nature with out the influence of higher intelligence like human beings. For example Cars, Robots and Software might fall into this category. Furthermore, since we are talking about software components, we reduce our definition of Agent to the virtual part of artificial agents. As a result, we restrict our notion of an agent to the definition of Wooldridge and Jennings [417] who define an artificial “weak” software agent in more details by requesting the following properties:

Autonomy: the agent can operate without interaction from some external or deterministic source and can control his services and internal state.

Social ability: the agent can interact with other agents.

Reactivity: the agent can perceive the environment and adapt to change.

Pro-activeness: the agent has, in addition to reacting to the change in the environment, initiative or goal-driven behavior.

We want an agent to have those properties since **autonomy** allows the agent to control its beliefs, **social behavior** allows the calling of other agents services, **reactivity** allows the agent to sense new concepts and reacts on service calls and finally **pro-activity**, so that the agent is able to plan and show goal driven behavior. We will especially postulate the social and pro-active properties of agents. On the one hand, we need agents to act together so that we can plan including all of their services to achieve a given goal and on the other hand we need pro-active behavior on the reasoning about the services descriptions of other agents.

With this definition, we can distinguish between software agents and software components like services which are not pro-active and autonomous. Looking at the variety of software which has been implemented using an agent paradigm, there are many different architectures which establish the properties of an agent differently [15]. However, all of them have in common, that the agent needs to interact with the environment through sensing and acting. We thus can still base our definition of an agent on Figure 3.1 to illustrate the notion of an agent.

Wooldridge and Jennings [417] further define a "strong notion of agency" as software which has the properties of the weak notion of an agent and also "...is either conceptualized or implemented by using concepts that are more usually applied to humans" [417, p. 117] like thinking which again is similar to most AI principles, where human behavior is simulated. The strong notion of agency is too abstract because being implemented with concepts to describe human thinking introduces too many attributes, like a psychiatric disorder.

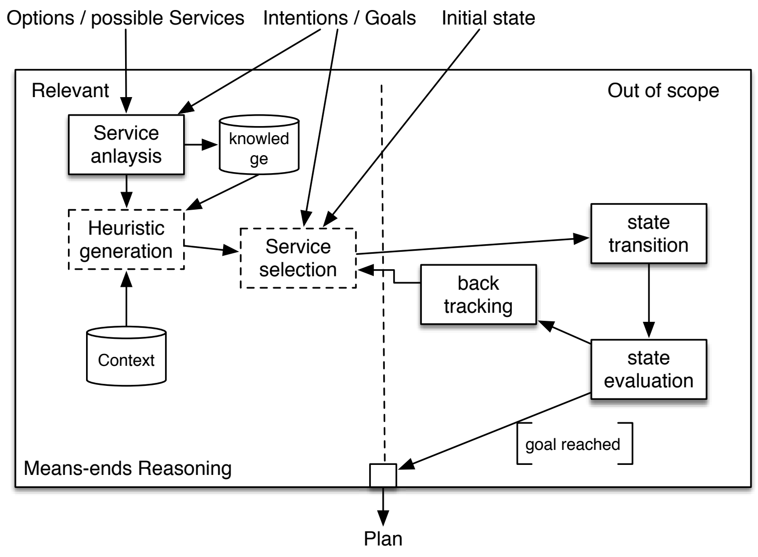


Figure 3.2.: The rough architecture of the means-ends reasoning of an agent.

In addition to the requirements of Wooldridge and Jennings [417] we want an agent to be intelligent, meaning to be able to find solutions for given problems on its own. With that autonomy is extended from the ability to make decisions on its own (autonomous, e.g., with a fixed set of decisions possibilities) and to create own decision possibilities (intelligent). Especially intelligence enables an agent to solve problems in a goal-directed way. Intelligent behavior merges all the properties declared by Wooldridge and Jennings [417] in a way, that the adaption of the agent — in a proactive or reactive way — adapts towards a given goal of the agent. Consequently, an agent should be able to adapt existing plans or create new ones if necessary, leading us to the planning problem which we see as one problem-solving method. Consequently, in this work, we try to make software agents more intelligent by giving them the ability to plan with a semantic and pragmatic understanding of service descriptions as service representations. For that, the representation of meaning in an artificial agent is subject to research, which leads us to the analysis of formal information representations in the next section.

Weiss [403] identifies two general architectures for software agents: Belief-Desire-Intention (BDI) architecture [320] and Multi-Layer Agent which are discussed in Weiss [403] with the example of *TuringMachines* [111] and *InteRRaP* [277]. There are many agent architectures [15], but since our approach is independent of the agent implementation, as long as it includes a means-ends reasoning with some kind of planning we do not pick an agent architecture here.

Since this means-ends reasoning [309] is in focus in this work, we will further embed our approach in a cognitive architecture discussed next.

We postulate that the means-ends reasoning part of an agent is the part we want to investigate if we want to create intelligent agents because this is the part where the actions of the agent are planned. The actions of an agent define its behavior and with that our perception of its intelligence. The means-ends reasoning part is abstracted in Figure 3.2 where we show the relevant parts for this work. We focus our analysis of an agent to the heuristic generation (dotted line in Figure 3.2) and the service selection as a use case to evaluate our approach.

Figure 3.2 describes the scope of our analysis. So there is no plan execution to evaluate the plan. From the agent perspective, this can be seen as thinking about the problem, making an abstract plan which could solve the problem, but not execute it. We focus on the pure planning, because of the execution of a plan, with all the hurdles, are too much to handle in this work. The challenges arising during execution are subject to other research like describe in the work of Ghallab [135] and Nau [281].

To outline this work, one could say that the proposed thesis extends the work of U*són and Faber* [387] with the goal to create a metalanguage to describe context-dependent meaning automatically as artificial thoughts. In the work of Castelfranchi [46] this ability of abstraction and conceptualization is called *Cognitive Emergence*.

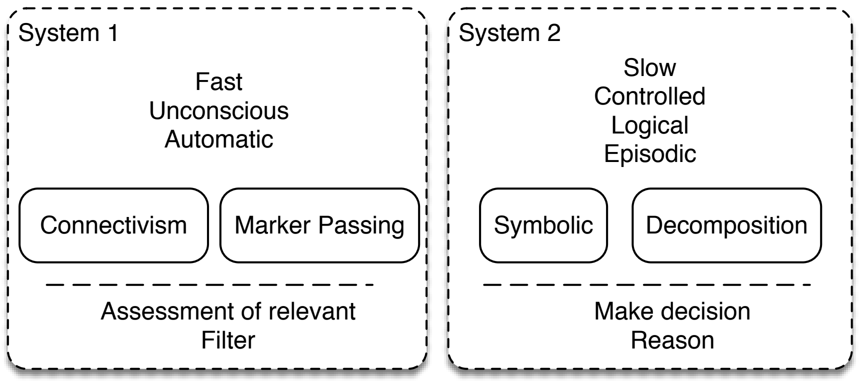


Figure 3.3.: Cognitive interpretation in an agent.

In the theory of Kahneman [186], he separates the cognition into two parts called System 1 and System 2. One is responsible for the fast, automatic evaluation and filtering of new information; the other one is in charge of the controlled, logical decision making. The ability to self-explain new information to oneself is an essence of learning. The decomposition allows us to create the semantic network used by the Marker Passing to, e.g., find a semantic distance between wanted concepts (goal) and provided once (services). To interpret this in a cognitive manner, we take the insight of cognitive decision making from Kahneman [186] and extend it to an artificial agent. Figure 3.3 illustrates the separation of an agents cognition into System 1 and System 2.

The agent as the basic acting component of our approach is part of a distributed system called Multi-Agent System (MAS) [242, 372, 403]. Here different functionalities are provided by different agents. The functionality of an agent is encapsulated in its actions. The next section will discuss the functionalities of agents and their use.

Now we can formulate our definition of an agent formally based on the "Abstract Architecture of Intelligent Agents" of Wooldridge [416]: An agent A as a tuple $A = (Beliefs, Goals, Services)$ of its beliefs (the knowledge of the agent), its goals (which formalize what the agent wants), and the services it provides for others.

3.2. Actions, Services and Capabilities

In this section we will describe how we see a *service* and its relations to terms like *action* or *capability*. We do so to connect the search done in the SOA research with the Multi-Agent research, showing parallel developments and map the terms used to our terminology in this work.

There are many names for the fundamental active part, which we call agent: the execution, the actor, a component or a subsystem. In the agent community, the activities of agents are often called actions. In the SOA community, an equivalent is called services. A service encapsulates functionality and specifies an interface for using this functionality. SOA do this to ensure separation of concern and reusability. The reusability is supported by describing the service regarding its input and output parameters, preconditions which need to be fulfilled to call a service and the resulting effect [255]. MAS separate functionality by agents and the interfaces to the functionality are its actions. Here we refer to an agents actions as service because we are interested in the semantic description normally present for semantic service. Additionally, we can argue that the functionality of an action can always be encapsulated in a service, which is just an interpretation of how we look at the functionality provided by an agent [204]. It does not matter to us, who provides the service, or how intelligent or adaptive it is or how it is implemented, as long as we can use its functionality as part of our search for a solution to a given problem. Sabatucci et al. [339] provide a definition of service, with the addition, that a service evolves a set of world states S, from a state $s^i \in S$ to a state $s^{i+1} \in S$. For each state, the effect of the executed service has to hold in s^{i+1}, and the precondition needs to be fulfilled in s^i for the service to be executed.

Sabatucci et al. [339] define their world states as closed worlds [74], which is not sufficient for the use of semantic technology from the Semantic Web [24] because there the closed world assumption does not hold [74]. Further, we differentiate between world-altering service and information services like Saboohi et al. [340]. The difference is that that world-altering services describe effects which differentiate s^{i+1} from s^i. Information services do not. Again the definition of Sabatucci et al. [339] is not sufficient here, since they demand s^{i+1} to be different from s^i. A world state is further defined in Definition 1:

Definition 1. ***A State*** *A state S is a set of facts which make up the state.*

Here a fact is a quantum (a measurable unit) which can be concrete or abstract and formalizes one aspect of the world. In either way, facts can be seen as abstracted entities in a context. A context is defined in Section 3.8. The state can represent a world state, like used in most planning approaches [339] or can be an abstract representation of a fictional world, which is kept in a belief of an agent, as an intermediate step during planning.

The main difference to the classical objective state of planning problems, like defined in [134], is that we do not claim a closed world.

The formal representation is based on a set of facts F which are technical represented in an ontology using the Web Ontology Language (OWL) [103]. We discuss more details on ontologies in Section 3.6.

The facts $f \in F$ which make up the state S^i are subjective to the agent i believing fact f. The facts making up S^i are subjective[2], since from another perspective or in other contexts, facts might differ in their evaluation. This means that not only the selection of facts describing a state but also their manifestation may differ from agent to agent.

Let S be the set of states of the environment and let $Services = s_1 \dots s_n$ be the set of services of an agent. We argue that we are not interested in the service implementation itself but rather in the functionality it provides. Thus, we define a service with its service description. Martin et al. define a service s_i as a sextuple in the manner of an OWL-S description [255]. We adopt this definition as follows:

Definition 2. *A **service** s_i := (Input, Output, Precondition, Effect, Name, Description).*

The Input, Output, Precondition, and Effect (IOPE) are equivalent to the profile of an OWL-S Service profile [255]. Here the input describes the information needs of the agent to perform a service, and the output describes the change done by the service. The precondition and effect, on the other hand, describes the state change the service might invoke. The name and the description of a service natural language description of the functionality the service encapsulates. Since, this so-called "profile" of a service, describes *what it can do* we will have a closer look at it, to determine if this description of a service can help the agent to think about the service and to decide to execute the service, or not. The name of a service is the natural language name of the service, e.g., ”BookFlightService”. The description is a natural language description of the service thought for humans.

Here the input describing the required information, objects (individuals), all parameters of the service needs to be executed.

Definition 3. *Input parameters are a set Input of facts Input* $= \{i_1 \dots i_n\}$.

These facts[3] making up the inputs can be abstract in the form of free variables or concrete as in grounded statements on individuals[4]. All input parameters are a subclass of the more general class parameter. Listing 3.1 illustrates the OWL-S definition of a parameter.

Listing 3.1: Definition of the OWL parameter class.

```
<owl:Class rdf:ID="Parameter">
 <rdfs:subClassOf>
   <owl:Restriction>
     <owl:onProperty rdf:resource="#parameterType" />
     <owl:minCardinality
         rdf:datatype="&xsd;#nonNegativeInteger">1</owl:minCardinality>
   </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Parameter">
  <rdfs:subClassOf rdf:resource="&swrl;#Variable"/>
```

[2]This means the facts depend on the beliefs of the agent.
[3]The inputs describes restricted facts which we reduce to class assertions.
[4]Individuals are further discussed in Section 3.6.

```
</owl:Class>

<owl:Class rdf:ID="Input">
  <rdfs:subClassOf rdf:resource="#Parameter"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="parameterType">
  <rdfs:domain rdf:resource="#Parameter"/>
  <rdfs:range rdf:resource="&xsd;anyURI"/>
</owl:DatatypeProperty>
```

Listing 3.1 describes the parameter class, which has a parameter type with a cardinality of one. This parameter is a variable and the input. The parameter type is given (here with the example “anyURI”), which describes that the “Variable” is of this parameter type. The “parameter type” is defined as “DatatypeProperty” which describes a domain and a range. The domain and range defined which type of facts are connected with this “parameterType” relation.

The output is the optional return values of a service. It is similarly defined to the input parameters:

Definition 4. *Output parameters are a set Output of facts Output* $= \{o_1 \dots o_n\}$.

Like for the input parameters the output parameters are defined as a subclass of OWL parameters as shown in Listing 3.2.

Listing 3.2: Definition of the OWL-S output parameter class.

```
<owl:Class rdf:ID="Output">
  <rdfs:subClassOf rdf:resource="#Parameter"/>
</owl:Class>
```

Listing 3.2 shows the change from an input to an output parameter. The rest of the description would be similar to Listing 3.1.

As an example of a service description, the interested reader can look at one example service from the Semantic Service Selection (S3) Contest[5].

Definition 5. *Preconditions are a set Precondition of SWRL rules Precondition* $= \{p_1 \dots p_n\}$.

OWL-S describes the precondition and effect in the Semantic Web Rule Language (SWRL). These rules are made of the form: **Body** $\rightarrow$ **Head**, which means that if the body evaluates to true the head is executed. The precondition describes a set of conditions which have to be fulfilled (in the current state) for the service to be executable. Such a condition can be formulated as a rule [175] without a head. The semantic here is that if the body of the rule holds, then the service can be executed. Therefore we define preconditions with an empty head, because we want nothing to be changed by the rule in our ontology[6].

Similar to the preconditions of a service, the effect of a service describes which facts are fulfilled after the execution of the service.

Definition 6. *Effects are a set Effect of SWRL rules Effect* $= \{e_1 \dots e_n\}$.

[5]see: `http://www-ags.dfki.uni-sb.de/~klusch/s3/` last visited: 22.07.2017

[6]Because the SWRL specification states that if a rule has an empty head, the rule is only true if all facts in the body of the rule are evaluated to false, we fill the head of a precondition with a fictional fact, which states something like the rule is fulfilled.

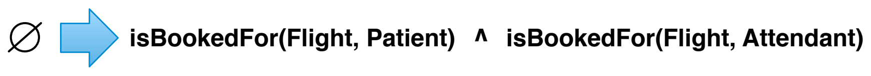

∅
isBookedFor(Flight, Patient) ∧ isBookedFor(Flight, Attendant)

Figure 3.4.: Abstract example effect rule.

The difference to a precondition is that an effect has an empty body. The body of the effect is empty because, if the service is executed, we want the effect to be executed as well. Here the empty body is interpreted as always to be true.

The effect formulates the expected effect of the execution of the service on the environment. Figure 3.4 describes an example effect. In this effect, we formulate that the effect of the executed service is that the flight *#Flight* is booked for a person called *#Patient* and a person called *#Attendant*.

The name of a service is a natural language name for the service, which in most cases give us a clue on which topic the service is. This is not a formal description of the service since the developer can choose names like “Service1” for a service. In these cases, the name of the service does not reveal much about the functionality of the service, but in other cases, where the service is for example called “BookFlightService”, it gives us a brought domain in which the service acts.

The description of a service is a natural language description, thought for humans, which describes the functionality encapsulated in the service.

Definition 7. ***Description*** *the description is a natural language description of the functionality of the service. It might include examples on the result of the service or example parameters.*

The name of a service can hint us on its domain or functionality.

Definition 8. ***Name*** *The name is a natural language identifier. A string with no spaces naming the service.*

Now we have defined the acting the part of an agent. Next, we will look at its knowledge representation by defining how we see its beliefs:

Definition 9. *A **Belief** is a set of non-contradicting facts which the agent evaluates to be true.*

We are especially interested in the knowledge base of an agent since we want to model meaning by combining symbolic and connectionist information. Updates in the knowledge base are interpreted as new states. We represent the belief of an agent as ontology. What kind of ontology we use is further discussed in Section 3.6

Definition 10. *A **Goal** is a set of non-contradicting facts representing what an agent wants to achive.*

The fulfillment of the goal is checked for each fact and if all facts are fulfilled the goal is reached. Such a definition of a goal represents a set of goal states. Depending on how abstract the goal is formulated the facts of the goal could be fulfilled in different states.

Because sometimes the achievement of a goal cannot be done by the execution of one service but needs the combination of multiple services, AI planning analyses methods to combine services to achieve a goal.

3.3. Planning

This section will describe planning as a problem-solving method. Since this work will not engage in defining a new planning algorithm, we will only look at state space planning to motivate which information is available for the creation of a heuristic.

Planning represents a problem-solving approach; where we think about how to solve a problem by determining which services we can use to solve the problem. By ordering the available service and choosing the right input parameters, planning tries to find a combination of services to reach a goal state.

A planning problem is formalized in the following way based on the description of a state transition system given in [134, p. 17]:

Definition 11. *A planning problem P = (S, A, γ, s_{start}, s_{goal}) with,*

S = $\{s_1, s_2, \dots\}$ *a finite or recursively enumerable set of states,*

A = $\{a_1, a_2, \dots\}$ *a finite or recursively enumerable set of services,*

γ**:** $S \times A \rightarrow S$ *a state transition function,*

s_{start}**:** *an initial state, and*

s_{goal}**:** *a goal state.*

The goal here, since it is a state, is a set of facts. It encodes all states in which these facts are true. We can imagine this planning problem as a directed graph, where the services determine which state is connected to which state by the state transition function. We changed the definition from [134] to fit the terminology here. Since actions of agents are services to us, we call the actions here services as well.

For the use of services of other agents, the agents need to describe their services to each other, so that they can evaluate if the service might be useful to them. To do that the agent needs, on the one hand, a formalization of meaning in its beliefs, goals, and functionality needs and on the other a semantic description of the services. Both are described by using semantic representations of meaning like ontologies.

3.4. Graphs

This section defines attributed typed graphs with node type inheritance. We will use this formalism as a basis for representing semantic networks in Section 6 and 7. The definitions in this section are taken from De Lara et al [66]. We start with the definition of a graph, consisting of sets of nodes and edges and a source and target function.

Definition 12 (Graph (Definition 1 in [66]))**.** *A graph $G = (V, E, s, t)$ with a set V of nodes and a set E of edges and a source and target function $s, t : E \rightarrow V$*

The nodes represent the vertices of the graph and the edges represent the lines between the nodes. The two functions *s* and *t* give the edges a direction by assigning a node as target and source to each edge. Giving two graphs a graph morphism maps the nodes and edges of one graph onto another graph by preserving the structures of the first graph. Like in De Lara et al. [66] we define a e-graph morphism in the following way:

Definition 13 (E-Graph and E-Graph Morphism (Definition 6 in [66]))**.** *An E-Graph G with $G = (V_G, V_D, E_G, E_{NA}, E_{EA}, (source_j, target_j)_{j\in\{G,NA,EA\}})$ consists of sets:*

- *V_G and V_D called graph and data nodes (or vertices) respectively;*
- *E_G, E_{NA}, E_{EA} called graph, node attribute and edge attribute edges respectively.*

and source and target functions

- *$source_G : E_G \to V_G, target_G : E_G \to V_G$ for graph edges;*
- *$source_{NA} : E_{NA} \to V_G, target_{NA} : E_{NA} \to V_D$ for node attribute edges;*
- *$source_{EA} : E_{EA} \to E_G, target_{EA} : E_{EA} \to V_D$ for edge attribute edges.*

$$\begin{tikzcd}
E_G \arrow[rr, bend left, "source_G"] \arrow[rr, bend right, "target_G"'] & & V_G \\
E_{EA} \arrow[u, "source_{EA}"] \arrow[r, "target_{EA}"] & V_D & E_{NA} \arrow[u, "source_{NA}"'] \arrow[l, "target_{NA}"']
\end{tikzcd}$$

Figure 3.5.: Illustration of the EGraph morphisms taken from [66].

Let $G^k = (V_G^k, V_D^k, E_G^k, E_{NA}^k, E_{EA}^k, (source_j^k, target_j^k)_{j\in\{G,NA,EA\}})$ for $k = 1, 2$ be two E-graphs. An E-Graph morphism $f : G^1 \to G^2$ is a tuple $(f_{V_G}, f_{V_D}, f_{E_G}, f_{E_{NA}}, f_{E_{EA}})$ with $f_{V_i} : V_i^1 \to V_i^2$ and $f_{E_j} : E_j^1 \to E_j^2$ for $i \in \{G, D\}, j \in \{G, NA, EA\}$ such that f commutes with all source and target functions, e.g. $f_{V_G} \circ source_G^1 = source_G^2 \circ f_{E_G}$

The E-Graph and E-Graph-Morphism allow us to define two types of nodes: graph nodes V_G^k and data nodes V_D^k. Further, we distinguish between three types of edges: the graph edges E_G^k, the node attribute edges E_{NA}^k, and the edge attribute edges E_{EA}^k. For those edges source and target, functions are defined respectively. This extension of a basic graph is needed to allow attribute for nodes and edges. The graph edges describe the edges of a graph, the node attribute edges map attributes to nodes, and the edge attribute edges map attributes to edges respectively.

Definition 14 (Attributed Graph and Attributed Graph Morphism (Definition 7 in [66]))**.** *Let $DSIG = (S_D, OP_D)$ be a data signature with attribute value sorts $S'_D \subseteq S_D$. An attributed graph $AG = (G, D)$ consists of an E-graph G together with a DSIG-Algebra D such that $\biguplus_{s\in S'_D} D_S = V_D$. For two attributed graphs $AG^i = (G^i, D^i)$ with $i = 1, 2$ an attributed graph morphism $f : AG^1 \to AG^2$ is a pair $f = (f_G, f_D)$ with an E-graph morphism $f_G : G^1 \to G^2$ and an algebra homomorphism $f_D : D^1 \to D^2$ such that (1) commutes for all $s \in S'_D$.*

With the mapping of attributes to nodes and edges, Definition 14 extends a graph to an attributed graph and a graph morphism to an attributed graph morphism, where the morphism maps in addition to nodes and edges the attributes of the graph as well. The difference to a attributed graph to a e-graph is here that is uses a data signature (sorts and operations) for

$$\begin{tikzcd}
D_S^1 \arrow[r, "f_{D,s}"] \arrow[d, hook] \arrow[dr, phantom, "(1)"] & D_S^2 \arrow[d, hook] \\
V_D^1 \arrow[r, "f_{G,V_D}"] & V_D^2
\end{tikzcd}$$

Figure 3.6.: Illustration of the AGraph diagram taken from [66].

attributes, where e-graphs just use an anonymous set of data nodes. Next, we define in Definition 15 how the type graph is connected to a graph so that we can create a typed attributed graph.

Definition 15 (Typed Attributed Graph and Typed Attributed Graph Morphism (Definition 8 in [66]))**.** *Given a data signature DSIG, an attributed type graph is an attributed graph* $ATG = (TG, Z)$*, where Z is the final DSIG-algebra.*

A typed attributed graph (AG, t) *over ATG consists of an attributed graph AG together with an attributed graph morphism* $t : AG \to ATG$*.*

A typed attributed graph morphism $f : (AG^1, t^1) \to (AG^2, t^2)$ *is an attributed graph morphism* $f : AG^1 \to AG^2$ *such that* $t^2 \circ f = t^1$*.*

This type graph which maps type to node and edges of the graph does yet not support inheritance. Definition 16 describes an attributed type graph with inheritance.

Definition 16 (Attributed Type Graph with Inheritance (Definition 9 in [66]))**.** *An attributed type graph with inheritance* $ATGI = (TG, Z, I, A)$ *consists of an attributed type graph* $ATG = (TG, Z)$*, where TG is an E-Graph* $TG = (TG_{V_G}, TG_{V_D}, TG_{E_G}, TG_{E_{NA}}, TG_{E_{EA}}, source_i, target_i)_{i \in \{G,NA,EA\}})$ *with* $TG_{V_D} = S'_D$ *and Z the final DSIG-algebra and an inheritance graph* $I = (I_V, I_E, s, t)$*, with* $I_V = TG_{V_G}$*, and a set* $A \subseteq I_V$*, called abstract nodes.*

The inheritance in the attributed type graph in Definition 16 allows us to describe a type graph by using inheritance edges between nodes and with that have type inheritance for the typing of nodes in a typed attributed graph.

Among others, we want to use this formalization to describe our connectionist representation of meaning. This representation, called ontology, has concepts as nodes and relations between concepts as edges. Next, we will look at how the basic building block of an ontology (a concept) is defined.

3.5. Concept

A single entity with its ontology is sometimes called a concept. We will use this term to specify a word with all its relations like its synonym relations or its definitions. To this end, we define:

This signature describes the signature of what we see as a concept. Here we define all the functions which a concept implements. For example the *getDefinitions* function as input a string representing the word we want to look up definitions for and a word type[7] and returns a list

[7]A word type is a syntactical classification of words like verbs or nouns. The word type is sometimes called Part-of-Speech (POS).

Signature *Concept*
sorts:
String
WordType
Definition
opns:
$getDefinitions : Concept \times String \times WordType \rightarrow Definition*$
$getSynonyms : Concept \times String \times WordType \rightarrow Concept*$
$getAntonyms : Concept \times String \times WordType \rightarrow Concept*$
$getHypernyms : Concept \times String \times WordType \rightarrow Concept*$
$getHyponyms : Concept \times String \times WordType \rightarrow Concept*$
$getMeronyms : Concept \times String \times WordType \rightarrow Concept*$
$getLiteral : Concept \rightarrow String$
$getWordType : Concept \rightarrow String$

of definitions which was found for this concept with the given word type. All other functions except the getLiteral work similar to the getDefinitions function. The *getLiteral* function returns the string representation of the word representing the concept in natural language.

In addition, we can get the literal[8] of the concept by invoking $getLiteral(c)$ for concept c. The word type or Part-of-Speech (POS) of a concept is made available by the *getWordType* function.

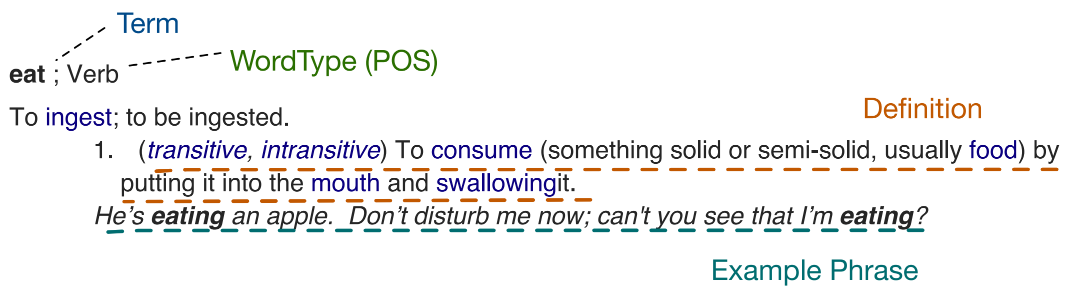
We define the signature of a definition as follows:

Signature *Definition*
sorts:
String
Concept
Definition
opns:
$getDefinition : Definition \rightarrow Concept*$
$getExamplePhrase : Definition \times Concept \rightarrow Concept*$
$getSenseKey : Definition \times Concept \rightarrow String$
$getTerm : Definition \times Concept \rightarrow String$

Here the definition of a concept can be collected by calling *getDefinition* which then returns a list of concepts consisting of those concepts making up the definition. Of course, a concept can have multiple definitions. A definition contains an example phrase as well. This example phrase shows the definiendum in a typical use in a sentence. The function *getTerm* returns the definiendum of this definition. If available, the *getSenseKey* function returns a sense key compatible with WordNet. This is done to compare the different definitions from different dictionaries. Here an example definition shown in Figure 3.7 from Wiktionary (`https://en.wiktionary.org/wiki/eat`) for the word “eat”:

We continue with the definition of ontology since this is the essence of the formalization of our connectionist representation of meaning.

[8]String representation of the word representing this concept.



Term
eat ; Verb
WordType (POS)
To ingest; to be ingested.
Definition
1. (transitive, intransitive) To consume (something solid or semi-solid, usually food) by putting it into the mouth and swallowingit.
He's eating an apple. Don't disturb me now; can't you see that I'm eating?
Example Phrase

Figure 3.7.: Overview of the state-of-the-art of semantic similarity measures.

3.6. Ontology

The goal of this section is to define what we will use as an ontology for the representation of our semantic graph, modeling the connectionist part of meaning [366]. In the remainder of this work, we will use ontologies to model, e.g., the beliefs of agents, the states — like the goal state of our planning problem — and describe the capabilities the agent can use to create a plan which might reach that goal state. Furthermore, we want to reason upon those ontologies (e.g., to calculate a semantic distance of words and sentences) and extend them automatically. Having set this purpose of ontologies, we will have to create a theoretic sound basis for the formalized meaning, complete with the benefits of automated reasoning. For that purpose the definition of Euzenat and Shavaiko [91] is adapted to our definition with the Definition 26 and 27 of Mahr.

We start out with the most basic element: the concept. We define a concept in the way of Mahr [249] defines an entity: A concept describes something that exists, and anything that exist can be described by a concept. Ontology then describes a concept and its relationship to other concepts or itself. In Figure 3.8 we define six types of concepts: four of them are relations. The other two are "Type" and "Individual". Types specialize concepts which can be instantiated. The individuals are realizations of abstract concepts.

Formally we defined an ontology similar to Euzenat and Shavaiko [91] but now we should even consider the mind set of the Model of conception from Mahr [249] as shown in Figure 3.8. Figure 3.8 describes that everything is a concept. Especially relations are concepts. We have specified three special relationships:

Named Relation a relationships with a name like a meronym or "FatherOf".

Instantiation a relationship which relates concepts to individuals.

Assignment a relationship which assigns concrete individuals to the sources or targets of a relationship.

This combines Euzenat and Shavaiko [91] with the idea of Mahr [249] since relations are concepts as well and because everything is a concept (Mahr calls this entity). We need to notice here, that because we are unable to describe cardinality in this formalism, relations reference zero or more concepts. The special relationships for instantiation and assignment are needed to distinguish between individuals and concepts. Here we extend the definition of Euzenat and Shavaiko [91] since we define what an individual consists of concrete values for the abstract concepts related to the concept under study.

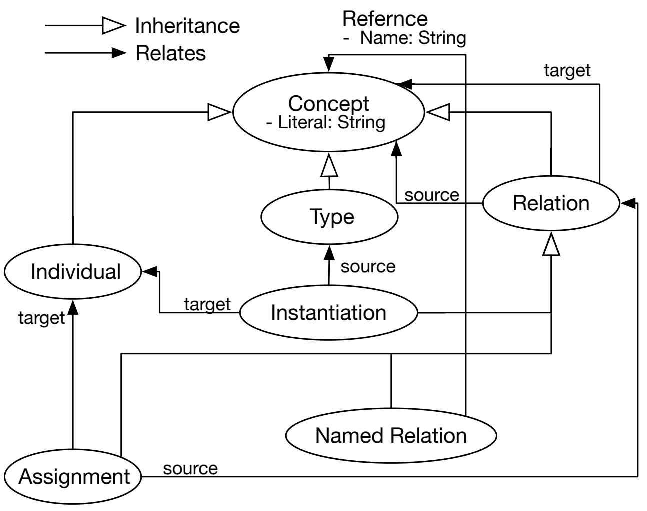


Figure 3.8.: Type graph of our ontology definition.

3.7. Semantic

Semantics, in general, is the theory studying the meaning of symbols [240]. For computer science, on the other hand, semantics can be understood as an enrichment of data with meta-information, for example, to describe the meaning of words used in a programming language. Here the meaning of the word of the programming language is defined by the calculation a computer will do when seeing that word. For natural language, this meta-information can be formalized in an ontology represented in abstract conceptualization. For more details, please see Section 3.6.

Semantics for formal languages can be seen as an investigation of the interpretation of the formal statements making up something formulated in this language like an ontology. Meaning can be used in a wide range, whenever a reasoner tries to understand a relationship of entities or, e.g., action might have a meaning as well. However, this work focuses on the meaning of concepts. In this regard semantic is a linguistic area concerned with the meaning of words or utterances. We define semantic meaning as in compliance with [45, 240]:

Definition 17. *Semantic is the context independent meaning of concepts.*

To establish our notion of meaning, we analyze different semantic theories in Section 4.1 to gain insight into how meaning is formalized, how it can be used and which inference is possible with each of these theories. For that, we want a formalization of symbolic and connectionist meaning which can represent meaning in an expressiveness of natural language and still is tractable for the reasoning done upon it as well as for us working with it. This means semantic information is the information a reasoner can extract from an utterance without knowledge extending the syntax. For example the sentence “Ich bin ein Berliner” (engl. I am a Berliner[9]) can be analyzed in the following way: There is a speaker (here the denoted by the “I”) who sees himself as part of the group with the name “Berliner”. The German grammar additionally allows the conclusion that the speaker is a male because of the use of “ein”. This completes the semantic analysis. The meaning for which this sentence has become famous: “We as America (Capi-

[9]An utterance made by U.S. President John F. Kennedy on June 26, 1963

talists) stand beside you (Germany) in the fight against the UdSSR (Communist), and we will support you (also capitalists)," is only extractable with a vast amount of contextual information.

The problem with the contextual information is that the uttering agent and the listening agent[10] might have different contexts like depicted in Figure 4.1 on Page 48. In linguistics, the context of the uttering agent is called "Äußerungskontext" (context at utterance) [240]. The semantic of the utterance itself is called "Ausdrucksbedeutung" (meaning of the utterance) which is the meaning of the utterance with the addition of relevant facts form the context at utterance [240, p. 11]. The pragmatic (context depending meaning) can then be seen to build up on the semantic meaning in the "Kommunikativer Sinn" (the wanted effect of the communication of the speech act) [240]. The receiving part from the view of the uttering agent is the effect the uttering agent wanted to create in the listening agents.

The semantic information does not have this problem because it describes the meaning of the concepts used in the utterance in general. Thus, it is not context dependent. The problem here is that now we need to describe the meaning of every concept we encounter. The question here is: Can we create a semantic where we do not have to describe the semantic of each word, but derive it. The semantic part of our artificial representation of meaning thus needs to be wisely chosen to be able to represent these three layers (Semantic, Syntax, and Pragmatic) of meaning. Following Tarski [377] a sufficiently expressive language can define its semantic in itself. Leading to the question of how expressive our language has to be, to be used in the artificial representation of meaning.

3.8. Context

Context has many definitions and is often used to describe all information available. One of the basics for defining context is the work of Dey [71]. Dey uses "relevance" of information to "describe a situation of an entity" as criteria for the definition of context. This is a vague definition since it lets us not distinguish between other information since all relevant information becomes contextual information. Here it becomes important to describe what relevance means, which is a other research question [385].

The context in Natural Language Understanding is often interpreted as the words surrounding a target word or in distributed models (with word embedding) the concurrence of a word with another word in texts in a given window [39]. The size of the context then is subject to discussion [270].

Miller [270] defined a linguistic context as a set of sounding words in which a concept can be used. This is sometimes referred to as Topical Context [270]. Here the vocabulary used to discuss a topic provides additional information to the creation of meaning.

In this work we use the *Model of Conception* of Mahr [250] as a formal representation of context. This section is meant for a short introduction of the notion of context used in this work so that we can understand the problem statement in Section 1.2. For further details Section 3.8 describes the notion of context in more detail.

To illustrate how we will use context in this work we will now look at an example of contextual information in our use case: A taxi calling service which takes two locations as input, the starting address, and the destination address.

[10]Here listening agent is not restricted to audio signal but means the receiver of an abstract message.

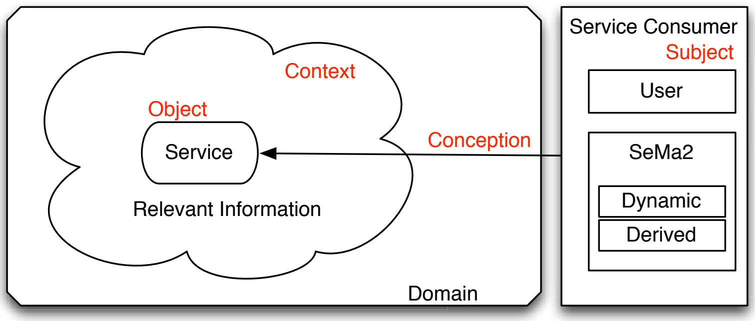


Figure 3.9.: Illustration of a context in regard to an object using the example of a service and its consumers.

For service planning an action selection component is needed. This action selection component selects the best fitting service for a given request. Depending on the contextual information given, let's say the location of the user, different services can be selected. As shown in Figure 3.9 the service is conceived by a subject — in this case, the Service consumer or an service matcher called SeMa2 — and the relevant information of this service the location of the user makes up the context. Figure 3.9 illustrates the transferred model of conception as one definition of context from Mahr [250] by using an example of a service.

Definition 18. ***Conception***

1. *A Conception is a relationship consisting of three entities:*
   - *An entity called subject of the conception*
   - *An entity called object of the conception*
   - *A complex called the context of the conception*
2. *The content of a conception is a complex consisting of the relationships of the context, of which the object of the conception is an element of.*

This can be imagined as an observer (the subject) observing an entity (the content) which is made up of the object and the context surrounding this object (all entities which stand about the object). The conception, therefore, is something like a perspective and depending on the subject the object might appear in a different context. If an entity (or for our purpose a concept) is part of different conceptions, the context regarding the subject might change. In consequence, one object can be part of multiple contexts. An example of such a construction is the meaning of a word, from the view of a speaker and a listener. An effect of different contexts for the word could be a misunderstanding. For structure with multiple conceptions Mahr [250] has defined the term: **Universe**

Definition 19. ***Universe:***

3. *A universe is a complex to which with every entity which belongs to the universe, a conception belongs to it as well, which has this entity as content.*
4. *A universe is called reflexive, if it belongs to itself.*

A universe, therefore, is everything which is part of a conception. In our example the following three components are part of the universe: the word uttered by the speaker, the speaker and the listener and the context of all three. From these definitions we can extract the definition of a context as follows:

Definition 20. *A **context** is a subset C of complexes.*

C *is the set of complex with* $C \subset E$ *where* E *is the set of entities.*

A *is the set of conception (german: "Auffassung") with* $A \subset R$ *where* R *is the set of relations.*

$\subseteq$ *is a relation over* $(E \times E) \cup (R \times R))$ *denoting if an entity is part of the relationship or entity called 'belongs to'.*

As argued above, semantic concepts as part of a description of a service can be extended to reflect the changes in context. This could, for example, happen if the viewpoint of the subject changes. In a connectionist view point, the meaning of an entity is defined by its relationships to other entities leading to the conclusion that the context of a conception defines the meaning of its object. This context-dependent meaning can be seen like the pragmatic extension of semantics in linguistics. Coleman et al. [57] notice that within communication between agents which are not context aware, the meaning cannot extend beyond the meaning explicitly carried by the message. Thus, the meaning needs to be predefined or explained within the message, which forms the crux of this approach. Following the argumentation of Robertson [330] adaptive software requires knowledge of the context as in: "what does the application/capability do?", how to react/adapt to change and runtime support for detecting change as well as modifying parameters to adapt to that change. On the one hand contextual meaning can be seen to be twofold: First, we can define abstract domain specific meaning, e.g., if we are talking about opening something in the smart home domain we talk about windows and doors. Second, during runtime, the abstract values have to be filled with concrete contextual information, e.g., this sensor measures if door A is open. With context-aware communicating agents, context-dependent meaning might change over time, since the context may change. Figure 3.9 illustrates the transferred model of conception as one definition of context from Mahr [250] by using an example of a service.

3.9. Meaning

Since the main goal of this doctoral thesis is to formalize meaning of an AI to use, this section analyzes the different definition of meaning. The goal of this section is too close the knowledge gap about meaning. Then we are able to answer our first research questions so that the approach to represent meaning selected in this thesis can be understood.

From the point of view of Minsky [273], we will use two aspects of meaning in this doctoral thesis:

Definition 21. *Connectionist meaning of a concept (definiendum) is given thought its definitions and other concepts it stands in relation with.*

Based on the Meaning Text Theory [262] the Definition 21 of the scope of meaning we interpret conceptual meaning as context-dependent since the definiendum might have relations to contextual concepts. This can be seen in an example like the concept "grandma" which in my context has the relation “reference to” connecting to my two grandmas and since the reader most likely does not know my grandmothers, it will reference other persons in the readers context. Even though we might have the same definition of a grandmother in our semantic representation, they are used differently. This part of meaning is represented by our second part of meaning:

Definition 22. *Symbolic meaning of a concept (definiendum) is given thought the use of the connectionist representation of meaning with a pragmatic reasoning algorithm.*

This second part of our artificial representation of meaning described in Definition 22 represents the symbolic view on meaning. Looking at our “grandma” example, even though we have the same definition of grandmother representing our concept of grandma, the way we think about it differs, e.g., with the contextual concepts activated regarding the grandma concept like cake or elderly. Depending on those contextual concepts our meaning of “grandma” changes. This change might be from happy situations with coffee to sad situations in hospitals.

The remainder of this section is spent discussing why we have selected our definition of meaning like described in Definition 21 and 22 and a concludes our view of meaning.

There are many definitions of meaning. This section states our view of meaning and discusses different views meaning. The goal of this section is to grasp the notion of meaning formally. We will see that not one single definition of meaning fits and that we need to switch from measuring “the right meaning” to e.g. “the same meaning” or “a different meaning”. To identify the definition of meaning; we look at the study of language because we are interested in the meaning of words and text not images or events. Meaning is often seen as something virtual, like an “idea” or a “thought” rather concrete objects which are referenced by an utterance [297]. Since meaning seems not to be an inherent property of things, the meaning is formed by observing things and within the head of the observer. This makes meaning a subjective matter which might be subject to change since we do not stop observing and with each observation, there might by new information which lets us change our preformed meaning about something. The definition of the term “meaning” needs further investigation into the different fields of research which meaning is subject of. In this section, we will look at different definitions of meaning to crystallize how we define meaning of words by comparing them.

The philosophical meaning has been discussed in a multitude of publications [136]. Some of them discuss the location of meaning from being inside ones head to being part of nature in the objects which are referenced by concepts [124] or as part of the communication act [414]. For Wittgenstein, the use of language as well as the use in the specific speech act itself represents the meaning. Grice [151] separates the “conventional meaning” from the meaning intended by the speaker.

Labov [217, p 342] noticed that the linguistic endeavors into meaning lead to a classification into categories, which are “... discrete, invariant, qualitatively distinct, conjunctively defined, composed of atomic primes.” Rieger [327, pp. 161] discusses this as a classical view of modeling. He differentiates between internal semantics (called a theory of meaning) and external semantics (called a theory of reference).

Theory of meaning states that expressions or symbols carry their meaning. There are two kinds of theory of meaning: *semantic theories of meaning* which explain the meaning of a symbol and a *foundational theory of meaning* which tries to explain what meaning is [422].

Theory of reference states that the meaning of words lies in the object (called the reference) which it points out in the world.

This type of theory is further discussed in the remainder of this section.

Since the meaning in both theories, seems to be connected to cognitive elements — carrying the meaning or holding the reference — we have to define the basis for such cognition: the **mind**: Without launching into a philosophical discussion this work is based on a theory of mind which is called monism. In that theory of mind — the most consistent ontological thesis at the beginning of the 21*th* century — the physicalism is, where, in opposite to the dualism, the mind is based on physical events [352]. With that, we can define the use of mind in this thesis as: A mind is every cognitive (mental) process of thought like reasoning, remembering or consciousness based on the physical stimulus of the body containing the mind.

With the physicalism as the basic theory of mind, meaning as part of thought has to be based on physical events. Hence meaning can be seen to be built up with the cognition we endure during our life. Majumdar et al. [251] call this "prelinguistic knowledge." Based on this assumption we analyze different theories of meaning and how they fit in an artificial application in AI.

Meanings of meaning

In this section, we will try to describe what meaning is about after the role model Ogden and Richards [291]. Furthermore, how it is used in AI and we will try to find a definition which is formal enough for this thesis to work with.

First of all the meaning is a vaguely defined concept. There are many words enclosing that we know as common sense meaning:

relevance a measure of how pertinent, connected, or applicable something is.

distinction two or more things being different from one another.

intension any property or quality connoted by a word, phrase, or another symbol.

consequence a result or effect, importance or relevance.

reason the power of the mind to think, understand, and form judgments logically

signify a symbol of an indication of something

deduction the action of deducting or subtracting something

implication the conclusion that can be drawn from something although it is not explicitly stated

purpose the reason for which something is done or created or for which something exists

substance the essential nature underlying phenomena

definition a statement of the exact meaning of a word, especially in a dictionary

sense a way in which an expression or a situation can be interpreted

These are just a few of them[11]. These words all collectively describe what we mean by questions like: "What is the meaning of ...?" [335, p. 21].

[11] Definitions of the words were selected from the Oxford Online Dictionary.

It seems peculiar that in philosophy the meaning of a sentence depends on the words used in the sentence and on the other hand, the meaning of a word depends on the sentence (context) it is used in [370].

Looking at the effect of meaning, we might notice that the speaker of an utterance and its listener have a common understanding of the utterance if they have a common interpretation of the utterance [335, p. 24]. Grasping this idea in a theory of meaning, the meaning is made up of the knowledge, beliefs, and experience or other cognition which is used by both speaker and listener to interpret the utterance. Therefore, the listener can interpret the utterance of the speaker, and with that extract the intended meaning of the speaker, if they use similar interpretation.

Scientifically, e.g., from linguistics, philosophy or other structural sciences, there are many different definitions of meaning, starting from semantics (context independent meaning) to pragmatics (context-dependent meaning). As our example from the short introduction to semantics 3.7 shows, semantics is a rather formal process, which lets us extract the information of words which is part of the common ground. For example, the utterance “I am a Berliner” shows a self-reference, namely a speaker (imaginary or real, alive or dead) had to excise, and the symbol “I” stands for the reference (the speaker). This *distinguishes* the speaker from someone else since he did not use the word *we*. There are further *implications*, or *consequences* to this utterance: since the speaker references himself he seems to be self-aware. For humans[12] this means the speaker is probably older than 30 months, since this is the age human children might need to learn to use the self-reference “I” instead of their names.

However, all this does not come close enough the questions: What did Kennedy mean with his sentence “Ich bin ein Berliner.”? The *sense* of connectedness, such a statement might give the people of Berlin in those times of war. This sentence was said with the *purpose* of reinsuring western Germany that they have an ally, support and especially financial assistance. That they are not left alone, and there is hope that change will come. On the other hand, it states the policy regarding the Soviet Union and with that against communism. Further the age of the speaker might not be *relevant* here.

All this can be *reasoned* with contextual knowledge, explaining what this visit of the American president to West Berlin might *signify* to the inhabitants of West Berlin. If this has been the *intension* of Kennedy is likely but not certain since we did not ask him to explain his actions but used *deduction* on his state of mind, thought reasoning on what he said.

We can see that the different concepts describing meaning are all entangled in the explanation above, and we need to formalize them. Further, we can see that the right meaning of an utterance often lets us reduce the vast amount of information which could be deduced from it. We stop the search for explanations once we think we have extracted the fitting meaning to the context.

Defining the meaning for concepts with the result of having a lexicon to look up concepts and have their explanation given by the description in the lexicon is one aspect of meaning [13]. Here the definition of the meaning of a word is dogmatic, an axiom for most concepts, since the concept itself, does often not introduce the meaning it carries. Especially, to learn a language, certain concepts need to be learned by heart to be able to understand the language. In our lexicon, this is the case. Here the meaning is defined [235, p. 157].

[12]That the speaker is human is context information, and should be ignored here, but for the sake of the argument and since humans are the only English speaking species we know of, we can argue that we talk about semantics for humans here.

[13]This is based on the meaning text theory [262], which states in abstracted sense that meaning of a concept can be explained using words.

This is different with dynamic compositions of concepts (e.g. sentences or larger texts). Here the meaning of the composition comes from the way the concepts are composed. There is an infinite amount of compositions which could be built. As a result, those compositions build a lexicon to look up the meaning of such text (composites of words).

Deyne et al. [65] describe a semantic network representation of the knowledge representation (called a mental lexicon). They try to analyze the structure of the mental lexicon to explain psychological phenomena like developmental shifts [280]. Collins et al. [58] argue that a network representation is psychologically plausible as a representation of a mental lexicon. Dowty [76, 77] on the other hand uses Montague grammars to describe a Decomposition of meaning.

From now on we can differentiate the meaning of words and the meaning of sentences. The composition principle implies that the meaning of sentences, phrases, and larger texts can be composed out of the components it is built on and the way they are composed (the grammar) [235, p. 157]. We will convene on the composition principle and with that analyze at first the meaning of concepts and have a glance at the composition to sentences and their meaning.

In the next sections, we analyze different definitions of meaning to their purpose of formalization for the use in AI.

Linguistics

In linguistics, the definition of meaning is often pushed aside to other research fields as Bloomfield [30] noticed 1933: Meaning of an utterance for linguists is, therefore "The situation in which the speaker utters it and the response which it calls forth in the hearer." [297].

The remainder of this chapter discusses context-dependent meaning with the goal of defining the difference between semantics and pragmatics for machine learning, knowledge representation, and artificial reasoning. The last few decades of research in computer science have been dedicated to the analysis of semantics as context-independent meaning. In this research effort terms like the Semantic Web [24] have been coined and we want to look at this achievements and work on a definition of meaning for AI.

As lexical context, we use a sentence which utilizes the concept in question c_1. This is chosen because a sentence has a distinct meaning and stands as one unit of meaning. Here a sentence encapsulates the formulation of one fact. If more than one fact is described in a sentence, then the sentence is broken down into multiple sentences, where each one represents one fact. Here in a lexical context the concept c_1 is only used in one sense. The lexical context could be extended to larger linguistic units if the concept c_1 is not used in two different senses. Since that would lead to the need for identifying sub-context for each sense of c_1 to identify the different senses.

From this philosophical discussion, we take away, that there is no sufficient formal definition of meaning to be used in our AI research. The formal definition of meaning is out of scope for this doctoral thesis. That is the way we try to find a way measuring meaning without a proper definition. Instead of defining the meaning of a concept, we look at the difference of meaning amongst concepts. The next discussions will try to analyze: **When is the meaning of two concepts the same?**, **When is the meaning of two concepts the opposite?**, **When is the meaning of a concept more or less abstract?**, **When is the meaning of a concept part of another concept?** and finally **How is the meaning of a concept explained?** We start out by looking at words which have supposed to have the same meaning: Synonyms.

Synonyms

Two concepts which have the same meaning are called synonyms. A closer look on what "the same meaning" means leads us to the different definitions of synonyms, later in this section. The problem of having to define synonyms comes from the imprecise nature of natural language. Since one concept might have different representations in a language, and different representations might reference the same concept we have a function defining which concepts reference the same entity and vice versa. Let's call this function synonym.

Since we define meaning to be two fold (connectionist and symbolic), and there is "true meaning" we use the notion of synonymy to measure the correctness of our definition of meaning, or more precise our artificial representation of meaning, by selecting synonyms of concept, sentence, and semantic service descriptions. This is why we need to define in this section what we mean by meaning and with that by synonymy.

One definition of a synonym is often traced back to Leibniz which states: "two expressions are synonymous if the substitution of one for the other never changes the truth value of a sentence in which the substitution is made" [271]. A concept consequently seems to be a synonym to itself. However, other than that, it will be hard to find concepts which "never changes the truth value of a sentence."

Löbner [240, p. 117] defines a synonym as: "two expressions are synonymous if and only if they have the same meaning." With having no precise definition of meaning this definition is too fuzzy for us to use. Löbner further elaborates that two concepts with a denotational equivalent, hence referencing the same entities do not have to be synonyms. For instance, in German, the word for "Weihnachtsengel" (Engl. Christmas angel) and the "geflügelte Jahresendpuppe" (Engl. winged doll for the end of the year) reference the same entities but are not synonyms, because the synonyms need conceptual equivalence.

Linke et al. [235, p. 167] define synonyms in the compositional semantic (see section 4.1.3) as concepts with the same semantic attributes.

Linke's definition of a synonym is restrictive because only equivalents can be substituted. Miller et al. [271] relax this definition by making the equivalence of a concept relative to the context of use: "two expressions are synonymous in a linguistic context C if the substitution of one for the other in C does not alter the truth value."

This definition still has synonyms in a true or false state, meaning that a concept c_1 is a synonym to a concept c_2 or not. This is a problem because with that the synonyms are synonyms in all contexts or in none. Thus, there are only a view words which are synonyms, not regarding their word sense. We extend this definition to a fuzzy one. Further the definition of Miller et al. [271] does refer to an undefined "linguistic context" which we cannot refer to. We rather use our definition of context formalized in Definition 20. So that we use the definition of a synonym as follows.

Definition 23. *Synonym(c_1,c_2): Concept $\times$ Concept $\rightarrow$ [0,1]:*

$$Synonym(c_1,c_2) = \frac{\sum\limits_{c \in C} synonym_{Miller}(c_1,c_2,c)}{|C_{c_1}| + |C_{c_2}|}$$

Where C is the set of all contexts and C_{c_i} is the set of all context including concept c_i. With $synonym_{Miller}(c_1,c_2,c)$ being the definition of Miller et al. [271] where two concepts are synonyms if there exists a context where the two concepts are substitutable. Definition 23 state that two concepts are more synonymous if they can be replaced in more contexts. This makes the synonym relation understandable since we can use our distance measure to look at all contexts in which a concept is used, and then create a discrete synonym measure.

To grasp a more rigorous definition of $synonym_{Miller}(c_1, c_2, c)$ we define $synonym_{Miller}(c_1, c_2, c)$ in Definition 24 with a semantic distance d:

Definition 24.

$$synonym_{Miller}(c_1, c_2, c) = \begin{cases} 1 & \text{, } if\ d(c_1, c_2, c) = 0 \\ 0 & \text{, } else \end{cases}.$$

Where $d(c_1, c_2, c)$ is a context-dependent semantic distance measure. Two concepts are then synonymous when there is a context in which they can be replaced.

Accordingly, we have a true synonym if concept c_1 and concept c_2 can be exchanged in all contexts they are used in (Synonym(c_1,c_2) = 1) and we are having conceptual equivalence, in fact, fulfilling the definition of Löbner [240, p. 117].

Synonyms of concepts are the first step towards the synonymy of phrases. As in natural language, most utterances can be expressed in different sentences (sometimes called paraphrasing). There has to be a synonym relation between sentences as well. Since this work analyses semantic relations and leaves out the syntactic analysis, we define the problem of sentence synonymy as out of scope for this work. Nevertheless, the results of this work can be seen as the first step of a connectionist representation of semantic sentence equivalence, mixed with the symbolic representation of syntactical information. Besides, we believe that the representation introduced in this work might be able to represent sentence synonymy, but leave the proof to our future work.

Antonyms

An antonym relation states an opposite relation like: "The antonym of a word x is sometimes not-x." [271]. Miller et al. state with their example of rich and poor of antonyms that rich is the opposite of poor but not rich does not imply poor [271].

In the same way, concepts can be in a synonym relationship with others they can be in antonym relationship with each other. Like in synonymy there are no "true" antonyms as one can see in the example of good being not bad. The example sentence "This example is not bad." does not mean that this example is a good one.

To state that antonyms are semantically distant as an opposite of synonyms poses problems as well since "Elephant" and "build" seem semantically distant, but they are not considered antonyms. With that, one could ask the questions of what the antonym of a chair is. The list of answers would be massive: since words like "the," "lonely" or "like" seem all alike not to be a chair.

However, there seems to be a difference in word type in relation to antonym. For example, consider that "hot" and "cold" are semantically closer to each other than "hot" and "which" and they are labeled as antonyms. Table 3.1 shows the different antonym types.

Further discussions on antonyms for artificial reasoning can be found for instance in Heim [162] and Novak [287].

For our purpose, we define a threshold similar to what we have done for synonyms but on the other side of the semantic spectrum.

Name	Description
Gradable:	hot vs. cold
Complementary:	off vs. on
Convers Relational:	seller vs. buyer
Revers relational:	adding vs. subtracting
Incompatible:	dog vs. banana

Table 3.1.: List of different types of antonyms.

Definition 25.

$$\textit{antonym}(c_1, c_2, c) = \begin{cases} 1 & \textit{, if the meaning of c using } c_1 \textit{ is different of c using } c_2 \\ 0 & \textit{, else} \end{cases}.$$

Here the difference means any kind of antonymy relation like shown in Table 3.1. We can notice that our definition of antonymy is compatible with our definition of synonymy since as soon as two concepts are not true synonyms, they start to be antonyms as well. Two concepts are true antonyms if concept c_1 and concept c_2 can be exchanged in any contexts they are used in and the meaning of the context c is different (Antonym(c_1,c_2) = 1). This allows us to interpret every two concepts c_1 and c_2 as antonyms if they always replace each other as a semantic opposite in a context, or as incompatible.

For all other antonyms, we have to depend on the word type and the semantic distance. As stated by Miller et al. [271] an antonym is sometimes seen as a lexical relation between word types and not as semantic relations. In our example, most humans would agree that “open” and “closed” are antonyms but it takes longer to realize that “open” and “the” or “and” are incompatible with the grammatical structure of the sentence, and the truth value of this new context cannot be evaluated. Hence I argue that an antonym c_2 needs to replace a concept c_1 and negate the meaning of the sentence as in: “The door is open.” and “The door is closed.” If this is not the case, we cannot talk about an antonym. With that, we have the definition of a true antonym.

More precisely, the truth of a fact represented by a lexical context *c* can be checked against to beliefs of the agents. Meaning that with the example of Tarski [378] context “Snow is white.” the fact that snow has a white color can be checked in the ontology which represents the beliefs of the agent and lead to a truth value. Then in this example, if “black” is an antonym of “white” the sentence: “Snow is black.” can be checked. If this fact is false in the belief of the agent, then we have collected evidence that “black” and “white” might be antonyms. Even with the open world assumption, we can check truth values with a third option stating “we do not know”. With examples for and against certain uses of the concepts in question, we can build our understanding of antonyms. This effect can influence our antonymy relation in both ways. If abstracted, and white is compared to “However”, then “black” can be seen closer to “white” then “However”, e.g., because they fall both in the category of “colors.”

Hypernyms and Hyponyms

A hypernym relation of concept c_1 with a concept c_2 is that concept c_2 that is a generalization of c_1. For hyponyms the opposite, a specialization is true. We now discuss the properties of the generalization, but the same might be argued for a specialization. This means c_2 is more general, more abstract and can be obtained through the introduction. Meaning can be generated from different aspects, like the observation of something specific a general rule, common properties, a super ordinate, governing law, conceptualization, a broader meaning, and an abstraction or a conception.

Most of these hypernyms leave at least one aspect of the specialization unspecified. Therefore, it is possible to choose between the different forms for this aspect. These hypernyms especially mean something similar than their specialization but are fuzzier or cover more realizations. They stand in contrast to the hyponym relations. This means that not all facts about a specialization are true about the hypernym as well. However, on the other hand, all facts about the hypernym are true for all specialization. Let's look at a little example: Let's suppose a "dachshund" is a specialization of a "dog, " and surely dog is a hypernym of a dachshund. Furthermore, let's define that dogs have four legs. This means dachshunds have four legs, too. Now let's describe the physic of a dachshund as "sausage-shaped". However, we are not able to say that all dogs are sausage-shaped. This is a special attribute of dachshunds.

This means that in most contexts a concept can be replaced with its hypernym, without changing the sentence truth value. E.g. "this dachshund is sausage-shaped" will become "this dog is sausage-shaped" which is still true if the dog is a dachshund. Of course, this can be done transitively. Let's suppose dogs are carnivores. Then the sentence "this carnivore is sausage-shaped" is equally true.

This means that with hypernyms/hyponyms we can create generalizations about a group of sub-concepts or entities, without having to specify it for all those entities. In ontological terms, the hypernym and hyponym relations are mostly combined into a single relation referred to as "is-a" relation. Meaning that a dachshund is a dog.

Meronyms and Holonyms

A meronym relation of concept c_1 with a concept c_2 states that c_2 is part of concept c_1. The inverse relation to a meronym is the holonym relation. Starting with Winston [413, 415] meronyms have been analyzed and classified in different types. Priss [311] brings those relation types into a formal construct. Here the relation types, e.g., collection, group, ingredient, and organization, are set into relation with formal attributes like inclusion, parting, possession, and property. Priss defines the formal context of meronyms as Triple: (G, M, I) where G is the set of objects, M the set of attributes and I is a binary relation $I(g, m)$ with $g \in G$ and $m \in M$ which states: "the object g has the attribute m". Priss continues by analyzing the relation of sub- and super-concepts and the effect on meronyms.

Watrouse-deVersterre et al. [400] separates these relations into the "has-a" (meronym) and "part-of" (holonym) relation. This means with the meronym/holonym relation we can describe a sense of whole-to-part relationship in either direction. Since in some semantics concepts are perceived to be more similar the more properties they have in common [232], meronyms are relevant to the formalization of meaning.

Explanations

The meaning of word or the concepts it stands for can be given by explanations. In the case of an explanation being provided with other words, we have to assume the Meaning-Text Theory [262] which states that language consists of a mapping between meaning and text. This mapping can be given by an explanation. Several definitions of explanations have been proposed. Each one specialized for the needs of some domain. To start with, in statistics we can identify evidence weights in a Bayesian believe network as explanations [161]. These weights represent the logarithmic likelihood ratio of the influence of an observation on a specific variable. Therefore, they can be and indeed are used to explain in which way the occurrence of an event influences the current systems state [79]. To ease the access of humans to these statistical explanations, different classes of techniques can be applied (e.g., verbal explanations [88] and graphical explanations [56]). Beyond, Druzdzel [79] identified two categories in which such explanations can be separated: *Explanation of Assumptions* focusing on the communication of the domain model of the system and *Explanation of Reasoning* focusing on how conclusions are made from those assumptions. It might be worthwhile to transfer these categories to self-explanation[14] since the meaning of concepts used might differ depending on the exogenous or endogenous origin of the fact explained. Therefore, the reasoner has to distinguish between the explanations of the system itself and how it can be interpreted related to the current context. This work focuses on the explanation of assumptions since the audience of such a description is seen as an external system component. As those approaches are quite fundamental and thus general, we further want to list some more practical approaches on explaining capabilities from the agent community:

- Braubach et al. [38] uses the beliefs, desires, and intents to formulate goals, knowledge, and capabilities for a multi-agent system,
- Grüninger et al. [153] uses First-order Logic Ontology for Web Services (FLOWS) to describe the functionalities of a service,
- Sycara et al. [373] formulates agent and service capabilities utilizing the Input, Output, Precondition and Effect (IOPE) approach,
- Martin et al. [255] uses the Ontology Web Language to structure the description of services.

The above-listed approaches are used to describe services and with that are used to create explanations. They can be seen as having some level of self-explanation [93]. Taking this into account, we can clarify that self-explanation enables the integration of new agents autonomously into the existing infrastructure [193, 279]. Following the idea of self-explanation this means that new agents, as well as existing ones, are able to learn the capabilities of each other and to comprehend in which way they can interact (e.g., which data format and concepts match). The explanation of new concepts then implies a new problem of interpreting the concepts in a new context.

Conclusion

In conclusion, we can say that it is hard to define meaning and that many different ways of defining meaning have different purposes. However, as we have seen, we can define similarity of meaning. Defining similarity of meaning or more precise synonyms of words is still subject to discussion. Synonyms can be extended with other semantic relations like meronyms and

[14]Self-explanation describes a learning process as an act of learning new concepts by explaining them to oneself.

hyponyms. Relations like those represent the connectionist interpretation of meaning (see Definition 21). The second part of meaning can be seen as the literal definition of the meaning of words (see Definition 22). The definition of meaning in a symbolic view that needs some kind of logic to reason upon. In consequence, creation of a formal representation of meaning leads to the "neat vs. scruffy" discussion of Minsky [273]. Both Definitions 21 and 22 together form our representation of meaning combining the neat and the scruffy view of meaning. They represent the knowledge we learn in a connectionist view and the thoughts using this knowledge in the symbolic view. Both combined enable our representation of meaning to share semantic information and to reason depending on context. How we approach this definition for an AI is subject of Part III of this doctoral thesis.

This notion of meaning coincides with the NSM theory since here the references are contextual. We will use the NSM theory through this work as a basis for our formal representation of meaning in the research done here exceeds the other theories with its empiric evaluation. Furthermore, the theory leads to the convenience that the meaning of complex concepts can be described with less complex concepts, which ends in the benefit for our artificial meaning representation that we require ca. 65 definitions of concepts, leaving the construction of meaning of more complex concepts to the AI. This discussion and the here presented description of meaning leads to the approach of the here presented doctoral thesis. The approach is further described in Section 5.

3.10. Natural Semantic Metalanguage

The Natural Semantic Metalanguage (NSM) is a linguistic theory originated in the early 1970s [137, 138, 142, 141, 143, 144, 145, 146, 406, 407, 408, 409, 410, 411]. It stated that each expression created in a natural language can be represented using a set of atomic terms – so-called universal semantic primes. These primes have an indefinable word-meaning and can be identified in all natural languages [140]. In conjunction with associated grammatical properties, NSM presents a decompositional system able to describe all expressions built in the appropriate language. Here an expression is decomposed into less complex expressions, where the process ends if the expression is decomposed to the atomic level of semantic primes which cannot be further analyzed. One can imagine that the Decomposition builds a tree, where all leafs are semantic primes. Consequently, for each natural language, a meta language exists which consists of the semantic primes in the specific syntax and their appropriated grammatical properties. In this work, those semantic primes are predefined by the applications domain. About 63 semantic primes exist which can be divided into 16 categories. Table 3.2 lists these semantic primes for the English language.

This theory has been tested on natural languages [140, 144, 406, 407, 408] and in many different cultures. The essential conclusion of NSM is that the primes used to build the meaning of more complex concepts are limited.

Category	**Primes**
Substantives	I, YOU, SOMEONE, SOMETHING/THING, PEOPLE, BODY
Relational substantives	KIND, PART
Determiners	THIS, THE SAME, OTHER/ELSE
Quantifiers	ONE, TWO, MUCH/MANY, SOME, ALL
Evaluators	GOOD, BAD
Descriptors	BIG, SMALL
Mental predicates	THINK, KNOW, WANT, FEEL, SEE, HEAR
Speech	SAY, WORDS, TRUE
Actions, events, movement, contact	DO, HAPPEN, MOVE, TOUCH
Location, existence, possession, specification	BE (SOMEWHERE), THERE IS, HAVE, BE (SOMEONE/SOMETHING)
Life and death	LIVE, DIE
Time	WHEN/TIME, NOW, BEFORE, AFTER, A LONG TIME, A SHORT TIME, FOR SOME TIME, MOMENT
Space	WHERE/PLACE, HERE, ABOVE, BELOW, FAR, NEAR, SIDE, INSIDE
Logical concepts	NOT, MAYBE, CAN, BECAUSE, IF
Intensifier, augmenter	VERY, MORE
Similarity	LIKE

Table 3.2.: Semantic primes for the English language [140].

4. State-of-the-Art

This section describes the state-of-the-art in the different parts of our approach. Some of the decisions we have made in Chapter 3, are the product of a discussion and a related work analysis. These discussions and the literature reviews are part of this chapter. For a theoretic base of our representation of meaning, discussion on the choices in design decisions as well as comparison with other approaches, we will look at different semantic theories in Section 4.1. Semantic information in computer science is mostly encoded in ontologies. Section 4.2 discusses different ontological representations. Then we will look at semantic service description language in Section 4.3 to decide how to describe our services. Next, we will look at related work for the first part of our approach: approaches on semantic decomposition in Section 4.4 and the formalization semantic primes. We then look in Section 4.5 at related work on the second part of our approach: Marker Passing. Finally, in Section 4.6 we will look at different service composition mechanisms which use AI Planning. State-of-the-art on the different experiments is described in the respective section of the experiment.

4.1. Semantic Theories

There is a multitude of theories which describe meaning in a semantic sense for computer science. This section will describe some of them, which could be an alternative to how we create our artificial representation of meaning. These theories can be seen as a formal basis on which we will base later design decisions. This state-of-the-art section shows that the challenge of creating a formal representation of meaning has been researched and builds a foundation on which we can build upon. We will start our by describing 11 different semantic theories from Section 4.1.1 to Section 4.1.11 and classifying them how they represent meaning: connectionist or symbolic or both.

4.1.1. Formal Semantic

This section contains approaches to formal semantics, which have no name but are grouped by the main contributors of certain schools of semantics. This is done to get a general overview of the development of formal semantics and stipulate a notion of formal semantics. Formal semantic has many explications. In this thesis, we will only look at a few of them, which are relevant to this work. We will hence not dive into the philosophical discussion as, e.g., presented by Frege [124] and the same but rather start out with 20th-century definitions of semantic and meaning with Bloomfield [29, 31, 32] who tried a definition of meaning with its "stimulus-response" model as follows:

> "We have defined the meaning of a linguistic form as the situation in which the speaker utters it and the response which it calls forth in the hearer [32]".

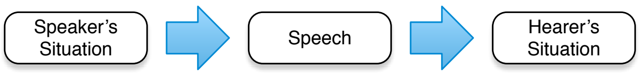


Figure 4.1.: Illustration of Bloomfields definition of meaning.

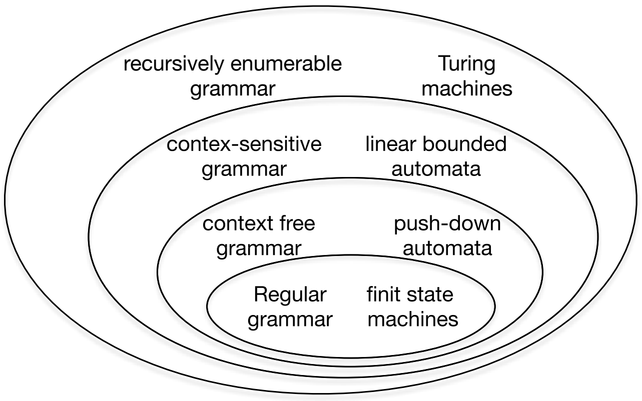


Figure 4.2.: Illustration Chomsky's grammatical hierarchy.

The meaning in the definition of Bloomfield, therefore, is not restricted to the speech act itself. It includes the speaker and the hearer. The meaning, therefore, is described by "the situation" of the speech act. Bloomfield additionally states that for the response of the hearer to reflect the meaning intended to be communicated by the speaker, the situation of the speaker has to be known [32]. This illustrates a pragmatic notion of meaning, which takes the speaker context into account.

As illustrated in Figure 4.1 the response of the hearer is not seen as a verbal response. The response is rather a mental interpretation of the speech act. In this case, we can not define how much information about the speaker is known. In consequence, reducing the definition of Bloomfield to a semantic meaning: the semantic meaning of an utterance can be defined by the meaning of Bloomfield when the hearer has no information about the situation of the speaker.

This first try on formalizing meaning is vague as it is abstract. The definition is an extension of Bloomfield [32] neglects to notice the difference of word to sentence meaning. In a sentence, the meaning might come from the structure and syntactical features of the words and in consequence can be analyzed as well to gain semantic insight [12, 36].

The next area of formalization of semantics is the area introduced by Noam Chomsky [53, 55]. His theory on a generative conception of grammar and its influence on meaning has become a standard content of computer science curriculum. Chomsky argues that grammar is a formal deduction process, rather than the application of operations on syntactical structures [188]. This gives us the ability to explicitly formulate a grammar describing an observed language.

With his hierarchy of grammars and their computational expressiveness shown in Figure 4.2 Chomsky introduced formal tools for the explicit description of meaning in an symbolic way. Since we could build a Turing Machine (or any other computable function) which can enumerate over all valid words of a natural language, natural languages are recursively enumerable and belong to the most general (type-0) class of Chomsky grammars.

Out of this research, linguists start to look at the meaning of concepts and with the work of Fillmore and Lakoff towards the meaning of sentences. The linguistic research began to analyze the difference of natural and artificial language and become aware, that the differences are not as big as Chomsky has postulated [52]. This leads to the research done today and as we will show later on, to the contextual meaning named pragmatics, as we attempt in this work. Since Bloomfield and Chomsky still lack the expressiveness (Chomsky) or are too abstract (Bloomfield), they can only be used as an abstract framework for our artificial representation of meaning. This means we have to analyze the other schools of thought which might be a basis for the contributions of this work. This leads to the next section describing cognitive semantics.

4.1.2. Cognitive Semantic

Cognitive semantics arrises form the field of cognitive linguistics. Here the focus of research is on the relation between language, cognition and meaning [8]. Supporter of cognitive semantics argue that not only meaning can be described in a conceptual structure but syntax, morphology, and phonology are conceptual as well [62]. The conceptualization of syntax makes this representation a symbolic one. There are three major hypotheses guiding cognitive semantics [62]:

Nonautonomous language processing means that the language is part of the whole cognition. This connects the language to the nonlinguistic cognitive abilities. Especially the way we see things, smell or feel (by touch) does influence our conception of our language. In simpler words: how we experience the world forms our language.

Syntax is conceptualization means that grammar can not be simply mapped to truth values of a model of the world. In particular, the syntax of a language forms its meaning and can not be described without semantics.

Language is learned means that the grammar of a language needs to be used to gain knowledge about it. As a result, there can not be an abstract or general description of syntactical properties of a language. This is needed because of the contextual manner the language is seen in cognitive semantics.

With this basis cognitive semantics constructs meaning out of a conceptualization which includes syntax. This construction is not independent of congestion and depends on how the world is observed. Different to other models cognitive semantics do not represent meaning through truth evaluations of statements. One of the most popular versions of cognitive semantics is the Frame-Semantic [62]. Two basic concepts from cognitive semantics are introduced in our approach: The notion of meaning-construction and knowledge representation. However, the mix-up of semantic and syntax is making the insight into meaning more complicated. This is why cognitive semantics are not chosen as used semantic for our endeavor to creating artificial meaning.

4.1.3. Compositional Semantic

The compositional semantic describes the semantic meaning of concepts as in the structuralist theory of meaning [67], with its relations. The founding axiom of a compositional semantic: “Meaning of all elements in languages are not atoms, but are composed.” [235, p. 164]. Hence the meaning of a concept is defined by its relations with other concepts.

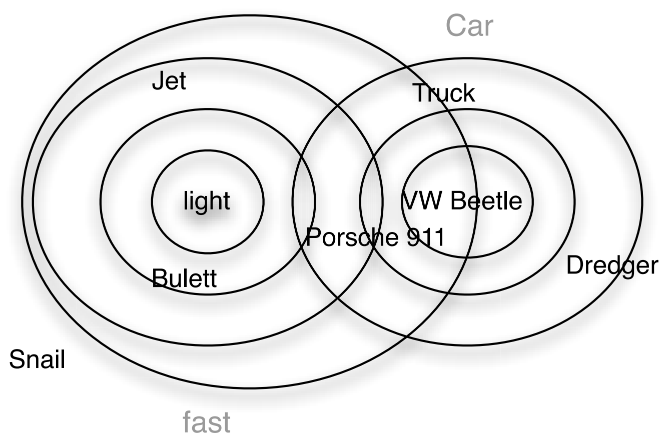


Figure 4.3.: Illustration of two prototypes for the class of car and of fast.

The theory of compositional semantic defines, therefore, distinctive semantic features called *Seme* to extract the elements making up the composition (semantic meaning) of a concept. For example, such a Seme could be $^{+}_{-}$*BIG* to differentiate the meaning of a river (+BIG) and a creek (−BIG) [235, p. 164].

The semantic analysis in the compositional semantic works well for nouns, adjectives, and verbs so-called *Autosemantica* which have a conclusive lexical morpheme[1]. The theory is less used to analyze functional words like preposition and articles so called *Synsemantica* [235]. Further, there is no fuzziness in the use of Seme. A concept has a relation with a Seme, or it does not. The theory of compositional semantic neglects this indistinct aspect of language and in fact seems not well fitted to describe a semantic for an AI. The compositional Semantic can be seen as a defuzzified version of the Prototypical Semantic, which we will analyze next.

4.1.4. Prototype Semantic

The prototype semantic denotes meaning of concepts with fitting references called prototypes. The other concepts then are set in relation to this prototype creating a semantic distance to the prototype [235, pp. 175]. This, in addition to being a connectionist representation, gives the meaning of a concept a fuzzy notion since a concept might be related to multiple prototypes.

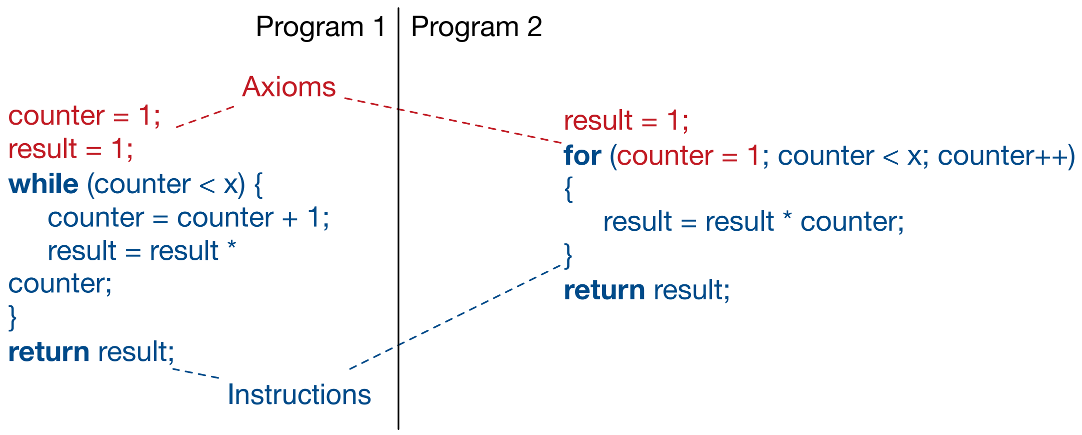
In Figure 4.3 we can see two concepts (car and fast) and different prototypes for the two concepts (light and the VW Beetle). In this example, a Porsche is close to a car and faster than a Dredger.

The distance to a prototype and its discretization are problematic in this semantic representation since the different concepts need to be set into a relation. Here the snail is two times as far from light than a Porsche, which would introduce a distorted measure of speed.

The prototype semantic as a result, lets us define fuzzy concept relations but lacks a formal definition of distance between concepts and has the deficit that the prototype needs to be known to all which use the representation. This might not be the case in different cultural contexts.

Prototypical semantic has the benefit, that abstract concepts can be directly reference by instances. Making it easy to compare their properties even if they are vague or fuzzy. For a representation of meaning in AI, this would mean to create a data structure for each concept with all other concepts. With languages like German, with an infinite amount of words, this

[1]A morpheme is the minimal meaningful linguistic sound



```
counter = 1;
result = 1;
while (counter < x) {
    counter = counter + 1;
    result = result *
counter;
}
return result;
```

```
result = 1;
for (counter = 1; counter < x; counter++)
{
    result = result * counter;
}
return result;
```

Figure 4.4.: Example of two similar programs and the annotation for a axiomatic semantic view.

seems impassable. Also, we would need to be able to place those concepts in the distance to each other. Further, this kind of representations captures the relation of concepts but does not combine their features as we do in sentences.

4.1.5. Axiomatic Semantic

In the axiomatic semantic approach, computer programs are seen as exact science where a computer program consists of a set of instructions [169]. This school of thinking got to the Hoare logic, where a set of logical rules and reasoning instructions make up a formal system with its semantics. Therefore this is an symbolic representations. This means that the meaning of a program is not given explicitly but needs to be executed to show its properties [348]. Here a property of a computer program is reasoned by using axioms given by the program and formal logic for proofing those properties. Here one of these properties could, for example, be the equality of two programs like shown in Figure 4.4.

The axiomatic semantics are used to guarantee general properties about a program. This is formally done with logic and formulation of precondition and post conditions. An example post condition of the example programs in Figure 4.4 could be $result = x!$.

For the creation of an artificial notion of meaning, the axiomatic semantics is too restricted in its interpretation. It is thought for probability and explains how program languages are interpreted, but it is based on first-order logic which is too restrictive to describe natural language.

4.1.6. Operational Semantic

In Operational Semantics the semantics of a computer program are described through execution of a program. Here the program code is not translated but interpreted into a mathematical model, which then serves as a basis for proofs about the behavior of the program. Which makes this representation of meaning a symbolic one. Similar to Axiomatic Semantics the proofs about properties of the program are done by logical conclusion. Here the definitions are less abstract than in Axiomatic Semantics [348]. A common example of an operation semantic is the interpretation of a while loop:

$$\frac{\langle \text{Condition, Term} \rangle \Rightarrow \textbf{true}}{\langle \textbf{while} \text{ Condition do } \textbf{Body}, \text{ Term} \rangle \rightarrow \langle \textbf{Body}; \textbf{while} \text{ Condition do } \textbf{Body}, \text{ Term} \rangle}$$

This states in the numerator, that if the condition holds in a given state, then the denominator is executed. We call the body of the loop **Body**, the condition of the loop is called Condition, and the pointer to where to go when the while loop has ended is denoted in “Term.” The above statement means the **Function** is executed and the while loop is repeated. This is denoted by the semicolon which means a sequential execution. If the conditions do no longer hold the following statement changes the current state:

$$\frac{\langle \text{Condition, Term} \rangle \Rightarrow \textbf{false}}{\langle \textbf{while} \text{ Condition do } \textbf{Body}\text{, Term} \rangle \rightarrow \textit{Term}}$$

Here the condition is false therefore the state is changed to the termination condition Term. This shows the essential drawback of a while loop: If the body of the loop (in a nonconcurrent program) does not change the condition, then the loop does never terminate.

Operational semantics are even less abstract then axiomatic semantics and interpret a computer program which restricts its interpretation to the mathematical model it is interpreted in. With this reason, the interpretative power of operational semantics is not enough to describe natural language and in consequence can not serve as a basis for this work.

4.1.7. Denotational Semantic

Denotational Semantics is a representation of meaning in simple statements of formal languages like programming languages. The realization of a statement is called its denotation.

In denotational semantics, the meaning of utterances is created of the composition of symbols. If we take a symbol like “X $\leftarrow$ 3;”. The X as a single symbol does not carry much semantic information[2]. Composed with the assignment symbol $\leftarrow$ and with that the value 3 being assigned, terminated with the “;” as an end of statement symbol, the composition is a statement, which has to mean: 3 is the denotation of X. Here the denotational semantics postulates that the composition of those symbols make up the meaning of the statement [348].

Schmidt [348] compares denotational semantics to operational and axiomatic semantics. In operational semantic, the change in memory an executed program inflicts is the meaning of the program. In axiomatic semantics axioms of a language in the form of properties of symbols are given, building up the meaning of a program using, i.e., logical inference.

Durst [82] argues that a denotational semantic is insufficient to describe the meaning of natural languages. He argues that depending on the context of the utterance the same denotations could have a different meaning. For example, the same loop of Java code could mean different things depending on the surrounding program it is used in. A program gets its meaning through the programmer. Every step from the Java code, through the compiler to the execution, is just meaning preserving transformations [348].

Here the operations (zero, one,... and $+, *$) are the meaning we want to map. In consequence, we postulate in the semantic that those operations are interpretable by the one for which this semantic is written. With this example describe next, we defining one way to interpret a binary code of numbers. With this semantic we have a direct interpretation which can be seen in small example of 101:

$$\mathbf{B}(010) = (((\mathbf{D}(1) * \textit{two}) + \mathbf{D}(0)) * \textit{two}) + \mathbf{D}(1)$$

[2]The pragmatic interpretation might be that as scientists we often use x as a variable, but this is pragmatic (context dependent) meaning not semantics.

Denotational definition of binary numbers

Abstract syntax:

$B \in \{0,1\}^*$

$D \in \{0,1\}$

Semantic algebra:

Domain : $\mathbb{N}$

Operations : zero,one,two,..., $+, *$

Valuation functions:

B(): $\{0,1\}^* \rightarrow \mathbb{N}$

B(BD) ::= (**B**(B) * two) + **D**(D)

B(D) ::= **D**(D)

D(): $\{0,1\}^* \rightarrow \mathbb{N}$

D(0)::= zero

D(1)::= one

With **D**(1) being one. With the continuous replacement of terms we reach:

$$\mathbf{B}(010) = (((one * two) + zero) * two) + one = five$$

In this example, we can see how denotational semantic directly maps the terms to meaning. This can be done for the formalization of many simple structures like done in formal specifications of data structures [85] but lacks the expressiveness to describe more than a functional adaption of operations on a mathematical domain [108]. Which makes this representation a symbolic one.

This semantics as been further developed by Wang et al. [399] into a so-called deductive semantics. Deductive Semantic is used to describe computer programs formally regarding so-called semantic **environments** and **behaviours** [399].

The denotational semantics are less abstract than most of the linguistics models and are well fitted to describe mainly deterministic computer programs. Especially in the form of a formal language, this representation lets us reason about the meaning of formally well-defined descriptions. This has the drawback that the formulation of more abstract notions becomes too complex or not expressible. In the same argument as Durst [82] this kind of semantics can not form the basis of the approach of this thesis.

4.1.8. Generative Semantic

Generative Semantics has developed out of Chomsky's ideas and is the part of semantics where philosophy or more precise logicians meet linguists. Now in the 20^{th} century the structure of sentences has been taken into account, which leads to the idea that this structure represents the meaning of the sentence combined with the word senses. This includes the idea that meaning can be formalized as first order logic. Which makes this representation a symbolic one. Two of the researchers in this domain are Fillmore [116] and Lakoff [218] which have represented meaning in this form and extended the logic with so-called "sentence operators".

The ideas of Frege [124] have been worked on by Kripke and Barwise [19, 210] to generate something called **Generalized Quantifiers**. They classify the Generalized Quantifiers in groups of "Logical" and "Nonlogical" symbols. The logical symbols are based on first order logic like and or, terms and quantifiers. The Nonlogical Symbols include symbols like for concepts like "most", "many" or "few" [19]. The proposed semantic for those Generalized Quantifiers extends the idea of Frege for sense, connotation, and denotation. This leads to formalizations like shown in Figure 4.5.

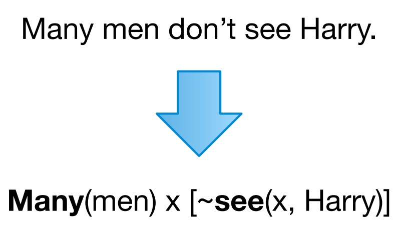

Many men don't see Harry.
Many(men) x [~**see**(x, Harry)]

Figure 4.5.: Formalisation of the sentence "Many men do not know Harry" with the formalization of Barweise [19].

With that, we can see the notion in which the 20th century has continued to develop the notion of meaning. We can see in Figure Figure 4.5 that the logic contains operator "**Many(x)**" which is such an Generalized Quantifiers. It extends the First order logic by a fuzzy construct which reaches from more than one to not all.

This lets us describe more aspects of semantic formally, but is restricted to symbolic meaning. Leading to a theory where in an application for AI every concept needs modeling like in an Explanatory Combinatorial Dictionary [261] or another model theoretic semantics which try to model an exhaustive formalization of meaning [188]. Also, the reasoning for many of those Generalized Quantifiers becomes undecidable because the extend the description logic and with that runs into the decision problems.

4.1.9. Discourse Representation Semantic

Further development in this direction of Kripke and Barwise [19, 210] has become research about **Discourse Representation Semantics** [188] which lead to the formalization of two problems: the first one is how to handle fuzzy concepts like **some**. The second one is a concept with tenses and aspects[3]. The Discourse Representation Semantic focuses on the integration of interpretational aspects of representations of meaning [188]. Here a mental abstraction of the "discourse" is part of the formal description of the semantics. Making this representation a symbolic one. This is done to be able to describe meaning on larger entities like sentences or paragraphs. The formal model describes two types of information: the "reference" which are the subject of the discourse and the "conditions" holding information about those references.

As shown in Figure 4.6 the references are kept separated from the descriptions of information about those references. In this way, the reference can overarch multiple sentences, and form a discourse. Sadly this good idea does not contain a set of grounded primes since it is specially designed to represent discourse. Which leads us to the conclusion that the discourse representation semantic can not be used as a basis for the here created artificial notion of meaning. However, it can be part of its extension, when the notion of meaning is extended from concepts and sentences to larger textual units and finally could be part of future work.

[3]This analyses grammatical structures like the difference between two types of past tenses like the French "Passe Simplé" and the "Imparfait".

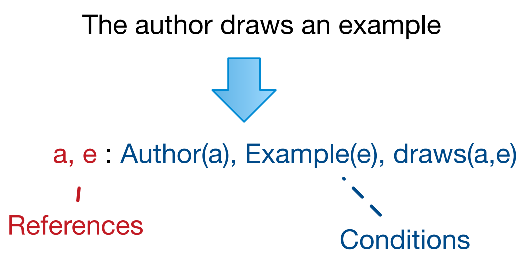

The author draws an example
a, e : Author(a), Example(e), draws(a,e)
References
Conditions

Figure 4.6.: Example in discourse representation semantic.

4.1.10. Distributional Semantic

An again hyping semantic representation is the distributional semantic (sometimes called word embedding) [118, 156]. The basic idea behind this is that words which cooccur carry similar meaning [156]. This theory origins from statistic linguistics where the usage of words is analyzed. The meaning of a word is therefor defined by the words it cooccurs with. This theory of meaning is called the **Distributional Hypothesis**.

In distributional semantic, the words which are most likely to cooccur with the given word are added to its representation vector. Depending on which kind of words are added to the vector, the representation changes. Figure 4.7 shows an example where the cooccurrence is a statistical one where the corpus "Google News negative 300"[4] is used. Here the vector contains the 300 words which are most probable to cooccur with the given word when using 3 million words and phrases from news articles.

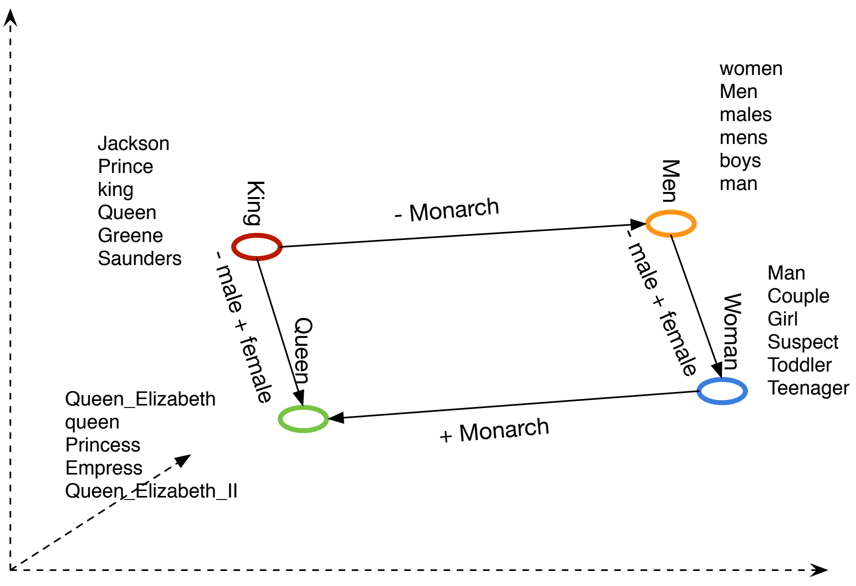


Figure 4.7.: Example of distributed semantics.

In Figure 4.7 we can see an example of how arithmetics are used to calculate with semantic information in a distributional vector space. This kind of calculation allows us to describe a similarity measure with geometric distance measures like the cosine distance [269].

[4] `https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit?usp=sharing` visited on 2017.03.16

The drawback of the distributional semantics, as a connectionist model, is that they encode relatedness not semantic. We can see this in Figure 4.7 that in the corpus which was scanned for this example Jackson was related to the concept king probably because Michael Jackson was called “the king of pop”. This disqualifies the distributional semantic for our approach.

4.1.11. Structural Semantic

Structural semantics are semantic which search for semantic meaning full components. We will look at two of these semantic theories which have a basis for many extensions. First, we will look at the conceptual semantic, because it is closer to the formal semantics. This is intriguing for the formalization of meaning for an AI since it lets us structure the semantics of a concept up a whole phrase.

Conceptual Semantic

Conceptual semantics theory after Jackendoff who was looking for the smallest semantically rich elements (morphemes) is based on the axiom that semantic meaning is built upon a finite set of basic semantic concepts called *conceptual primitives*. Jackendoff beliefs that these conceptual primitives are based on human cognition and therefore have to be universal [182]. These conceptual primitives are further order and categorized by Jackendoff. As an example, we look at the movement of objects and with that the primitive *GO* taken from the Studienbuch Linguistik [235, pp. 188]:

$$[_{Action}CAUSE(PAUL, [_{(Event)}GO([_{Thing}butter], [_{Path}TO([_{Place}bread])])])]$$

In the example above we can see that there are *Action, Event, Thing, and Place* primitives, which carry meaning which makes this representation a symbolic one. The CAUSE (as in causality) is classified as an Action and GO is classified as an event. The primitive GO seems to have two parameters: First a Thing and second a Path primitive. In our example, butter is a thing, and bread seems to be a place. This leads to the conclusion that example above formalizes the following sentence: Paul butters the bread. The example of movement and the primitive GO is further discussed in Goddard’s semantic on coming and going [138].

This approach is more formal then the others looked at so far, but it is mainly concerned with verbs that can not (without extension) be generalized [235, pp. 188]. Because of its degree of formalization, this is a candidate for our representation of meaning. The primitives used here are subsumed by the primitives used in the Natural Semantic Metalanguage which will be discussed in Section 6.1.

4.1.12. Conclusion

In conclusion we can define the scope of connectionist meaning for this work as described in Section 3.9. From the literature review done in this section, we can see that no approach takes account for both sides of meaning: the connectionist and the symbolic. Table 4.1 lists all analyzed theories and our classification of them into the two sides. Which leads to the gap filled in this work: the need for an artificial representation of meaning combining connectionist and symbolic views. Therefore the semantic representation and the language which describes this

Table 4.1.: Comparison of semantic approaches regarding the classification into symbolic and connectionist theories.

Theory	Symbolic	Connectionist
4.1.1 Formal Semantic	✓	×
4.1.2 Cognitive Semantic Section	✓	×
4.1.3 Compositional Semantic Section	✓	×
4.1.4 Prototype Semantic Section	×	✓
4.1.5 Axiomatic Semantic Section	✓	×
4.1.6 Operational Semantic Section	✓	×
4.1.7 Denotational Semantic Section	✓	×
4.1.8 Generative Semantic Section	✓	×
4.1.9 Discourse Representation Semantic Section	✓	×
4.1.10 Distributed Semantic Section	×	✓
4.1.11 Structural Semantic Section	✓	×

representation consists of two parts: The connectionist part (represented by an ontology) and the symbolic part represented by the Marker Passing.

Semantic or with that meaning is hard to grasp - especially if we are looking for a formal definition. We started with definition 17 and looked at different theories of semantics. Now we have a broader understanding on what different theories focus on in semantics. The different theories of semantic look at different aspects of semantics depending on the aspect they want to explain regarding meaning.

We have seen in this section that the formalized description of semantics are still too abstract like the Chomsky hierarchy or are too specific for one part of language like conceptual semantics. To represent our definition of meaning as defined in Section 3.9 we need a semantic theory which incorporates both, a symbolic and a connectionist interpretation of meaning.

In computer science, this kind of semantic has been described in special semantic languages, e.g., for the description of functionalities in services. We will look at some of those description languages next.

4.2. Ontology

The goal of this section is to select an ontology language for the representation of our semantic graph, modeling the connectionist part of the meaning. We do so by surveying the formal definitions of ontologies since they are the theoretical foundation for most connectionist knowledge representations [366]. At first, we will identify a set of language requirements that are relevant for our semantic graph. We structure our survey by starting with the simpler, more abstract definitions and will find our way to more precise and complex definitions, leading to the definition of an ontology we apply in this work described in Section 3.6. Furthermore, we use the derived set of language requirements to substantiate the selection of the concrete ontology language, which we use to model our semantic graph.

We have defined an ontology in Section 3.6 and now we derive language requirements from this definition for the language describing our ontology: The first differentiation in Figure 3.8 is that the ontology consists of concepts and relations. Therefore our first language requirement is:

Language Requirement 1. *Concepts and Relations: the language has to be able to describe concepts and their relations.*

Since some of the concepts used in natural language are sometimes ambiguous, we need to be able to identify concepts uniquely. An example of an ambiguous word is, e.g., the word “can”, which could be the ability to do something or a metal container. Both have the literal “can”, but represent other concepts in our ontology. Thus our second languages requirement:

Language Requirement 2. *Unique identifiable: the language has to be able to uniquely identify each concept.*

With the semantic prime “I” NSM introduces a self-reference. Our language for our ontology should be able to describe such a self-reference, which leads us to our third language requirement.

Language Requirement 3. *Self-Reference: A concept or relation belongs to one of its Complex.*

A concept is a self-reference if it belongs to its complex (for the definition of complex see [250] or Definition 28 on Page 63). A relationship is a self-reference if one concept which is related by this relationship is the relationship itself. A Complex is a self-reference if the complex belongs to its relations or its concepts. Also, every concept is the content of a conception, which is a complex [249]. This theoretical requirement means for us that we can use, e.g., the literal of every concept, every relation or every individual and define it as a concept. This cyclic definition of a concept is formulated in our forth language requirement:

Language Requirement 4. *Cyclicity: Every concept belongs to a Complex.*

Some of the relations in an ontology are hierarchical like the inheritance relation. This is why relations should have the ability to describe a hierarchy, which leads us to our fifth language requirement:

As defined in Pickert’s definition we derive the need of a hierarchy in concepts. Which lets us formulate the following language requirement:

Language Requirement 5. *Relations should be able to form a Hierarchy. The concept and relation types can build a hierarchy. As we can see in our example in Figure 6.2 “give” is a specialization of some kind of transfer here depicted as the “is-a” relation. If those are modeled as types, then the type graph needs to be able to reflect this kind of inheritance.*

How concepts relate and which concept belongs to which conception can be formulated in axioms. Axioms describe how facts which all concepts in an ontology have to obey. To obey means here in a logical term: to evaluate to true. Formulating axioms on concepts gives us the ability to describe facts like “All humans are mortal.”, which leads to the effect that every concept inheriting from human has to be mortal as well. This brings us the sixth language requirements that the ontology should have the ability to describe axioms:

Language Requirement 6. *The ontology can describe axioms which map a relation with its concepts (or individuals) to a boolean value.*

Form the Semiotic Triangle (see Figure 4.8) from Ogden and Richards [291] we derive that we want to be able to distinguish between concepts and individuals. This differentiation is equivalent to the difference of symbol and the reference in the Semiotic Triangle, which gives us our sixth language requirement:

Language Requirement 7. *The difference between concepts and instance should be made explicit.*

Since natural language relations do not only connect two concepts but sometimes more, e.g., if we can express something like: "John writes with chalk on a blackboard." [189] in one relation, we need n-ary relations which are formulated in our seventh language requirement:

Language Requirement 8. *N-ary relations: the language has to be able to describe relations with arbitrary many entities.*

Because of n-ary relation in natural language like the example as depicted in Figure 6.2: `give( teacher, advice, student)`, a relation can relate more than two concepts or relations, making it necessary for our formalism to connect multiple concepts or other relations with one relation.

Furthermore, the definition of Euzenat and Shavaiko identifies types which are a more pragmatic notion of a relationship (is-of-type) between a typed concept and a concept of this type. This is reflected in our definition by the instantiation, which leads us to our ninth language requirement:

Language Requirement 9. *Typed concepts: the language has to be able to describe types for each concept.*

Language Requirement 9 describes that the formal representation of concepts and relations are typed. This enables a different interpretation for each concept and relation type. We can see the example decomposition depicted in Figure 6.9 on Page 114 where the different concept types could be stop-words, negation, syntactic relations or semantic primes.

In addition to typing of concepts, concepts can have additional attributes. The possibility of modeling those attributes is formulated in Language Requirement 10.

Language Requirement 10. *Attributes: In addition the entities and relations need attributes, e.g., for us to be able to give them names.*

The need for modeling relations as concepts can be seen in Figure 6.2 on Page 98 where the relation "give" as a relation is itself a concept. Concepts which represent relations can again have relations to other concepts or relations. Making it necessary to introduce a relation-to-relation connection.

Language Requirement 11. *Relations are Concepts: Each relation might be interpreted as a concept as well, hence the set of concepts and relations are not disjunct.*

Language Requirement 11 comes from the need of natural language relations are concepts as well, and therefore can be described by concepts, a relation should be a concept as well.

With those requirements in mind, we will now analyze different ontology representations and evaluate their usefulness for our representation of connectionist meaning.

The representation of information or more accurate of knowledge is a well-researched topic. Starting with Philosophy[5] as a part of structural science our knowledge is formalized in models called ontologies. The term Ontology is taken from philosophy where an ontology can be seen as a theory of the things in nature [152]. With this theory, we try to understand the world and remove errors in our thoughts. For example, the color green is often seen as a property of something, but a coat of green paint can be modeled as layers of color [41] even though the object is called green. If we have the knowledge of an object which is of the color red and is covered by a coat of green paint, would it fall under the description: "This is a green object."? To formalize such knowledge, the research in ontologies has fostered many languages and engineering tools.

With an ontology, we try to analyze the meaning of entities. We define entities like Mahr [249] as everything that is. This includes the real object of interest, e.g., the apple and the word describing the apple as well as the relations the objects have with each other. The meaning of things is strongly coupled with our language and the symbols we use. Ogden and Richards [291] have studied this relationship to conclude the semiotic triangle [6] as shown in Figure 4.8.

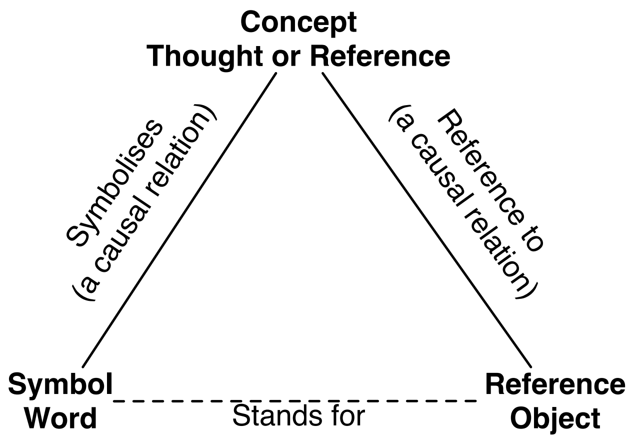


Figure 4.8.: The Semiotic Triangle.

In the Semiotic Triangle entities which exist are referenced by symbols like words, for example, "Apple" which symbolizes every fruit which falls under the definition of an apple. If we read the word "Apple" and we have a reference for that word, then there might be thought, which references the actual reference (the apple). Here the symbols themselves are entities, which have references and we consider thoughts as entities as well, hence having symbols and references for them as well. We investigate the process where the thought is created regarding a symbol and its references to grasp the notion of meaning which lets us connect symbols to references and reason upon them. We use the terminology of concepts to describe a symbol and an instance to describe the reference.

Peacock [303] analyses a philosophical description of concepts, e.g., by Kante as basic building blocks of an ontology: "*...concepts have pride-of-place in epistemology, semantics, and the philosophy of mind: they function as rules for organizing perceptions, as the primary object of rational analysis, as singular or general propositional terms and as the basic constituents of beliefs.*" [303, pp. 266]. For our work, we want to transform such a philosophical definition in

[5]The study of existence or being.
[6]some times called Semantic triangle or triangle of reference.

a more formal one, to be able to implement AI reasoning upon it. This is the way we define language requirements so that we can select which kind of ontology definition we need.

In computer science, an ontology can be seen as a practical application of such a theoretical view [365]. In computer science ontologies have earned their place in an own layer of the technology stack of the Semantic Web [25], which gives the notion of its importance in the scientific community regarding artificial semantics. However, now let us look more closely at the definitions of ontologies starting with abstract ones to more technical.

A quite general description of ontologies has been given by Gruber with "*an explicit specification of a conceptualization*" of entities of concern and their relationships[7]. This definition is quite abstract and does not formalize the notion of an ontology but rather explains what is done to create an ontology which can be done by a multitude of modeling techniques.

To grasp the notion of ontologies more formally than the definition of Gruber, we need to look at similar terms first, to distinct them from ontologies:

Model: A model can be seen as an abstraction of an entity. Containing the relevant information regarding a given context. Favre [106, 107] would describe it as "*a system that enables to give answers about a system under study without the need to consider directly this system under study.*" [107, p. 3] and further Miller [272, p. 3] describes a model as "... *a model is a formal specification of the function, structure and/or behavior of a system*".

Epistemology: Epistemology can be seen as "*the field of philosophy which deals with nature and source of knowledge*" [289] (cited in [355, p. 6]). Regarding AI research the knowledge is interpreted as consisting of propositions and logical reasoning upon those propositions to create new knowledge in the form of formal structures [154].

Taxonomy: A Taxonomy is the science of classification of entities into classes. Taxonomies structure entities into an order which can be but does not have to be hierarchical [354]. Further Euzenat and Shavaiko [91, p. 27] define a taxonomy as "*a partially ordered set of taxons (classes) in which one taxon is greater than another one only if what it denotes includes what is denoted by the other.*"

A model as defined above is on the one side to general to examine the meaning, since there is no formal or better-structured definition of what the model consists of. However, on the other side, we can define an ontology as a kind of model since an ontology is an abstraction as well. Euzenat and Shavaiko [91] define an ontology as a conceptual model with features of an entity-relation model. The ontology is thereby seen as a logical theory with model theoretic semantics.

For this work, Epistemology is, on the one hand, a use case in which an ontology is used to reason upon and on the other hand, the process of ontology creation[8] where the output of this process is an ontology.

A Taxonomy seems to restricted since the classification of entities is not enough if we need more details in our ontology then the sub-class or "is-a" relation to represent relationships. This is necessary for example if we want to reason upon relations like "part of" or concepts like the relation "give" in an example like: "A teacher gives advice to a student."

Figure 4.9 illustrates a simple taxonomy using the "is-a" relation which identifies two kinds of things: living ones and not living ones. Also, a salad and a pig are classified as living and

[7]`http://www-ksl.stanford.edu/kst/what-is-an-ontology.html` last visited 2017.09.01

[8]sometimes called ontology engineering

stone as dead. This gives us an example of how an ontology can capture part of our beliefs and knowledge.

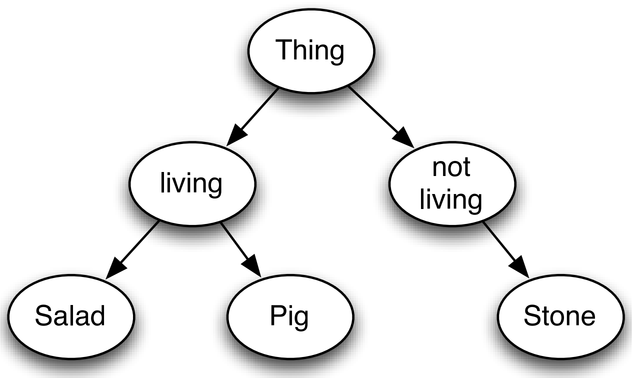


Figure 4.9.: An example ontology for living and not living things.

These definitions so far (here Model, Epistemology, and Taxonomy) is not precise enough for a formal analysis, and we need to further detail our notion of an ontology. For that, we will have a detailed look at the most common formalization of structured information and definitions of the term ontology. They all have in common that they are used to describe the world and structure it as a model as far as possible.

However, first, we have to look at a different kind of formalization. There are two well-researched theories to formalize ontologies: Set theory and Mereology [361]. We will have a short look at both:

Mereotopology We start out with the theory of Mereology which became popular in the early 20^{th} century. Mereology is based on the principle of parthood [350] which is similar to the notion of a subset relation in set-theory. The fundamental element in Mereology is called "Object" and is specialized into *thin* and *thick* objects to model change. Where thick objects undergo change and, in consequence, are volatile, and the thin objects are invariant. For that reason, a thick object can be identified over time by its set of thin objects which are its parts[9].

Set-theoretic Ontologies A second and more common formalization of an ontology is a set-theoretic view [361]. Set-theory is one of the most basic concepts in mathematics. The basic entity in set-theory are *elements* which are joint to group of elements called *sets* by using the *element relation* ($\in$). With the *subset* ($\subset$) relation one can declare an hierarchy of sets. More formal approaches model set-theoretic ontologies as category[10] like in the work of Patterson [300], e.g., allows only relations between two sets of concepts.

The argument against Set-theoretic Ontologies models for formal ontologies is that they are inadequate since, e.g., we are not modeling any continuous universe [361]. We argue that in computer science every data is discretized. Thus continuous models are not needed. Especially, every sensor value is first discretized before it is passed to the agent. Because of that, we are going to focus on the set-theoretic approach described next.

[9]This leads to the question of identity: How much change can an Object undergo before becoming another object?
[10]Category theory formalized structures in directed graphs with labels on edges and nodes.

We now will have a look at different set-theoretic definitions of ontologies next. Again starting from a more abstract and getting more practical. Starting with an abstract definition of what entities and their relations are (Section 4.2.1). To a more mathematical definitions (Section 4.2.2, 4.2.3 and 4.2.4) ending in a conclusion in Section 4.2.5.

4.2.1. Mahrs Model of Conception

The first definition we will look at is from the **Model of Conception**. The model of conception has been introduced by Mahr [249]. The basic building blocks here are entities.

The first definition of the Model of Conception is an Entity:

Definition 26. ***Entity:***

1. *An entity is anything that is.*
2. *Any two entities are different.*

This gives us the basic building block for further investigation. We formalize this in a set E of all entities. Further entities can have relationships with each other and with them selfs, with this in mind, Mahr defines a "Relationship" as follows:

Definition 27. ***Relationship:***

1. *A relationship is an entity by which entities are related.*
2. *An entity is an element of a relationship if it is one of the entities related by the relationship.*

Since every thing is an entity, a relationship is an entity as well. This is quite general and needs a formalization. Because the natural language representation of the definitions of Mahr allows interpretation, let us look at the formalization of the model of Mahr by defining a Signature [85, pp. 14–15] like it has been done in [189, 405]:

Definition 28. ***Complex:***

1. *A complex is an entity by which entities belong to relationships.*
2. *A relationship is an element of a complex, if the entities which belong to this relationship belong to this relationship by this complex.*
3. *An entity is an element of a complex, if it belongs to a relationship which is an element of the complex.*

A complex can be seen as a bigger structure which includes concepts and their relationships. An example of a complex could be a topic, where certain concepts belong to the topic with relationships among the concepts, but other do not. In a topic, for example, some word senses could be excluded so that when we talk about the topic of bank robbery, we do not connect concepts of rivers or the sea into the complex.

The definition of a model of conception of Mahr [249] describes an abstract view on a conceptualization. This maps to natural language, where each concept can be described by a set of concepts [262]. With that, we can describe, e.g., the relation "is teacher of" in "Bob is teacher of Alice." which in itself is a set of concepts. To be able to implement reasoning upon the connectionist part of our representation of meaning we will further restrict this general description to a more technical one in the next sections; starting with the definition of an ontology by Pickert.

4.2.2. Pickert's Definition

The next definition we look at, is the one of **Pickert**. We look at this definition because it introduces a taxonomy and its properties into the definition of an ontology. Pickert[11] defines an ontology as a seven tuple $O := \{L, C, R, F, G, H, A\}$ with:

L: the lexicon is a set of lexical symbols for concepts L^C and relational symbols L^R with $L := L^C \cup L^R$,

C: the set of concepts which are used in the ontology,

R: the set of binary relations over C,

F: a function $F : 2^{L^C} \rightarrow 2^C$ which maps symbols to concepts,

G: a function $G : 2^{L^R} \rightarrow 2^R$ which maps symbols to relations,

H: a taxonomy which is an irreflexive, acyclic and transitive relation $H :\subset C \times C$,

A: is the set of axioms defined in the ontology.

Here function F and G map symbols from the lexicon L to concepts (F) and to relations (G). This mapping can be seen as a naming of concepts and relations with the symbols.

The first thing to notice in this definition is the distinction between concepts and symbols just like illustrated in the semiotic triangle shown in Figure 4.8 on Page 60. The second interesting fact of this formalization is that there is a hierarchy which lets us, for example, build a class hierarchy like in Figure 4.9 on Page 62. It should be noticed that it is unclear what the set of axioms A contains. One interpretation of A could be a set of all relations and concepts which can be extracted out of the hierarchy H. However, the definition is incomplete at this point.

Building such a taxonomy over each relation allows us to define, e.g., a hierarchy in relations like the "is-a" or "is-part" relations. Next, we look at the definition of an ontology by Maedche und Staab [245] because they define what an axiom is.

4.2.3. Maedche und Staab

Maedche and Staab [246, 245] use an ontology definition with two semiotic levels. First, the lexical level which describes how terms are used to convey meaning. Second, the conceptual level describing the conceptual relations between terms. The definition of an ontology by Maedche and Staab can be seen as a formal basis for a Web Ontology Languages (OWL), which has become a quasi-standard for the semantic web [25]. Maedche and Staab present a conceptual language with an interpretation. In their formalization, they start out with Definition 29.

Definition 29. *A **concept Language** $\mathcal{CL}$ is defined by starting from a domain $\mathcal{U}$ which is made up by individuals, atomic concept and roles, where concepts are unary predicates and roles are unary predicates over a domain $\mathcal{U}$. A concept symbol is an element of $\mathcal{A} \subseteq \mathcal{U}$ and a role symbol is an element of the subset $\mathcal{P} \subseteq \mathcal{U}$. The languages Interpretation $\mathcal{I}$ is a function that assigns each concept symbol a subset of the domain $\mathcal{U} : \mathcal{I} : \mathcal{A} \rightarrow 2^{\mathcal{U}}$ and each role symbol a binary*

[11]Taken form a technical report from `http://www.dbis.informatik.hu-berlin.de/dbisold/lehre/WS0203/SemWeb/artikel/2/Pickert_Ontologien_final.pdf` "Einfhrung in Ontologie", Gregor Pickert, last visited on 02.03.2012

relation of $\mathcal{U} : \mathcal{I} : \mathcal{A} \rightarrow 2^{\mathcal{U}\times\mathcal{U}}$. Concept terms and role terms are defined inductively with terminological axioms and using operators. $\mathcal{C}$ and $\mathcal{D}$ denote concept terms, $\mathcal{R}$ and $\mathcal{S}$ denote roles.

The concept Language from Definition 29 describes how the interpretation of concepts is mapped to a domain and range, telling the interpreter, on what entities the concepts are defined on. An example could be the formalization of a relation "IsBookedFor(Flight,Person)" where the relation has the domain of a flight and the range of a person. Meaning that we can not use the "IsBookedFor" for trains and cats. Each of these related concepts additional can have a role like "booked flight" and "ticket owner".

For Maedche and Staab an atomic concept has as interpretation a subset of $\mathcal{I}(\mathcal{C}_{atom})$, which are elements of the domain and are a result of the interpretation. Here the domain can be seen, for example, as the real world object and the concepts $\mathcal{C}_{atom}$ as a symbol denotating the objects of one kind. Let $\mathcal{C}_{atom}$ be "Apple" then the interpretation $\mathcal{I}(\mathcal{C}_{atom})$ are on the one side apples inheriting from Fruit and inheriting from a company.

Here Maedche and Staab introduce two kinds of symbols: Firstly the equivalent, which lets to concepts or roles be equivalent if their interpretation is equivalent. Secondly, the subset relation which defines one concept or role being subsumed by another one if all its interpretations are a subset of the other. The lexicon of the language can then be defined as set of terms consisting of concepts and relations of the given concept Language $\mathcal{CL}$. A **Lexicon** $\mathcal{L}$ consists of a set of terms for concepts $\mathcal{L}^C$ and a set of terms for relations $\mathcal{L}^R$ thus the lexicon is the union? $\mathcal{L} := \mathcal{L}^C \cup \mathcal{L}^R$

Next, we need a reference function linking the lexical entries to the concepts and relationships. A **Reference Function** links a set of lexical entries $\mathcal{L}^I \subset \mathcal{L}$ to concepts and relationships. Therefore $\mathcal{F}$ and $\mathcal{G}$ are reference functions with $\mathcal{F} : s^{\mathcal{L}^C} \rightarrow 2^A$ linking concepts to lexical entries and $\mathcal{G} : 2^{\mathcal{L}^\int} \rightarrow 2^P$ linking relationships to lexical entries.

Here one lexical entry can reference multiple concepts or relationships and one concept or relationship can reference multiple lexical entries. A **Core Ontology** $\mathcal{O}$ is a six-tuple $(\mathcal{A}, \mathcal{P}, \mathcal{D}, \mathcal{L}, \mathcal{F}, \mathcal{G})$ which consists of a set of concept symbols $\mathcal{A}$, a set of relation symbols $\mathcal{P}$, a set of data or facts $\mathcal{D}$ in the concept language with its interpretation function, a lexicon $\mathcal{L}$ and two corresponding reference functions $\mathcal{F}$ and $\mathcal{G}$

Upon this core Ontology, a concept hierarch can be defined. A **Concept hierarchy** $\mathcal{H}$ is defined by $\mathcal{H} := (C, D) \mid C, D \in \mathcal{A} \wedge C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. The domain of a relation and its range are defined as **Domain** and **Range**. **Domain** $d(R)$ and **Range** $r(R)$ of a relation R are defined by $d(R) := d \mid \exists\, e(d, e) \in R^{\mathcal{I}}$ and $r(R) := e \mid \exists\, d(d, e) \in R^{\mathcal{I}}$.

This allows us how to map relationships with their arguments to truth values like "isHuman(Johannes)" modeling the belief that Johannes is a human is true. Here we have to emphasize that we do not derive that the fact is false because it is not modeled.

With the addition of the domain and range of relations and the concept hierarchy, this formalization of the notion of ontology can be seen as one formal definition of an ontology. There are some unclear statements in this definition since the definition lacks to describe $\mathcal{U}^{\mathcal{I}}$ and, e.g., what atomic means. The use of a semi-standard set-theoretic definition of semantic formalizes the meaning of used terms in a sound and appropriate way.

4.2.4. Euzenat and Shavaiko

Euzenat and Shavaiko define a more practical notion of ontology in their book "Ontology Matching" [91] which is meant for the comparison of ontologies.

Definition 30. *Ontology: An ontology is a nine-tuple* $o := \{C, I, R, T, V, \leq, \perp, \in, =\}$ *Where*

C *is the set of classes,*

I *is the set of individuals,*

R *is the set of relationships,*

T *is the set of datatypes,*

V *is the set of values (with C, I, R, T and V being pairwise disjoint),*

$\leq$ *is a relation on* $(C \times C) \cup (R \times R) \cup (T \times T)$ *called specialization,*

$\perp$ *is a relation on* $(C \times C) \cup (R \times R) \cup (T \times T)$ *called exclusion,*

$\in$ *is a relation over* $((I \times C) \cup (V \times T)$ *called instantiation,*

$=$ *is a relation over* $I \times (I \cup V)$ *called assignment.*

Euzenat and Shavaiko include the typical entities of ontologies: Classes which are interpreted as a set of individuals. Individuals are with this a synonym for object and instances and represent the entities referenced by classes. Classes and individuals are connected via relations which are interpreted as subsets of the products of the domain. For the more practical part, Euzenat and Shavaiko include in their definition data types and Data values, which are individuals without identities and make up the individuals. Additionally, Euzenat and Shavaiko do not define what an individual is, neither what a variable represents.

The functions defined are the relations concepts can have in ontologies using this definition. The function $\leq$ describes the inheritance of classes, relationships, and data types. Meaning that classes can be put into a hierarchy, where the one is a specialization of the other, and the other is a generalization of the first one. The function $\perp$ defines an exclusion of classes, relationships, and data types. The exclusion states that elements of the one class, relationships, and data types are not part of the other one. The function $\in$ describes the instantiations of individuals or values. Individuals are instances of classes and values are instances of types. An example instantiation of a human could be the reader. The function $=$ describes an assignment which maps more individuals with a relation to other individuals or values. This can be seen like a relation, e.g., assigning a variable: $\text{int}\, x = 1$.

We can see that this definition is more practical and is meant for the comparison of ontologies since there is an explicit difference between classes and instances as well as between an instantiation and an assignment. Furthermore, Euzenat and Shavaiko declare datatypes and data values as part of the ontology, which is a kind of relations and entities with the purpose of comparing the individuals of an ontology.

4.2.5. Conclusion

Regarding our requirements and how we have defined an ontology in Section 3.6 we choose OWL as ontology description language.

This leaves us with an ontology with a model-theoretic semantics after Tarski [378] which need an interpretation of the language given by an ontology. An interpretation allows us to define the meaning of expression formulated in the language defined by the ontology. Tarski, e.g., does this for atomic well-formed formulas and defines that if A is a fell formed formula so is $\not A$. Formal or logical semantics define here the truth value of such an expression. For example, from first-order predicate logic an expression "A or B" is true if A is true or if B is true. However, we would probably not agree with this statement. Adding further information, e.g., by the name a A and B this could change. In our example ontology, we could formulate: "things are living (A) or are not living (B)".

All these formalizations have entities which might have some kind of relation with other entities or themselves. The relations are quite general and might differ depending on the language used to describe the ontology. We will use such an ontology to represent the knowledge and beliefs of an agent. We will describe reasoning by inference upon such ontologies and represent the semantics of a capability description with such ontologies.

In such an ontology we can state facts (individuals and their relationships) about entities (TBox) and facts about the abstract classes (ABox) (concepts and their relationships). We call a fact **grounded** if an individual exists in the ontology, which makes the fact true, meaning there is an axiom mapping the fact to true.

As a language to formulate the ontology in, we chose the OWL because the W3C Semantic Web suggests OWL [148] as a standard language for the formulation of ontologies. There are many versions of OWL varying in expressive power and domain specialization [259] and our choice, OWL 2, is a practical one since OWL 2 suffices most our requirements and the evaluation framework, as well as the basis for our implementation, is based on OWL 2. Furthermore, all requirements are met by OWL 2 as shown in Table 4.2:

Table 4.2.: Requirements for the chosen language for the representation of meaning.

Requirement	Name	Language Feature
Requirement 1	Concepts and Relations	OWL Entity [148]
Requirement 2	Unique identifiable	OWL IRI [173]
Requirement 3	Self-Reference	OWL ObjectHasSelf [173]
Requirement 4	Cyclicity	OWL PropertyChain [148]
Requirement 5	Hierarchy	OWL SubClassOf [148, p. 312]
Requirement 6	Axioms	RDF Axioms [148, p. 311]
Requirement 7	Individuals	OWL Individuals [148, p. 312]
Requirement 8	N-ary relations	OWL N-ary relations [288]
Requirement 9	Typed concepts	OWL entity [173]
Requirement 10	Attributes	OWL DataProperties [148]
Requirement 11	Relation are concepts	$\times$

The Language Requirement 11 is not fulfilled by OWL 2 because it the OWL 2 ObjectProperty is defined for individuals and not for classes. OWL 2 literals are analog to typed RDF literals and thus could be expressed by a URI. Further, there are drawbacks to using OWL as languages as well. Now here are a view things that OWL can not do: Being a fragment of first-order predicate logic, the DL cannot express the following:

Fuzzy expressions OWL does not support fuzzy concepts. This hinders us to express "It often rains in autumn".

Non-monotonicity Exceptions are hard to model in OWL. For instance, something like "Birds fly, a penguin is a bird, but penguins do not fly." has to be modeled in the fundamental class structure.

Propositional attitudes are attributes which are not supported by OWL. This is because the ontology describes facts in triples which can describe propositional attitudes like: "Eve thinks that 2 is not a prime number." (It is true that she thinks it, but what she thinks is not true.)

Modal logic like possibility and necessity like in the example of "It is possible that it will rain today." Alternatively, epistemic modalities with something like "Eve knows that 2 is a prime number." Further temporal logic is not supported by basic OWL to formulate something like: "I am always hungry.". This is the fuzzy representation handled in other work extending OWL with temporal logic or fuzzy concepts. The same is done with deontic logic allowing us to model something like: "You must do this."

Logical relations are relations which have different levels of complexity. An example of a more complex relationship is the so-called uncle problem [174]. These tasks need reasoning or an extension to OWL like a rule language.

Some ontologies are used to describe domain specific object; we will use it to describe our semantic graph and services. There are many languages describing the functionality of a service. Next, we will look at some of them.

4.3. Semantic Service Description Languages

In this section, we select a semantic service description language, which is the technical basis for our semantic representation of services. We do so by looking at the different semantic description languages. These languages are different from service description languages like the Representational State Transfer (REST) [115] or Web Service Definition Language (WSDL) [51] which describe the technical interface of a service. These technical descriptions might be referenced in the grounding of a service. The semantic description languages describe the meaning of those technical parameters. In an example of a flight-booking-service, the technical description of a service parameter "Destination" might of type `String`, e.g., BER. The semantic description of this parameter, in contrast, could be, that this parameter is a name of a destination Airport, e.g., connecting BER as Airport of Berlin to the city of Berlin.

There are many semantic descriptions of a service [290]. We have selected the most prominent once since there is a multitude of special purpose description languages, which are less relevant to the state-of-the-art of service descriptions available in practice or in golden standard data sets. These description languages can be separated into two groups: Bottom-Up and Top-down. The Bottom-Up approaches start from a technical description and build upon those with, e.g., annotations. The Top-down approaches start from a specification and describe a binding to the technical description.

Bottom-Up approaches are:

- Semantic Annotations for WSDL (SAWSDL) [205]

- Web Service Modeling Ontology (WSMO-Lite) [81, 390]
- Micro Web Service Modeling Ontology (MicroWSMO) [12]
- Reference Service Model (RSM) [241]

Top-down approaches are:

- Semantic Web Services Language (SWSL)[13]
- Web Service Modeling Ontology (WSMO)[14]
- Web Service Modeling Language (WSML) [109]
- DIANE Service Description Language (SDS) [214]
- Planning Description Language (PDDL) [132]
- A PDDL XML dialect (PDDXML) [196]
- Linked Universal Service Description Language (USDL) [18]
- Web Ontology Language for Services (OWL-S) [255]

All those description languages have their benefits and drawbacks. Selecting one of them to describe semantic services might depend on how the tools support it, what will be done with the service description, how much resources can be allocated for the description or which reasoners are available.

Conclusion

We chose the OWL-S approach [255]. In OWL-S a service is described by an abstraction of its interface and side-effects. This kind of description is called Input, Output, Precondition and Effect Description (short IOPE Description). We selected OWL-S as a description language since it is standard for service descriptions in the semantic web. This means that most modern reasoners are implemented to reason upon OWL [2]. This selection fits to the selection of the ontology description language selected in Section 3.6. In difference to planning languages like PDDL, it additionally specifies semantic information which will be used in our heuristic we want to build in this work.

The semantic description language only forms the basis for how to describe the meaning the concepts carry. The description still needs to be created by humans. To ease this task, research on the automatization of such a description will be discussed in the next section.

[12]http://www.wsmo.org/TR/d38/v0.1/ last visited on 09.09.2017
[13]http://www.w3.org/Submission/SWSF/ last visited on 09.09.2017
[14]http://www.w3.org/Submission/WSMO/ last visited on 09.09.2017

sing: **do**’ (x,[**sing**’(x)])

break: **do**’ ((x,ϕ) CAUSE [BECOME **broken**’(y)])

Figure 4.10.: Formalisation the concepts to sing and to break taken from [387].

4.4. Semantic Decomposition

In this section, we will look at mechanisms which can split up a meaning of complex concepts into less complex concepts. The decomposition is checked for its foundation in a linguistic theory, its implementation of an algorithm, and the amount of automatism it conducts the decomposition in. We start out by analyzing existing approaches.

Uson et al. [387] present a first approach on formalizing decomposition of meaning using NSM Primes. They tackle the challenge of finding an interface between the syntax and semantics by using a generalized syntax called Roles and Reference Grammar (RRG) [388]. The resulting decomposition creates a theoretical decomposition of concepts into a formal construct based on Generative Semantic. The result is a formalism which uses some primitive symbols to form languages. Those primitives are based on the semantic primes of NSM but are restricted to a few of them.

Figure 4.10 shows two examples of the description of the concepts sing and break. Here we can see that for more complex concepts (state before the colon), semantic primes like CAUSE and BECOME (stated in all upper case) are used to describe the concept to sing or break (predicated depicted in bold). The example follows the syntax of the RRG [388], where square bracket describes a nesting of syntactical structures. However, it is not explained if concepts are created in their past participle. For the simpler example of the word sing, the only information encoded is here that something has to do the singing. This is a decomposition by using the same term, which leads to a cycle of definitions. Upon this language, Uson et al. [387] describe lexical rules and morphosyntactic structures which allow a more compact description of the decomposition. The approach is highly theoretical and a methodology on how they are created not included.

Mel’čuk and Polguère [262] describes in their paper the structure of an Explanatory Composition Dictionary (ECD) regarding the Meaning-Text Theory. The focus of the paper is the semantic representation. The different elements of an ECD are described regarding an example. The formal definition of an ECD is seen as a decomposition of lexemes in a semantic network. The network is defined in a hierarchical manner so that the definition of a concept is made up of simpler concepts down to the point of a level of primes. The primes are not given as in NSM but rather learned from the decomposition. A Maximal block Principle is used to guarantee that if a more complex element is described, that it is used as a shortcut for other definitions of higher lexemes. This is an early work and does not include any automatic creation of an ECD part of any kind but rather formulates the formal basis for an ECD.

Schank and Andelson [345] describe the decomposition of sentences to derive a structured representation of its meaning. The analysis of a sentence reaches from causal connections among words, over planning and goal representations to story telling. Schank and Andelson separate the knowledge to be used for human-like understanding into general and specific scripts. Each script describes knowledge about situations, constructed or real. The scripts are made up of conceptualizations which have dependencies.

John killt Mary: John **DO** ⇨ Mary **HEALTH**(-10)

Figure 4.11.: Script in the notation of Schank and Andelson [345].

This enables Schank and Andelson [345] to create formal representations like shown in Figure 4.11.

Here we can see that the example need additional concepts like “to do” which further needs decomposition to become understandable. This leads to a set of concept which is similar to those described by Wierzbicka [406] where the found primitives are grounded in applied (spoken) languages. The dependencies between conceptualizations and their semantic description of meaning presented by Schank and Andelson [345] are more complex and do not separate specialization of described concept. The concept PTRANS, for example, could be seen as an abstraction of a set of movements (MOVE). The example of Schank and Andelson here is “to go” which shows that the decomposition of “to go” as a set of movements is not done so far.

Further, there are no dependencies to time. It could be argued that there should be a concept like TRANS which describes the abstract relation to time enabling us to describe if two events occur sequentially or at the same time. This is needed to describe the causal relation of events and with that the creation of a script in the sense of Schank and Andelson [345].

These approaches are different from the research area of ontology learning [245]. They both extract formalized knowledge from more or less unstructured data, but the approaches we look at here use some linguistic representation of meaning to break down the meaning of concepts into smaller parts.

Conclusion

In conclusion, to not just extracting facts, but doing so with a foundation in the linguistic theory of meaning is rare. Their implementation of an algorithm is even less common. For an agent to be able to acquire new knowledge about concepts, the extraction needs to be automatic and be integrable into the agent’s beliefs. None of the approaches found in the literature review describe an automatism how their decompositions are constructed. Since we want to describe an automated mechanism for a decomposition, which allows an Agent to build up a connectionist meaning representations, we will describe in Chapter 6 how we approach our semantic decomposition.

Now that we have discussed the extraction of facts and building an ontology through semantic decomposition, we need a mechanism which can use this graph representation of facts. Inspired by the human brain, the next section will discuss such mechanisms.

4.5. Activation Propagation, Activation Spreading and Marker Passing

This section describes the state-of-the-art in graph traversal techniques. Those graph traversal techniques are a way of using the connectionist representation to reason of upon the collected information. This reasoning represents the symbolic part of our approach. This section is a literature review to find an approach using symbolic information to reason upon a semantic

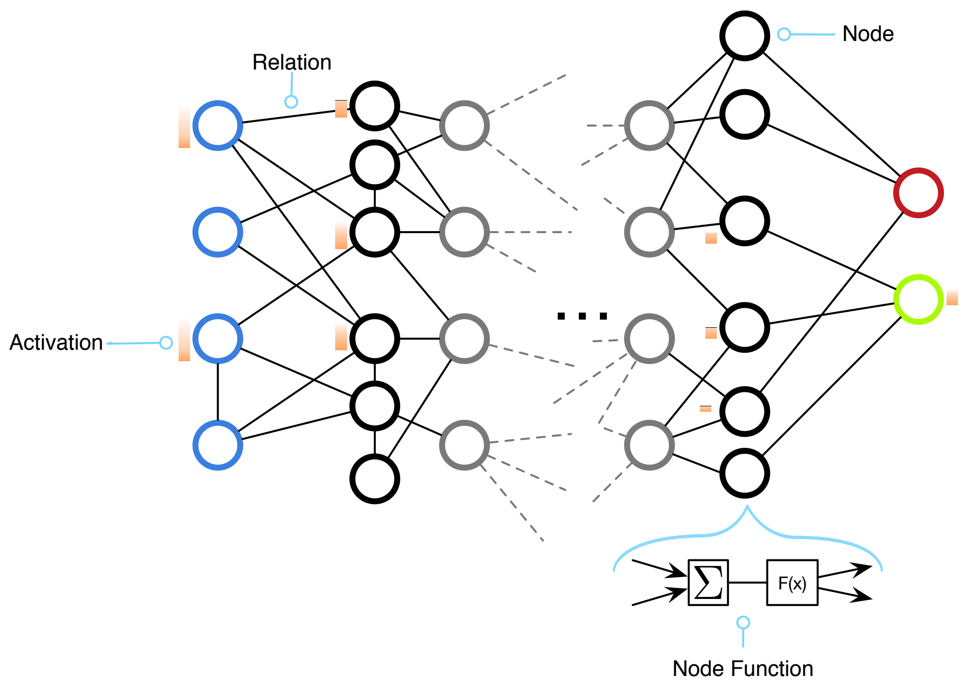


Figure 4.12.: Abstract approach of activation spreading.

graph. The approaches described in this section build the foundation upon which we have build our Marker Passing algorithm in Chapter 7 and motivate why an extension of the existing approaches is needed. Therefore we are going to analyze research done on Spreading Activation and token passing over semantic networks in combination with automatic ontology creation aka semantic decomposition. Activation spreading is a type of graph search. The basis is a graph, where information is passed from a set of start node to connected nodes. This spreading of information follows rules which guide the information through the graph. If the information is some activation, then mostly the nodes have some activation threshold like in Artificial Neuronal Networks [80].

Figure 4.12 shows the abstract approach of activation spreading. Here the set of start nodes (blue) carry activation (orange box) to adjacent nodes. Each node has a node function which decides how to propagate its activation to its neighbors. The result of the activation spreading can be the activation of specific nodes (in Artificial Neuronal Networks called output-layer) or the activation of the network. Figure 4.12 shows the activation of two interpreted nodes in green and red for an exemplary classification task. If one node is more activated than the other, we need some interpretation for the result.

Using examples from nature, which seem to work well as a prototype for algorithms in Artificial Intelligence, has become a standard approach. Since the dissertation of M.R. Quillian [314] about “Semantic memory” at the Carnegie Institute of Technology in 1966, the idea of representing knowledge in a network and using the connection in the network, has spilled to the Artificial Intelligence community. Multiple questions have been subject to research like the structure of the network [59] which has been refined from Collins and Loftus [58], how similarity models over features of concepts can be determined [362] or how the connection of concepts is influenced by episodic memory [316]. Rogers et al. [333] further extends the features from which the semantic knowledge is derived to perceptual features. Up to modern neuroscience which maps how the human brain organizes knowledge [301].

All this research has influenced the theory of Artificial Intelligence. Since we are going to use such an approach in our artificial representation of meaning, we discuss its connection to our approach next: Creating two theories of how Artificial Intelligence can be formalized: "Logical versus analogical or symbolic versus connectionist or neat versus scruffy" like the title of the paper of Minsky [273] suggests. The first one thinks that symbols carry meaning and that the use of logic like in first order logic can represent intelligence and if the intelligence is not sufficient, then the logic needs to become more complex. The second one thinks that the meaning of things is represented in the connection a concept has with other concepts. Our approach emulates the one of Minsky by trying to connect the both sides and creates a spreading activation mechanism over a semantic network with the symbolic logic to steer the spreading activation. This mechanism is used to enable an AI planner to choose actions of a plan wisely; meaning we create a heuristic to improve the AI action selection step of an AI planning algorithm in regards to precision a recall [97].

Liu et al. [239] use activation spreading over a semantic network, build from a mix of domain ontologies and WordNet resources, to identify extension point for semi-automatic ontology refinement. The analysis of Liu et al. [239] is limited to nouns only.

Marker Passing is a more general term for activation propagation or activation spreading, where more general information can be encoded into a maker than just its activation. We are going to use the more general term Marker Passing from now on to describe that kind of approaches. *P. Maes* [247] uses activation spreading for action selection in a task network. Maes describes an additional parameter of her Marker Passing which is a mean level of activation, regarding the entire network. *Fahlman* [92] uses Marker Passing on semantic networks with a focus on massively parallel computing. Fahlman was the first to use the markers to reason more complex relations on a knowledge graph. By combining Marker Passing with a special type of edge, a so-called cancellation edge, reasoning can be done in a more complex manner.

There is much different application of this Marker Passing or activation spreading technique. One of them is Wang et al. [398], for example, use such an activation spreading for ontology matching. Another area of Natural Language processing is Fischer [119] like the querying of an ontology.

Anderson [10] proposed a Theory of memory using Marker Passing to model delay behavior of human brains. The idea behind the work of Anderson is that the semantic network and the distance of two concepts in its encoded semantic information can be structured by measuring how long it takes for a token to reach a given concept from a given start concept. Measuring the time is such a graph, abstracts the step count. Which is why Anderson's work has been subsumed by other more general approaches like the one of Thiel and Berthold [379].

Thiel and Berthold [379] proposes two-word similarity measures which use activation spreading as a basis: node signatures similarity and node activation similarity. The first one is based on the comparison of norms of the velocity vectors of the activation spreading and can be interpreted as structural similarity. The second one compares the accumulated activation and can be interpreted as a spatial distance measure. This approach is evaluated through the analysis of Schools-Wikipedia[15] (2008/09) as a data set to firstly find nodes which are structurally similar to a query node and secondly to find closely connected to the query node. The resulting graphs are displayed in a centrality layout and analyzed manually for structural similarity.

[15]http://schools-wikipedia.org/ last visited on 09.09.2017

Next, we will discuss a formalization of the activation spreading in the form presented by Thiel and Berthold [379] which allows us to formally grasp activation spreading. They use a weighted undirected graph $G = (V, E, w)$ The nodes (V) of the graph have three functions:

- Input function: combining all activation passed to the node in the activation step.
- Activation function: determining if the node is activated at time t by the weighted sum of activation passed through the incoming edges [379, eq. (1)]:

  $$a_v^t = \sum_{u \in N(v)} w(u, v) a_u^{t-1}, \forall v \in V \tag{4.1}$$

  With $N(v) = \{u \mid \{u, v\} \in E\}$ representing all neighbors of node v and a_v^t being the activation of node v at time t and $w(u, v)$ being an element of a weight matrix W with $w(u, v) = 0$ if $(u, v) \notin E$, stating the activation passed from node u to node v.
- Out function: determining the outgoing activation depending on the activation level of the node.

The activation vector is calculated if the weights are seen as a matrix $W_{i,j}$ and the activation function is seen as [379, eq. (2)]

$$a^t = W^t a^0 \tag{4.2}$$

where we can calculate the activation at actiavtion step t be iterativly activationg the network based on the start activation a^0. The activation vector then is normalized by its euclidian length with each iteration [379, eq. (3)]:

$$a^t = \frac{W^t a^0}{\parallel W^t a^0 \parallel} \tag{4.3}$$

where the direction of the vector is not influenced by the normalization because the convergence of the principal eigenvector v is given through $\lim_{t \to \infty} a^t = \frac{v}{\|v\|}$

This kind of activation as the convergence of activation towards the eigenvector of the weight matrix independent from the query concept initially activated. Finally having the result that pure activation spreading leads to a stable state after enough activation steps. This effect is discussed in detail by Berthold et al. [26].

The speed with which this convergence is reached, on the other hand, depends on the query node initially activated. The velocity δ is measured for each activation step by [379, eq. (4)]:

$$\delta^t(v) = \begin{cases} \varnothing & \text{, if } t = 0 \\ a^t(v) - a^{t-1}(v) & \text{, else,} \end{cases} \tag{4.4}$$

Here the $a^t(v)$ is the activation in activation step t and $\vec{0}$ is a zero vector. With $a^t(v)$ being a function of the collected tokens like e.g. the sum of all activation passed by one token. The velocity of the query node at the start is one:

$$a_i^0(v) = \begin{cases} T_{start} & \text{, if } i = v \\ \varnothing & \text{, else,} \end{cases} \tag{4.5}$$

With T_{start} being the start activation of node v.

Thiel and Berthold [379] additionally introduce a post processing step, in which they normalize the activation of a node by its number of edges the nodes have. The normalisation is done by the division of the “weighted degree Matrix” D which is a diagonal matrix with each non zero element being $d_{i,i} = \sum_{j=1}^{n} w_{i,j}$. The normalized activation then is calculated as [379, eq. (7)]:

$$\hat{a}_i^*(v) = D^{-\frac{1}{2}} \left(\sum_{t=1}^{t_{max}} \alpha^t a^t(v) \right) \tag{4.6}$$

This normalization has the effect that well connected node are not over estimated by the activation. With the decayed activation $a_i^*(v)$ multiplyed with an decay $0 \geq \alpha \leq 1$ [379, eq. (6)]:

$$a_i^*(v) = \sum_{t=1}^{t_{max}} \alpha^t a^t(v) \tag{4.7}$$

With t_{max} being the maximum amount of activation steps done before evaluating the result. Hence t_{max} represents the termination condition of this activation algorithm. This formalization of activation spreading with its double valued activation is not able to encode further information into the activation.

The approach of Fahlman [92] extends this activation spreading to Marker Passing but describes its Marker Passing with a two-way linked list M for each marker, representing all nodes which are marked with this marker. The Marker Passing is then done by setting and removing elements in those lists. With that Fahlman is able to create queries which state: “if the node holds marker X, then...”. With that, he can create queries based on multiple markers, but can only encode the presents of a marker type, no further symbolic information because the information encoded depends on the order in which the markers are put on the semantic graph.

Conclusion

In conclusion, the current Marker Passing approaches can carry more information than the basic activation, but we were unable to identify an approach which uses the symbolic information for reasoning, except the work of Fahlman [92] which has special purpose symbolic information encoded on the markers. Fahlman encodes the edge information onto the marker, to describe which edge should be traversed. This is not enough since not only the edge which has to be traversed my change, but the reaction of a concept concerning markers may change as well. All of those algorithms are thought for the passing of markers and an analysis of the resulting markers or the marker distribution. Furthermore, none of the approaches used semantic information on the markers except Anderson [10] more than 30 years ago. The structured analysis of the use of Marker Passing did not reveal any use of more symbolic information on markers since then.

Additionally, we can see the analyzes done on the Marker Passing so far. The main concern seems to be an application of Marker Passing for some use case and not the Marker Passing itself, its benefits, prerequisites or limitations.

4.6. Service Planning

This section surveys the state-of-the-art of service planning approaches. The goal of this survey is to find planning algorithms which make use of semantic information provided by semantic service descriptions to build a heuristic. We do this survey to make sure that we are not reinventing some problem-solving mechanism, which uses semantic heuristics for problem-solving. The heuristic is our point of application; making one step towards filling the performance gap between static and learning planners [112]. We try to narrow this gap by creating a service planner using semantic heuristics like it is done in service composition techniques. The related work on service composition is well researched, but there are only a view approaches using planning as composition mechanism. The literature provides a set of different service composition approaches and concepts.

Those planning and composition approaches are analyzed to their **composition type** describing if they are (semi-) automatic or manual. Furthermore, we analyze how dynamic an approach is (**Adaption**) and on which **Basis** the planning builds up on, e.g., a Workflow or heuristic search.

Heuristics can be build by simplifying the given problem [304]. Those implications are for example used by Fast-Forward-Planner by removing the delete list of service effects [171]. Because of the open world assumption, this does not work in service planning. Other heuristics guide specific problems which then can not be used in problem independent (sometimes called general purpose) planning [304], because they do not adapt to, e.g., changes in the goal or to new services. Thus we need another way of gathering information about the available service and the goal. In this work, this information is the meaning of the concepts used to describe, e.g., the service or goal.

One service composition approach is WSPlan, developed by Peer [305]. WSPlan uses a knowledge base and services described in WSDL extended by semantic annotations in a PDDL syntax. The knowledge base and the annotations of web services are transformed into PDDL documents. It uses an online planning method for service composition which means that the planning and execution are interleaved.

There are other solutions which also transform service descriptions into another description language for service composition. The solution developed by Okutan et al. [293] transforms OWL-S descriptions of services into the Event Calculus framework in which actions and their effects are expressed, the solution by Kuzu and Cicekli [216] transforms OWL-S descriptions into PDDL and the solution by Sirin et al. [360] transforms OWL-S descriptions into the SHOP2 domain to use SHOP2 as an HTN planner.

Rodriguez et al. [331] state the Quality of Service (QoS) parameters are a significant research challenge which is not reflected in the approaches found in our state-of-the-art analysis. Having QoS parameters in a service description and having planner use this information would be a way to create an optimization problem out the service composition problem. Markou and Refanidis [254] analyze nondeterminists methods and whether a heuristic is used or not. Neglecting which information the heuristic build upon or how it is used.

Zou et al. [430, 431, 432, 433] analyze the performance of different planning methods, but neglect any semantic information from the service descriptions, which leads to a focus on approaches which are based on the planning language PDDL. With the translation from OWL and SWRL to PDDL, most semantic information gets lost and with that the ability of the planning approach to create a semantic heuristic. Another service composition solution based on such a translation is OWLS-XPlan, developed by Klusch [196]. It transforms OWL-S descriptions of services into PDDLXML, an XML dialect for PDDL. For service composition, it uses a combination of a Fast-Forward-planner and an HTN planner.

Rodriguez-Mier et al. [332] neglects nonfunction parameters but uses a general heuristic in an A* search. This heuristic is build so that the evaluation function $f(n) = g(n) + h(n)$ counts the already executed service in state n in $g(n)$ and the still needed service execution to the goal in $h(n)$. Creating the heuristics includes creating a service dependency graph, in which the mappings from out- to in-put parameters are the edges between the service calls, which are represented as nodes. With that, the services form layers in the dependency graph and allow an estimation which layers the current service to call is located in. With this heuristic, the planning minimizes the number of web service calls. Calculating the service dependency graph includes solving the planning problem, making the heuristic useful in the path selection, if multiple paths from start to goal state are found.

Meyer and Weske [264] use a similar heuristic: The number of service calls to achieve the goal state. The idea is similar to Rodriguez-Mier et al. [332] since graph plan is used in a relaxed planning problem, by ignoring the negative effects. Then the service dependency graph is used to estimate the number of service calls to the goal.

Hoffmann and Nebel [171] neglect the delete list of service effects as well to form a fast-forward planner. This is possible because a PDDL effect consists only of deleted and added axioms. It can be argued that with the open world assumption of OWL such a heuristic is not possible since no delete and add list can be identified. Klusch et al. [201] use the heuristic of Hoffeman and Nebel and use a breadth-fist search in case of heuristic equivalence of to states. Mediratta and Srivastrava [260] also use the heuristic of Hoffmann and Nebel to buidl an AND/OR-graph. The heuristic than takes the minimal cost of an OR-branch and a sum of the cost in case of an AND branch.

Bonet and Geffner calculate their heuristic by taking each fact of the goal state and searching backward to the state space until they find a service which fulfills this fact. The functions *getPrecondition*() and *getEffect*() get the fact describing the service. *Services* describes here the set of all available services. Selecting the maximum of all the path reaching to the service which fulfills the preconditions of the current service guarantees an admissible heuristic. This heuristic uses the semantic information of precondition and effect but neglects the input and output. Additionally, they ignore any semantic similarity and only check for equivalence of the facts in precondition and effect.

Akkiraju et al. [6] present an approach called SEMAPLAN, which uses WSDL-S[16] service to create a service description. The focus in SEMAPLAN is on the combination of in- and output parameters through ontology matching using WordNet or domain specific ontologies. The planning is done with in iterative forward search using all services which are semantically close to the wanted ones. The algorithm terminates if all facts in a goal are created through an effect of a service or if there are no more service to try.

Cavallaro et al. [47] propose a dynamic service composition framework, called Connect, which focuses more on the software development perspective of a service composition. The goals of the service composition are formulated as "expectations" and describe requirements of the services used. The Connect framework then searches for services fulfilling the requirements during runtime. Goals are formulated as operation calls. Thus the planning at runtime consists of finding a service providing the operation and connecting they're in- and output.

Fernandez-Olivares et al. [113] describe a system call SIADEX which can use multiple planning methods form the domain of Hierarchical Task Network (HTN) to create service compositions. The translation from OWL-S is here to an HTN. A goal is can be formulated in OWL-S

[16]see https://www.w3.org/Submission/WSDL-S/ for more details. Last visited on 09.09.2017

syntax and is translated to an HTN goal. The result of the HTN planner then is translated back into a service composition. A benefit of SIADEX is that it includes a monitoring component to measure plan failure and trigger a replanning at runtime. The benefit of using HTN as planning problem formalization is that SIADEX can execute services in parallel.

Kuter et al. [215] describe an HTN based planning called ENQUIRER. Like in the SIADEX approach the OWL-S descriptions are translated into an HTN, the goal is translated as well, and an HTN planner is used to create a service composition. Novel in the ENQUIRER approach is that during the planning, services can be executed to gather dynamic information for the planning process.

The Eclipse-plugin WS-Engineer proposed from Foster et al. [121] is a manual approach to service composition. A service composition is described in the Business Process Execution Language (BPEL) and UML Message Sequence Charts. As a basis for an execution of a service composition, Foster et al. focus on the development and distribution of service compositions with execution details like authentication.

Wang et al. [396] propose a web service composition based on Markov Decision Process (WSC-MDP). The approach is based on an explicit description of service alternatives and the use of reinforcement learning to learn the best alternative during runtime. The service composition is modeled as Markov Decision Process, with state transitions modeling service calls. The reinforcement learning then defines the transition probability, which describes the availability or performance of the service. Through this learning, better performing services have a high execution probability. The used languages for service description and implementation details are omitted by Wang et al.

Rao et al. [321] describe an approach called "Web Service Synthesis Architecture" (WSSA) which uses linear logic theorem proving to create service compositions. The WSSA approach translates OWL-S service descriptions into linear logic axioms and goals in theorems. A theorem prover is then used to find a proof for the given theorem with the given axioms. If a proof is found, then a service composition can be derived which fulfills the goal. During the prove finding process semantic reasoning is used to identify type hierarchies which are formalized as implications and ease the proving. The paper does not describe how the Effect and precondition are translated, e.g., from SWRL to logical axioms. The analysis of Rao et al. is restricted to a level of input, output, precondition, and effect as a whole.

El Falou et al. [87] describe a decentralized multi agent approach to handle the ever growing amount of services. Concerning the heuristic used, the approach assumes that the relevant services have been selected beforehand. The plan is separated into partial plans which are handled by different agents. The result of the partial plan is then combined to a service composition reaching the given goal. The approach is quite abstract and misses a description of the mechanisms of finding the services or how to evaluate their goal archival.

Papadopoulos et al. [299] describe a decontrolled multi-agent service composition (DMASC), which implements an self-organizing distributed service composition. Prerequisite is that the service provides logic to be part of the decomposition where Papadopoulos et al. talk of a coalition. Each agent then searches for predecessor or predecessor by matching in- and outputs.

Hatzi et al. [159] describe the PORSCE II service composition approach which also translates OWL-S description int PDDL. PORSCE II then uses different external planners to solve the planning problem. The focus of Hatzi et al. is on the specification of the start and goal state, for which they provide a user interface. The user interface also provides a visual representation of the service composition and allows a manual adaption.

Fujii and Suda [126, 127] describe a service composition system called Component Service Model with Semantics (CoSMoS). The focus of CoSMoS is to create goal descriptions from natural language. The goal is described in a logical predicate, which they derive from natural language requests. CoSMoS uses semantic analysis to verify the used service to fit the context in which the services are called.

Fanjiang and Syu [105] use genetic algorithms for service composition without any heuristic [226]. Here the logical structure of precondition and effect rules are evaluated to create a correct plan. By this logical analysis, a causal links are created. This leads to a valid plan but does not create a heuristic.

Luca and Massimo [339] base their approach upon the BDI agent model. The approach uses the beliefs of the agent about goals and capabilities to analyze a goal hierarchy by ordering the goals in an AND/OR-Graph and moving down the graph towards the leaves to see which goal can be reached. For each step in this search, a "goal-capability deliberation" is done by simulating the execution of a classical forward search, where the amount of used service to reach a goal is then used as a heuristic value. Since this approach uses planning on subproblems as a heuristic, the algorithm might be used for one-shot-planning but does not describe how it uses the semantic information available. Further, they do not describe how the "$select_{capabilities}$" function is implemented, which would tell us how the approach selects service to be executed and they do use some kind of "domain-specific utility function" which lets them rate the services. Because of these two reasons this heuristic is not a general purpose heuristic.

Oh et al. [292] describe a Web-Service Planner augmented with A* algorithm (WSPR*) based on WSDL services and OWL parameter descriptions. The main contribution of this paper is a heuristic based on the QoS parameters of the services. The approach analyzes in input an output of services but neglects preconditions and effects.

Wang et al. [397] propose an automated web service composition based on uncertainty execution effects which can handle uncertain and contradicting service effects. The planning used to create the service composition is the GraphPlan algorithm, which is extended to be able to support service with multiple effects. The approach translates OWL-S to STRIPS and a STRIPS planner is used to solve the planning problem.

Wagner et al. [393] describe a QoS-aware automatic service composition by applying functional clustering to the functional description of services and their QoS parameters. Services are clustered by their functionality, and those clusters are combined to create a service composition. A cluster represents service alternatives. The service composition neglects the variable bindings for parameters, and the evaluation of precondition and effect is not described.

Rodriquez et al. [331] provide a survey on automatic service composition and AI methods in software development. The existence of heuristics in the surveyed planning approaches is checked, not how they are used or which information the heuristic is based on.

Conclusion

In conclusion, we can say that there are many approaches which use planning for problem-solving, but only a few of them use semantic information in search of the state space (see Table 4.3). Semantic information is used for service composition but is mostly static. There are

Table 4.3.: Comparison of Service Composition approaches regarding their composition type, their adaptiveness and the basis they build their solution upon. The approaches are classified to the heuristic they use: Here “ff” stands for fast forward (problem relaxation), “dom” stands for domain specific heuristics and “×” stands for no used heuristic.

Approach	Composition type			Adaption	Basis		
	automatic	semi	manual	dynamic	Workflow	Planning	Heuristic
SEMAPLAN [6]	×	✓	×	×	×	✓	sem
Connect [47]	×	×	✓	✓	✓	×	×
SIADEX [113]	✓	×	×	✓	×	✓	×
ENQUIRER [215]	✓	×	×	✓	×	✓	×
WS-Engineer [121]	×	×	✓	×	✓	×	×
WSC-MDP [396]	(✓)	(✓)	(✓)	✓	✓	×	dom
WSSA [321]	✓	×	×	×	×	✓	sem
Falou et al. [87]	✓	×	×	✓	×	✓	×
PORSCE II [159]	✓	✓	×	×	×	✓	×
CoSMoS [127]	✓	×	×	✓	✓	×	sem
Zou [432]	✓	×	×	✓	×	✓	dom
Kuzu [216]	✓	×	×	✓	×	✓	×
Wagner [393]	✓	×	×	✓	×	✓	×
WSPR* [292]	✓	×	×	✓	×	✓	dom
UDCP [260]	×	✓	×	✓	✓	×	dom
DMASC [299]	✓	×	×	✓	✓	×	×
OWLS-XPLAN [196]	✓	×	×	×	×	✓	ff

also service composition solutions which use multi-agent systems for load balancing. One approach which uses a multi-agent system is the approach by El Falou et al. [87]. There is one central agent which receives a request from a client which includes the initial and goal state. It forwards the request to service agents, each managing a group of web services. All service agents compute a local partial plan and send it back to the central agent. The central agent merges the partial plans together to obtain a global partial plan. Then it applies it to the initial state to obtain a new state and sends a new request based on the new state to the service agents. They, in turn, compute a new plan iterating until the goal state is reached. A similar approach is DPAWSC (Distributed Planning Algorithm for Web Service Composition) which also uses a multi-agent system for service composition.

Table 4.3 shows the conclusion of the literature review on semantic service composition approaches using AI planning. The classification has been done by the following three aspects:

Composition type, which describes the degree of automatism: manual means that the approach supports developer integration, semi-automatic means that the approach is not able to create service composition automatically and automatic means the approach is able to create full service composition automatically.

Adaption, describes the degree of dynamic adaption points the service composition supports. Dynamic means the composition is formed at runtime, static means the composition is created during design time and the selection of executing services might be at runtime.

Basis, describes which information the composition is formed on. The description of a workflow normally means a human was in the loop. If planning is a basis for the composition, the composition might be simpler in structure (e.g., fewer loops or if-then-else constructs) but the developer is supported by an AI planning on creating the service composition. Heuristics describes which kind of information was used to use an informed search while creating the service composition. Here the options are “semantic” for semantic problem analysis, “ff” for problem relaxation, “dom” for domain specific heuristics or “$\times$” if no heuristic was used.

The overview reveals that there exist different, mostly domain specific approaches for solving the task of service composition with AI planning. The planner used in this work will not transform service descriptions into a PDDL like description language. Instead, the planning is done directly on the results of the semantic Service Matcher and semantic service descriptions. This requires the creation of sound heuristics and backtracking from dead-ends in the planning process.

Part III.

Approach

5. Abstract Approach

This section describes how I will approach the creation of an artificial representation of meaning. Described in Figure 1.2 my approach has two parts: The first part is the connectionist representation of knowledge. The second part is the symbolic reasoning. As a theoretical foundation I use the Natural Semantic Metalanguage (NSM) (see Section 3.10) theory. The NSM theory states that from the set of semantic primes, explanations can be created to describe more complex concepts. I transfer this “bottom-up” approach, to a ”top-down” approach, start with the complex concepts, facing the problem to find a way to describe complex concepts using less complex concepts down to the semantic primes of NSM. This process is called decomposition and identifies the task of describing a concept using less complex, but related concepts. Hence, the question that is approached with this task is: *Which concepts are related to the concept we are interested in?* Figure 5.1 describes the knowledge representation part of Figure 1.2 in Section 1.5 in more detail. This part represents how the connectionist part of my representation of meaning is created. The knowledge acquisition is inspired by the way we adult humans learn a new concept: If we encounter a new word, we look it up in some resource like a dictionary and connect the there found explanation with our current knowledge.

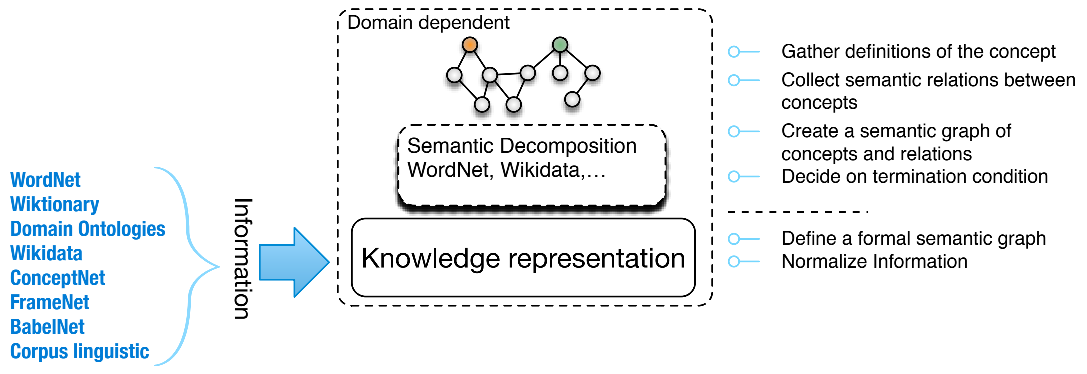


Figure 5.1.: Abstract description of the knowledge part of this approach.

Figure 5.1 describes which information can be used in the decomposition as input, e.g., WordNet or Wikidata. Most of these knowledge sources are general-purpose dictionaries. However, the input for the decomposition could also be a domain-specific ontology, giving facts about the concepts needed. This creates a domain dependent representation of knowledge.

The so created knowledge can be seen as the belief of an agent and the process as belief update. The main steps of this decomposition are the gathering of new information which includes integrating new concepts and their relations into the agent’s beliefs. Here one problem is to describe when to stop with this information gathering process to avoid integrating unneeded information.

The knowledge presentation is the collected information in the form of a semantic graph. Naturally, since the approach collecting connectionist information, we might represent this knowledge as a graph. In this first part, I will describe my graph formally and show how I formalize the fuzzy semantics of natural language with its concept and relational hierarchies, semantic relations like Synonyms and Meronyms and more complex relations like "giving advice". In this, I normalize the information gained from natural language resources to a strict formal model. Further details for this semantic decomposition are described in Section 6.

Figure 5.2 details the second part of Figure 1.2 in Section 1.5. Here the reasoning part of this approach is described in more detail. This part has as input the connectionist representation of meaning from the first part and a problem specific question. These two component are then used to specify the parameters of a Marker Passing algorithm. These parameters include symbolic information encoded on the markers.

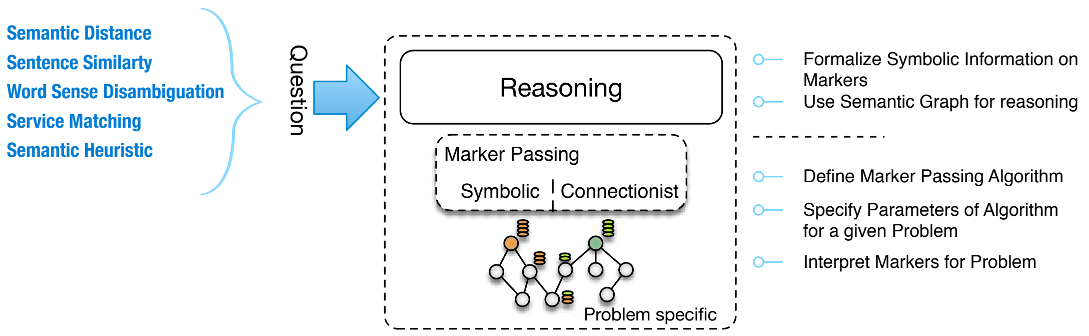

Semantic Distance
Sentence Similarty
Word Sense Disambiguation
Service Matching
Semantic Heuristic
Question
Reasoning
Marker Passing
Symbolic
Connectionist
Problem specific
Formalize Symbolic Information on Markers
Use Semantic Graph for reasoning
Define Marker Passing Algorithm
Specify Parameters of Algorithm for a given Problem
Interpret Markers for Problem

Figure 5.2.: Abstract description of the reasoning part of this approach.

This symbolic information is then passed through the graph where each node and edge can change the information on the marker depending on their interpretation. The resulting marker distribution and the information encoded upon them is used for answering the given questions. If the first part represents the knowledge of an agent the second part could be seen as the process of thought over this knowledge. This approach on semantic reasoning is further detailed in Section 7.

The example problems listed in Figure 5.2 as input for the reasoning is then tested in the experiments in Section 9. These example applications are selected because they all show some understanding of the meaning of concepts. The task to determine a semantic distance of two words, for example, shows that without context, we can still estimate if two concepts are similar or not. The task of Word Sense Disambiguation as another example shows if we can select the right word senses of words given a linguistic context.

The so created representation of meaning reflects a cognitive model of humans in the way that the semantic graph represents an abstracted physical part of our brains, and the Marker Passing represents the electronic or chemical parts of our brain[315]. Thus the one side represents learned knowledge (concepts) the other processes like thought to create coherent reasoning [295].

As illustrated in Figure 5.3 the creation of artificial meaning as defined in Definitions 21 and 22, starts with building a model for all meanings of a concept by decomposing it. Here Figure 5.3 focuses on temporal dependencies on the approach described in Figure 5.1 and 5.2. This leads to a semantic graph representation (Ontology) of its denotation that represents the connectionist knowledge representation of meaning. Such a decomposition is done to create a

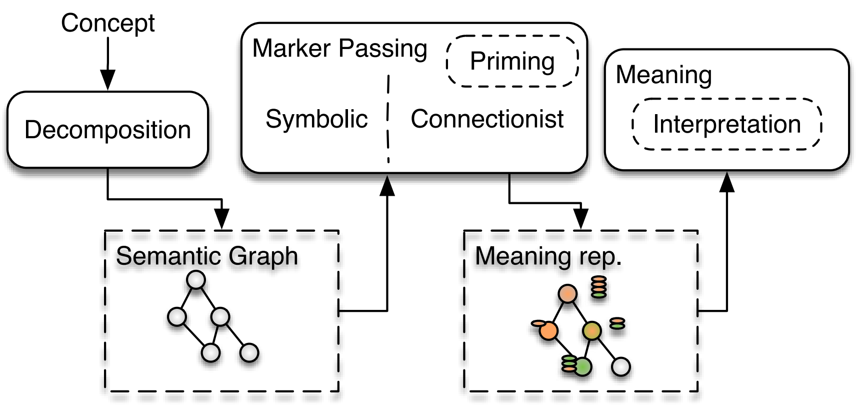


Figure 5.3.: Abstract approach to represent artificial meaning.

semantic graph [98]. One challenge here is to select the right definitions of the word[1] from the utilized data sources to be used in the decomposition.

This semantic graph is used to pass markers through to relevant concepts.[2] The information carried by the markers is denoted by the different colors of markers in Fig. 5.3. The marker (represented as chips next to each concept in the depiction) might carry symbolic information that steers the Marker Passing. To be able to react to different markers, each concept in the semantic graph has an interpretation function reflecting its behavior. The interpretation function inflects how the concept processes incoming markers, how outgoing markers are passed on to other concepts and whether it is activated. In this way, e.g., a "NOT" concept passes its markers to the next concept so that this one activates its opposites (in linguistics named "antonyms"). Since semantic relations like synonym and antonym relations have different meanings as well, the relation interpretation function allows specifying how a relation passes on markers. In this way, symbolic information like temporal logic can be encoded in the graph. One challenge at this step is the amalgamation of the connectionist representation in the semantic graph and the symbolic representation provided by a concept and edge in the semantic graph.

During the activation, we can influence how the amalgamation of symbolic and connectionist representation of meaning is contextualized. By activating the right concepts out of context, the Marker Passing will activate different concepts in the semantic graph and thus contextualize the representation of meaning. Here the selection of parameters and concepts to activate is challenging. Finally, we need an interpretation of the output of the Marker Passing to extract the represented meaning.

If we want to extract the meaning of the word 'Bank' in a text discussing the financial crisis, the Marker Passing will have more markers on 'Bank' as a financial institute then on the seating accommodation. This is because the activation of contextual concepts will probably activate concepts like money, accounting, currency or equivalents from the text during the Marker Passing. Through this combination of activated concepts and interpretation functions, we create a pragmatic representation, which leads to a context-dependent interpretation of the knowledge encoded in the graph.

[1]This challenge is related to the word sense disambiguations and is one reason for the need of contextual information during the decomposition.

[2]Marker passing subsumes activation spreading since the classical activation spreading can be modeled with a marker that carries the activation level as a numeric value.

This part is organized as follows: In Section 6 I will describe the semantic decomposition and Section 7 describes the Marker Passing algorithm. I then evaluate my approach in Part IV of this thesis. In Part IV I test the reasoning upon the newly learned facts by letting the AI we created solve different example problems like the estimation of a word and Sentence Similarity or Word Sense Disambiguation problems. The implementation of this approach is described in Section 8 consisting of two parts: The Semantic Decomposition and the Marker Passing. The short introduction should give the reader the ability to use the implementation of this approach and apply it to other problems.

6. Semantic Decomposition

Semantic decomposition is a process to take apart the meaning of complex concepts into multiple less complex ones. This section describes the semantic decomposition proposed in this work. I will look at how knowledge sources like WordNet or Wikidata are used to gather information about the target concepts. Then I will look at the result and analyze the outcome, and its usefulness for our creation of a heuristic.

The goal of the decomposition is to collect all facts about a Concept of Interest (COI) needed to form an accurate representation of the meaning of these concepts in a given context. Further, the goal of this decomposition is specialized to an artificial audience, our agents. The decomposition has the purpose of describing the connectionist part of meaning by connecting a concept with other concepts, which are simpler, or less complex in the notion of the NSM theory [410]. How this is done will be discussed in this section.

The method in which the decomposition is created is the following: We collect all information available about a COI. Then I filter this information for its relevance to remove unnecessary facts by, e.g., removing stop words or stopping at semantic primes. The collection of information about a COI is done by looking up the COI in different information sources, e.g., dictionaries. A decomposition represents the paraphrases which describe a concept [328, p. 213]. Riemer [ibid.] describes in the decomposition as it has been practice in dictionaries in the following way: "the meaning of the definiendum is recast in terms of several defining terms which, if the definition is to be explanatorily successful, must each in some sense be simpler than the definiendum or, at least, already understood by the definition's user." Here the term simpler is still too unspecific to understand how we can describe meaning, which I will elaborate closer next: In the theory of NSM, the simplest concepts are those that make up the semantic primes. I build my approach to this theory by assuming that the meaning of these primes is known. Based on those primes I can describe more complex concepts only by using primes. These are then less simple than the primes itself. I then, define the meaning of more complex concepts by describing concepts which I was unable to describe by only using primes and the first set of more complex concepts [410, p. 7]. I repeat this process until all concepts have a definition. The so created hierarchy builds my scale of complexity (simplicity in anti-proportionally).

There are two reasons for inverting the direction of the theory of NSM in my decomposition: First in the theory of Wierzbicka [406] and Goddard [145] the idea is that upon semantic primes every concept of a natural language can be described. This forms different complexity levels, where the semantic primes are the least complex. Since we are looking for an explanation of a complex concept, searching from the bottom up (from the primes in complexity upwards) would result in a big search space since we do not know which concepts are needed, which means we would need to build all concepts which are solely described by semantic primes and build our way upwards to the COI. Narrowing the search space, the decomposition tries to find a decomposition of all concepts defining the COI, top down.

The second reason is: Semantic approaches often struggle with their use in AI since the appropriate knowledge or needed cognition is still not accessible for computers. In combination with the Natural Semantic Meta Language (NSM) theory, a decompositional approach seems promising in representing meaning for AI. Decompositional approaches describe lexical content by splitting it up into smaller parts [328].

Using the theory of NSM, I can use the semantic primes to describe all other concepts in a natural language from bottom-up, to building the meaning of complex concepts. The fundamental idea of the semantic decomposition for AI is: if the meaning of the semantic primes is given, I want to enable my agent to build the meaning of more complex concepts on its own by first creating a decomposition and then using the semantic primes to give the complex concept meaning. The decomposition is seen as a way of an agent to self-explain new concepts by incorporating its definition or some other explanation into its beliefs. This means that by looking up explanations of unknown concepts, e.g., in a dictionary, the agent extend his semantic graph to include the unknown concept.

In the Cambridge Dictionary[1] **to explain** is defined as follows: 'to make something clear or easy to understand by describing or giving information about it'. By examining this definition, we notice, that explaining is the act of giving information about a COI to an audience with the intend to foster both the knowledge and the understanding about the COI. The explanation given here seems to be a description of the relations the COI has with other concepts. This explanation has to consist of concepts already known to the audience, else the explanations of those concepts need to follow recursively. This means that the explanation can be context dependent because if other concepts are known by the audience, the explanation of the same fact about a COI can change. This can be seen in the way we explain the concept of numbers to children in second grade and during a course of algebra in mathematic studies. Each new fact about concepts gives the agent a new connection point, which might allow it to connect the new concept to its beliefs.

Some other approaches have been investigating decompositions and how semantic primes influence the expressiveness and the reasoning upon such semantic graphs. Schmidtke [349] represents such meaning in logical concepts and proposes a reasoning with primes for qualitative reasoning on the example of spatial concepts. For this Schmidtke extends the first order logic with primes, e.g., spatial reasoning. The extension from Schmidtke are pragmatic but are limited to his use case. Ferrand [114] analyzes priming effects from a psycholinguistic view by using a decomposition level of letters, words, and semantic-level decomposition. The other way around the theory of compositionally state the meaning of expressions is composed by the meaning of its parts and the syntax composing them [208]. But Kracht [209] argues as well, that logical languages are typically not compassable. This means we can not take an arbitrary logical language and build our decomposition upon it.

The decomposition of meaning is used here as a basis and is not subject to the research itself. Andrews [11] analyzes the formal representation of NSM primes and their properties. He started the analysis of combining NSM semantic with formal semantics. Even though this is only a start for a subset of primes, it shows that the logico-semantic features of NSM might be formalized.

These approaches show that theories from linguistics are used to describe formal semantics in a decompositional manner, but these approaches all lack a construction mechanism which allows the agent to extend its known concepts. The decomposition we propose creates an automated mechanism which collects explanations for concepts and integrates them into the beliefs of the agent, i.e., the semantic graph. Depending on the context of the explanations, different facts

[1]Cambridge Dictionary Online, visit `http://dictionary.cambridge.org/` last visited on 09.09.2017

about the concepts and its relations to others is described.

Now for us humans, the decompositions to explain new concepts can be based on a multitude of information sources. We can learn through conversations, by processing arguments of a discussion, by reading a text, by thinking about the information we already know or by experience. For AI this is different since the cognition is restricted to the implementation of the agent which encapsulates the AI. The here presented decomposition represents a mechanism which is motivated by the way adult humans learn new concepts and use dictionaries and at the same time builds upon the theory of NSM which allows us to give the agent a set of basic concepts, creating the foundation to bootstrap the creation of an artificial meaning representation for more complex concepts.

The definition of a dictionary is most likely not of this form since they take into account the common sense knowledge of humans and therefore do not try to use as many primes as possible. This is anticipated in the decomposition, and that is why the decomposition replaces concepts close to semantic primes with semantic primes.

The next sections are structured in the following manner: Now that our objective is clear we take a deeper look at NSM in Section 6.1. Semantic primes have already been used in artificial languages which is shown in our analysis of three example semantic description languages for their containment of semantic primes in Section 6.2. We then take a look at the used data sources in Section 6.3. In Section 6.4 we look at the formalization of the output of our semantic decomposition algorithm. This formalization finally allows us to describe an algorithm which will create such a decomposition automatically in Section 6.5. We conclude our automatic semantic decomposition in Section 6.6 and describe its benefits and shortcomings.

6.1. Natural Semantic Metalanguage

Above we have described an explanation as breaking a complex concept down into simpler concepts. To determine the complexity hierarchy of concepts, we use a theory from linguistics called **Natural Semantic Metalanguage**. To recapitulate: we have looked at NSM in Section 3.10. In their works, Wierzbicka [411] and Goddard [143] for example have analyzed natural languages for a common semantic crux, and they found approximately 65 **semantic primes**. Making up a hierarchy of concepts only described with primes, then concept described with those "first-level" concepts and primes, making up the second level and so forth [p. 7][410]. Our decomposition will thus stop at those words since they do not need a further decomposition. This leads to the decision that synonyms of semantic primes are replaced with the prim.

An example of the first level might be the description of surprise [139]:

```
[X] was surprised:
    [X] felt something at that time
    like people can feel when they think like this:
        ``something happened now
         I did not think this would happen."
```

In this example, the complex concept of "surprise" has been described by only using semantic primes. Accordingly, in future descriptions "surprise" can be used to describe further concepts. Of course, this example description does not fit our need for AI. We do not restrict our explanation to syntactically well-formatted sentences and add additional information needed by an AI which is naturally available in humans, like the grammatical forms of “to feel” and synonyms and references for concepts like “time”. This is why we abstract from syntax in our decomposition. Even with this simplification, a decomposition of complex concepts could still become large. To decompose every concept into solely semantic primes seems like a rather theoretical goal like creating an Explanatory combinatorial dictionary (ECD), which is not needed since the level of decomposition can be adapted to the reasoning task the agent has to do.

Utilizing this theory for AI, we have multiple benefits besides the hierarchy of concepts: First, to bootstrap our natural language understanding we only need to describe the meaning of 65 concepts. Second, the primes fall into categories, which lets us describe syntactical rules for them in a more abstract way. This could be done, e.g., for logical primes, which might be handled in other manner or by other components than the other primes because they could be more universal. Third, those primes are natural language, which makes them understandable by humans, allowing understanding and analysis of the decomposition it self. This is a benefit over representations which are regarded black box approaches like artificial neural networks. Fourth, we have an example decomposition done by humans as in the example of surprised above. This allows an automatic decomposition since humans can help the automatic decomposition if it gets stuck. Fifth, the NSM theory gives the decomposition a theoretic grounding, separating this kind of ontology creation from other ontology learning methods. The theoretic grounding ensures that the selected primes are not arbitrary or can not be understood by humans. Since the primes have been shown to occur in many natural languages [144].

A Drawback of our decomposition approach is that it highly depends on the quality of information sources it can use. Furthermore, since we are using multiple data sources, multiple definitions of the same concept, which could lead to ambiguity. By using information sources which are created by humans, the decomposition contains faulty information, contained in those information sources and hence can be manipulated.

NSM is defined for natural languages. It is not clear if this property of semantic primes is also present in artificial languages, and if primes exist in artificial languages if they are the same as in natural languages. Nor is it clear if we can transfer the insight we have gained form natural languages through the NSM theory, to artificial languages. To see if the primes are as well present in artificial languages, e.g., as keywords in programming languages, we analyze three artificial example languages used for semantic modeling if they contain semantic primes. With that, we want to show, that the theory of NSM can be applied to artificial languages used in AI by showing that artificial languages already use semantic primes as keywords. Also, we want to answer the questions which semantic primes are already in use in artificial languages. By finding semantic primes in artificial languages, we ensure that those primes are not solely representative for a human fundamental semantic core and with that make it more likely that the rest of our work will not end into domain specific interpretation of some arbitrary concepts. Furthermore, with the NSM theory we would only need to provide the semantic for the ca. 65 primes, and with the right composition, we could create the meaning of all other concepts automatically. With the analysis of which primes already exists in artificial languages, we have a list of primes for which we have an explicit semantic defined already. Leaving us with the remaining NSM primes for which we still have to define a semantic.

Table 6.1.: List of semantic primes and equivalent concepts found in OWL.

Category	Semantic prime	OWL
Substantive	I	self.entry
	SOMETHING/ THING	owl:thing
Relational	KIND	owl:SubClassOf
substantives	PART	owl:topObjectProperty
Determiners	THIS	owl:entityURI
	THE SAME	owl:equivalentClass

6.2. NSM Semantic Primes in Artificial Languages

In this section, we look at how NSM semantic primes (see Section 3.10) are used in artificial languages. Artificial languages have semantic primes which need to be defined with a clear semantic for a compiler or interpreter to be able to use these languages. In computer science, such primes are called keywords. Since the NSM theory states that primes make up natural languages, we are going to analyze which role primes play in artificial languages. The retrieval of semantic primes as a subset of the natural semantic primes form NSM theory, in artificial languages shows that those primes are not reserved to natural language and that the implications drawn from the NSM theory could apply to artificial languages as well. We do so by looking at OWL as our ontology description language, PDDL as the most common planning description language and MOF as one of the most used modeling languages. This part of this work has been published in [97].

We are doing this because we want to show that the implications from NSM theory can be applied to artificial languages. The existence of semantic primes shows us that primes can be used in artificial languages as well and that those primes are not only used in natural languages. Which forms the theoretic basis for our representation of meaning. We use this basis, e.g., in the decomposition to argue when to stop decomposing,

Our analysis is done in the following way: We select a semantic prime and search for an equivalent in the given language. All found primes are contained, all not found primes are not contained, and the other concepts of the language are not primes, they are merely shortcuts or more complex concepts.

This experiment will allow us to declare the semantic primes from natural language also relevant for artificial languages and acts as a starting point for further discussions on which semantic primes are needed in artificial languages and which ones can be ignored. We start out with the analysis of the Web Ontology Language.

Semantic Primes in the Web Ontology Language (OWL) OWL is a semantic markup language to create structured knowledge representations and enrich them with semantics. OWL is a W3C standard since 2004 and has been continuously developed since [148]. OWL is an extension of the Resource Description Framework [220] and has become one of the most used languages to describe knowledge for AI. Since OWL is meant to describe structured knowledge the concepts used are abstract. Table 6.1 list all equivalents found in comparison with NSM primes.

Table 6.2.: List of semantic primes and equivalent concepts found in PDDL.

Category	Semantic prime	PDDL
Substantive	SOMETHING/ THING	:define
Determiners	THIS	::=
Existence	THERE IS	:exists
Time	BEFORE	:precondition
	AFTER	:effect
	A LONG TIME	:maintain
	A SHORT TIME	:wait-for
Logical concepts	NOT	:not
	CAN	:action
	BECAUSE	:imply
	IF	:when/constrained

Semantic Primes in the Planning Domain Definition Language (PDDL) PDDL is a first-order logic based language defined as an extended BNF [122]. Commonly, it is used to provide a standardized way to describe planning problems and the associated domains. The syntax allows to define among others actions, effects, quantifications and constraints and was intended to enable developers to describe the "physic" of a domain. Given such a description the reasoner uses a goal defined in PDDL to search for a plan that satisfies all constraints, requirements, and preconditions. The concepts which are equivalent to semantic primes are listed in Table 6.2.

Semantic Primes in the Meta Object Facility (MOF) MOF has been introduced by the Object Management Group and is formally defined, e.g., by Smith et al. [363]. MOF has been developed to model the structure and behavior of entities in software development. For example, UML[2] implements MOF. MOF has the special properties that it can describe its meta model, which is exactly what semantic primes are supposed to be for natural language. MOF has mostly structural semantic primes. Table 6.3 lists all equivalents.

In conclusion, NSM provides a restricted set of primitives, which in the theory of NSM is sufficient to describe all concepts of natural language. We have shown here that three of the artificial modeling languages do also contain some of the semantic primes. Also Andrews [11] has shown that some of them are reconcilable with formal semantics. Depending on the use case of the modeling language other primes are used.

Since those languages are specially designed to be understood by machines, the primes used have a specified semantic and can be understood by reasoners. Since those languages are designed for a special purpose, they contain special purpose concepts, which do not map to semantic primes but rather are shortcuts to more complex concepts.

Especially for the creation of an artificial representation of meaning, this seems like a good choice, because with the interpretation of those primes the semantic becomes more expressive. If semantic primes are expressive enough to describe all concepts, then we need a way of finding the right primes in the right order to describe a given concept. The next section discusses where we can get information as a starting point to construct our decomposition into semantic primes.

[2]see: http://www.omg.org/spec/UML/

Table 6.3.: List of semantic primes with and equivalent concepts found in MOF.

Category	Semantic prime	MOF
Substantive	YOU	uri
	SOMETHING/ THING	object
	BODY	instance
Relational	KIND	type, extent
substantives	PART	property
Determiners	THE SAME	element.equals
Quantifiers	ONE	multiplicityElement
Location	BE (SOMEWHERE)	link
Existence	THERE IS	element
	HAVE	classifier
	BE (SOMEONE/SOMETHING)	extend
Life and death	LIVE	create
	DIE	delete
Logical concepts	CAN	operation
	IF	event

6.3. Data Sources used in the Semantic Decomposition

For the decomposition we have to choose the data sources we can use to look up information about a concept. Here the first idea is to use structured knowledge, which has already been formalized, sparing us from the effort of formalizing it. This could, e.g., be done by using ontologies which model information we need. The second idea is to use information in text form which is semi-structured. Such an information source could be a dictionary, where information about words is given in the form of definitions, words with the same meaning and how they are used in example phrases. The third idea then is to structure unstructured information like videos, speech or figures. Next, we will discuss our selection of data sources and how they are used.

For the structured information, there are many ways of formally describing facts about concepts and the relation between them (see Section 6.4). This ranges from data base schemas, over annotations to descriptions logics which use formal representations like logic. The decision on which information sources to include is based on information available and the amount of effort needed to integrate it into the decomposition. Figure 6.1 describes an estimate of effort to structure and specificness of the information of data sources. Here, the colors describe the effort needed for integration into the decomposition: green for less effort and red for more effort.

In Figure 6.1 we draw the line, of which information sources we integrate, between structured text and text for ontology learning. We selected to integrate the first three types of information (above the line) ontological, taxonomical and structured text because the fourth one ((unstructured) text) is still subject to research [245, 274, 28] which is not in the focus of this dissertational thesis. Since the mining of text is less complex than the analysis of audio, where most approaches translate the audio into a text to do semantic analysis, we will not look at audio information. The same arguments can be held for visual information. The colored areas symbolize the range a given information type can be argued to describe. Here, e.g., taxonomical information consists of only one relation, which makes its description more specific then ontological information.

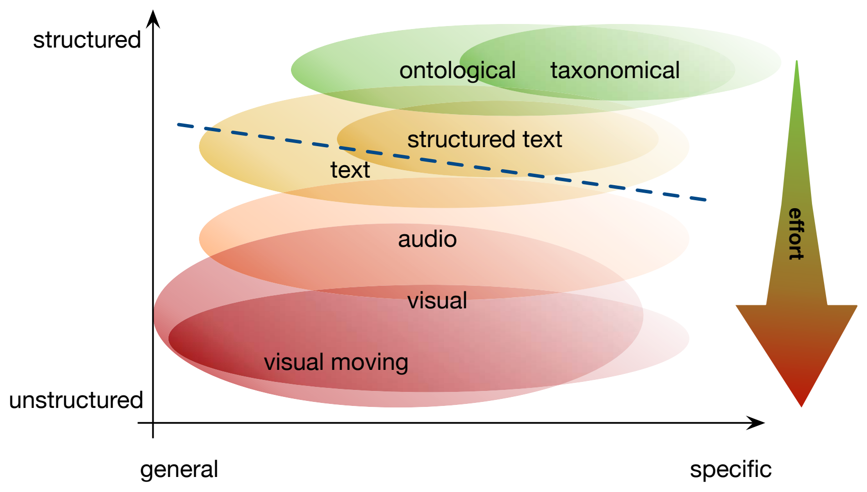


Figure 6.1.: Classifiction of data sources in information type and formality.

To select data sources, we surveyed structured data sources displayed in Table 6.4. Here we selected knowledge graphs which have been automatically generated or handcrafted which is denoted by the column “automatically extracted”. This differentiation is done because it gives us information about the quality of the created graph and how much effort an update takes. The second parameter (Manually modified) distinguishes if the graph can be modified without modifying the underlying data source. This gives us the choice of correcting errors manually. The third column (collaborative) indicates if the knowledge is modified by one organization or by collaboration. This gives us information about which quality the information has, e.g., if it has been created by one organization or author, e.g., Roget’s Thesaurus or multiple once like Wikipedia. The amount of information collected in the information source is shown in column four (number of entities/relations) which is an estimate of how many entities and relations are collected in the information sources. The last column (availability) describes if the information source is publicly accessible. This is needed for the information source to be part of our research.

These information sources can be separated into three types of information structures: The first nine data sources are graph representations like DBPedia [227] and Wikidata [392]. The second type is dictionaries like WordNet [270] and Wiktionary [263] and the last one is a Thesaurus [257]. Here DBPedia [227], Wikidata [392] and Wiktionary [263] have Wikipedia [3] as source. Each of them extracts different aspects of this crowd created information source. While Wiktionary creates a dictionary with words and their definitions, word type, and example sentences, Wikidata extracts an ontology with entities and their relations. Wikidata is mostly manually created and serves as an upper ontology for DBPedia to extract information out of Wikipedia articles. Where in BabelNet [283] the number of entities and relations is only given for all 271 languages. In addition, for the dictionary LDOCE [312] we could not find a relation count.

For a proof of concept we selected at least one of each category to be integrated into our decomposition: As the only thesaurus we integrated Roget’s thesaurus. Since WordNet is a standard for defining word senses in NLP tasks [283] we have integrated WordNet as exam-

[3]https://www.wikipedia.org/

Table 6.4.: List of knowledge graphs with the amount of entities and relations in thousands (T) and millions (M).

Name	automatically extractracted	manually modified	colla-borative	# entities # relations	availability
Knowledge Vault [75]	✓	×	×	$45M\backslash 241M$	×
YAGO [371]	✓	×	×	$0.9M\backslash 5M$	✓
YAGO2 [170]	✓	×	×	$9.8M\backslash 80M$	✓
YAGO3 [248]	✓	×	×	$10M\backslash 120M$	✓
Freebase [33]	×	✓	✓	$40M\backslash 637M$	✓
BabelNet [283]	✓	×	✓	$6M\backslash 380M$	×
Wikidata [392]	×	✓	✓	$20M\backslash 100M$	✓
ConceptNet [238]	×	✓	✓	$1.6M\backslash 300T$	✓
DBPedia [227]	✓	×	×	$3.7M\backslash 400M$	✓
WordNet [270]	×	✓	✓	$118T\backslash 166T$	✓
LDOCE [312]	×	✓	×	$230T\backslash ?$	×
Wiktionary [263]	×	✓	✓	$1.6M\backslash 380T$	✓
Rogets Thesaurus [257]	×	✓	×	$59T\backslash 62T$	✓

ple ontology. Because WordNet has missing vocabulary, we added Wikidata [392] to our data sources. As Dictionary we integrated Wiktionary because of it was the only one freely available offline.

With this information sources for the decomposition, we are able to look up concepts, and integrate them with other concepts in a semantic graph automatically. This enables the agent to integrate additional information about a concept into its current semantic graph by getting explanations from the different information sources.

Extracting concept form those information sources and integrating them into the beliefs of an agent to form an artificial representation of meaning, we have to define how the connectionist part of this representation formally looks like. The formalism we will use is described in the next section.

6.4. Formalization of the Semantic Graph

This section describes the formalization of the graph which is created by the decomposition. We first discuss our requirements for the representation, grounded in our use case. We do so by looking at our example for relations which is the relation “to give advice” shown in Figure 6.2. Here the relation “give” inherits from an abstract “transfer” relation. It is a tertiary relation since it has three arguments: The adviser the advisee and the object given. Then we look at different formalization types and select one of them. We have the requirements regarding this graph which are an extension of the requirements described in Section 4.2 where only one requirement is not fulfilled by OWL 2: the requirement that Relations are concepts (see Language Requirement 11 on Page 59). Therefore we define a formalism fulfilling this requirement in this section.

Relations and concepts are often presented as a graph like structure. We have defined our for-

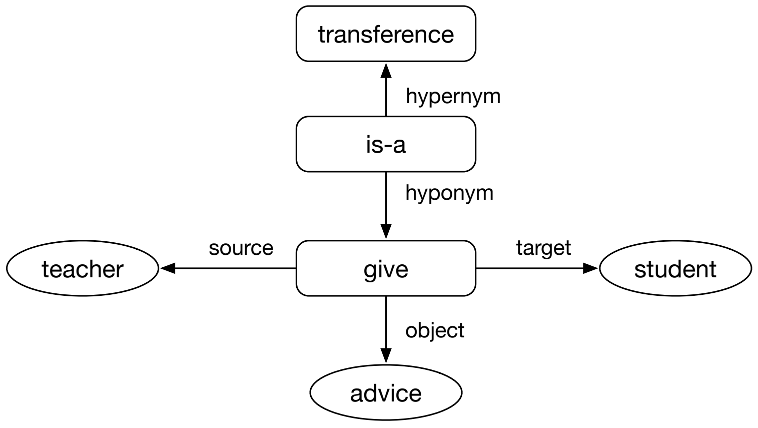


Figure 6.2.: Example for a relation which connects three concepts and is it self reference by a relation.

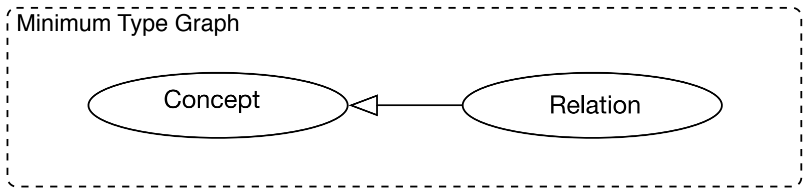


Figure 6.3.: Minimal type graph for the description of a decomposition graph.

malization of an ontology in Section 3.6, which is used here. Since a graph with nodes and edges where each edge connects two nodes is not expressive enough to satisfy our requirements (e.g., n-ary relations and relations between nodes and edges), we need something like an attributed graph [66]. Attributes allow us to model, e.g., concepts with literals and relations with names. Further, we need types of concepts and relations which lead us to typed attributed graphs. Further, the types we describe for concepts and relations describe a hierarchy which leads us to attributed typed graphs with a type hierarchy.

Therefore based on the requirements described in Section 4.2 we formalize the graph structure as an Attributed Type Graph with Type Inheritance introduced by De Lara et al. [66]. For a formal definition see Section 3.4.

The requirement of having an n-ary relation (multi-graphs or hyper-graph) and the requirement of having edges among edges leads to the need of describing more complex edges. The fact that relations are concepts as well is problematic for standard graph representations. Having edges which are nodes makes it necessary to extend a hyper-graph. The requirement of having a type hierarchy and types of concepts and relations leads us to specify a base type graph, which can be adapted to a specific use case. This basic type graph is shown in Figure 6.3.

To distinguish concepts from relations, we define a minimal type graph shown in Figure 6.3. This minimal type graph can be extended regarding its use. In this minimal type graph, the relations inherit from concepts and account for the requirement of “relations are concepts”.

We define a minimal type graph so that we can guarantee for our algorithms, that we have at least the type of concepts and relations.

Definition 31. ***Minimum Type graph:*** *The minimum type graph TG_{min} for describing a semantic decomposition is the attributed type graph with inheritance depicted in Figure 6.3.*

The minimum type graph like we have defined it in Definition 31 formalizes the minimal properties of a graph which describes the types used in the semantic decomposition and later on during the Marker Passing. The minimum type graph states the following things:

Relations are concepts all relations are inherited from concepts. This means each relation can be seen as relation or as a concept.

For our requirement of a conjunction of relations being concepts, we have to look at how we define our Type graph *ATG* and its Inheritance graph *I*. We can see in Figure 6.3 that the inheritance in the type graph can be used to define that "Relations" are special kinds of "concepts" where concepts are the nodes and relations are the edges for our application. For an extension of the type graph we require that the extension is conforming the minimal type graph:

Definition 32. ***Type conformance:*** *A type graph TG conforms to a minimum type graph TG_{min} if the following conditions hold:*

1. *$\exists$ Morphism $\phi : TG_{min} \to TG$, meaning that TG_{min} is contained in TG*
2. *$\forall$ nodes N are in the clique of node type "Concept", meaning that everything inherits from concepts.*
3. *$\forall$ edges $e \in TG$: $source(e)$ are from the clique of node type "Relation", meaning references only refer from a relations to the related concepts.*

Here the first conditions ensure that all specialization of our type graph at least contain concepts and relations with relations being a special concept. This guarantees that we can work with those node types in our algorithms later on. The second conditions ensure that everything is a concept. Allowing us to see each relationship as a concept as well. The third condition describes the semantic of the arrows used in the type graph: The arrows always have their source in a relation node and point to concepts which are related to the relations.

The types of nodes and edges can be imagined as a mapping to a common concept in the type graph. In our decomposition, an "equals", "is-A" or "synonym" relation could all be mapped to the same type of an equivalence relation. There is the possibility of introducing cardinality with constraint languages into such a type graph. We neglect this possibility because it is not used in our use case.

Furthermore, the requirement of having n-ary edges can also be handled by the type graph. This is done by the use of outgoing edges from relations to concepts. Each reference has a name and therefore can describe something like our "gives" relation in our example shown in Figure 6.2. Since our type graph as multiple references the give relation can have the relations advisor, advisee, and object as shown in Figure 6.4.

To conclude we use the attributes of the type graph to label our nodes and relations. We use the type hierarchy to define our node and edge interpretation. For our example shown in Figure 6.2 we can define the following basic type graph shown in Figure 6.3. The minimal type graph $I = (I_V, I_E, s, t)$ of our *ATGI* is shown in Figure 6.3. It serves as a basic type graph which can be extended by the different applications like shown in Figure 6.6. This means all edges have a label which again can be decomposed. To illustrate, in our example `give(teacher, advice, student)` we need the label "give" to describe the relation. In consequence, each edge is a concept, which can be seen as a node having relations with other nodes or edges.

Here relations are concepts and with that nodes and edges are no longer strictly separated. This means each edge can be interpreted as a node. In our semantic graph, the entities are concepts. This means that relations are concepts, too, which can be imagined as having a sub graph (its decomposition). In Figure 6.3 we have defined that at least concepts and relations need to be defined by the type graph.

For the understanding of how we can do so, we now define an example graph with an example type graph to formally describe our example shown in Figure 6.2.

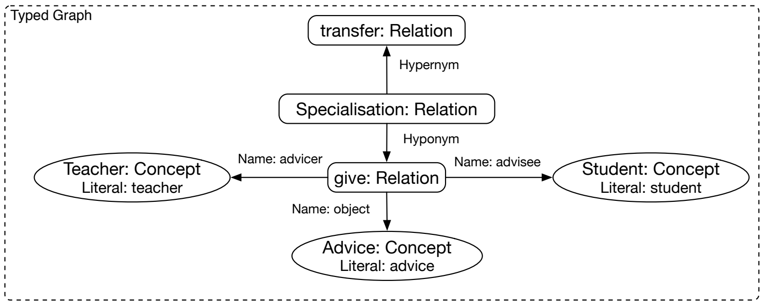


Figure 6.4.: Example of a typed graph.

This leads us to the definition of our typed graph from our example in Figure 6.2 to the typed graph depicted in Figure 6.4.

Now we can define a type graph and use it to type our nodes and relations of our representation of meaning in our semantic graph. With the typing of the graph, we want to make sure that each named relation is represented by a node.

Now we can look at the example application of our formal graph and the description of the semantic graph with our formalism. With that, we are going to bridge the rhetoric gap between the formal representation of a graph and our application in the semantic decomposition. We do this, by explaining how we use nodes and edges and how the above-postulated requirements are used. Further, we extend the basic type graph to fit our application.

This example is shown in Figure 6.5 is taken from a test data set of Rubenstein and Goodenough [336] for semantic similarity of words. The example shows two words "Noon" and

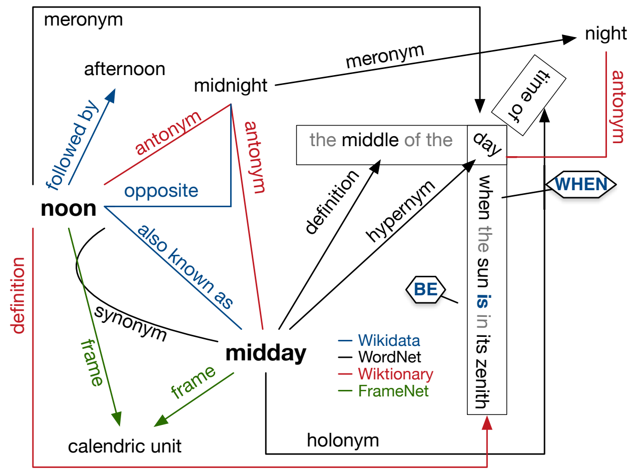


Figure 6.5.: Abstract example of an decomposition of midday and noon.

“midday” and their relations to other words and each other. The colors of the edges describe from which data source the relations has been taken (Wiktidata in blue, WordNet in black, Wiktionary in reading and FrameNet in green) and the sentences in boxes are definitions of the words. This is a more complex decomposition regarding two concepts as shown in Figure 6.5. We can see that some semantically close relations are named differently in different dictionaries. Here, e.g., “midnight” is in “opposite” relation with noon in the Wikidata and in the Wiktionary a similar relation is called “antonym”.

Further, we can specialize the type Graph *TG* with the graph from Figure 6.6 to fit our application as an example type graph used in this word. The type graph shown in Figure 6.6 is an extension of the type graph shown in Figure 6.3. The extension is the specification of different relation types which are needed to describe the example decomposition shown in Figure 6.5. With this extension, we can distinguish between the different node types like concepts and semantic primes and between the different relations types, e.g., the named and semantic relations.

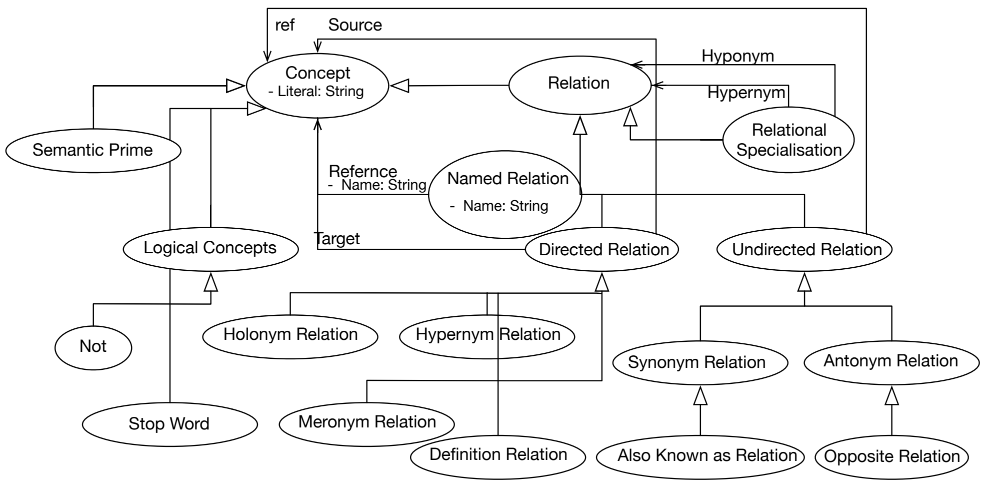


Figure 6.6.: Example of a type graph (big).

In Figure 6.6 we describe all the relations from our example shown in Figure 6.7. This example is based on our decomposition example shown in Figure 6.5. Here undirected edges are depicted by neglecting the target and source reference. Further to safe space, the definitions are depicted as a set of concepts through the dotted lines. This is done because we could not order the concepts due to the multiple occurrences of the same concept. For the sake of simplicity, we neglect this here in our example.

We model these types so that we can react to them with the Marker Passing. This means that we can interpret the same markers differently in every type of node or relation. To model a graph like in Figure 6.2 and 6.5 we create a example type graph like depicted in Figure 6.6 which we will use for our formalization in the doctoral thesis. Here, e.g., we modeled relations as concepts and semantic relations as a relation, and antonymy as one of those semantic relations which has the subtype of opposite relation. This is done because we see an opposite relation as binary (complementary antonyms) and with that as a special case of an antonym. With this differentiation, we can react differently to the special type of opposites during the reasoning upon the graph. This leaves us with a formalization of our semantic graph. Now the question remains on how to create such a knowledge graph automatically. We call this creation algorithm **Semantic Decomposition**. Now we will describe the basis for the decomposition algorithm

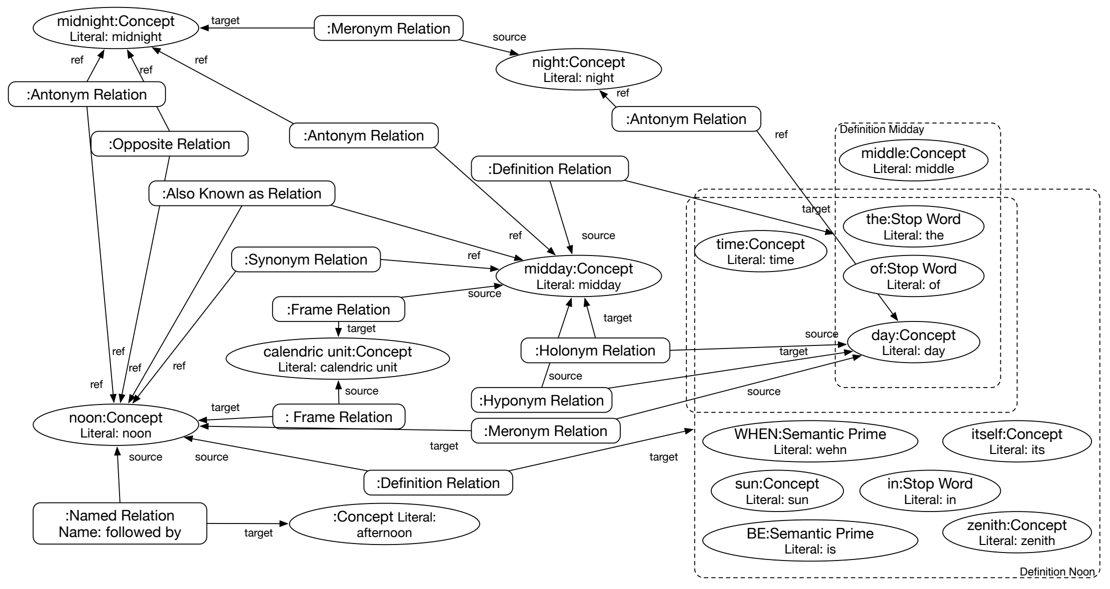


Figure 6.7.: Example of a typed graph of noon and midday.

which produces such a graph.

6.5. Decomposition into Semantic Primes

In this section, we will go into details on how we can decompose meaning of complex concepts into semantic primes. The goal of this chapter is the creation of what Wierzbicka [410] calls 'Mental Lexicon' for artificial intelligence. For us that means transforming the information given by the information sources depicted in Figure 5.1 into a graph as described in Section 6.4.

We will define an algorithm for the semantic decomposition in Section 6.5.1. We will go into detail on how the semantic relations contribute to the decomposition in Section 6.5.2 to Section 6.5.6. Then we look at the use of syntax in the decomposition in Section 6.5.7 and argue why it is not being used. We then look at the conditions for the termination of the algorithm in Section 6.5.8. We conclude our discussion in Section 6.6

6.5.1. Algorithm for the Decomposition of Meaning

This section will propose an automatic decomposition of meaning starting from a more complex concept and ending up with a description build upon semantic primes. This decomposition is done similarly like monolingual dictionary[4], meta languages like MOF and OWL which both describe their typing (the meta model of the language) and how Wierzbicka [410] describes a mental lexicon. Further Melcuk calls this kind of dictionary an Explanatory combinatorial dictionary [261].

It is important to notice that this is not meant for natural language processing. The languages decomposed here are formal languages like OWL. Hence our mental lexicon only contains concepts of this language and therefore has the goal to be computer readable. The task of building a

[4]Even though the dictionary does not forcibly follow the decompositional direction (e.g., the definition of simplicity of words) proposed in the NSM theory, but it presents us with the possibility to connect one word with a set of words called its definition.

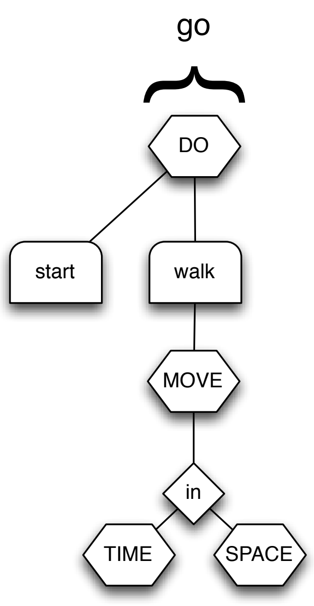


Figure 6.8.: Example decomposition of one meaning of the concept “to go”.

concept even if it is composed of multiple words the search for lexical paradigms and inflections is left to linguists and will not be subject to our work.

However, there are primes which are human readable, which is no surprise, since the artificial languages have been created by humans. The prime KIND, for example, is part of OWL and MOF as we have argued in a previous publication [93]. In both languages, such a prime has the same interpretation of tagging the inheritance of an object. Inheritance means that all properties, attributes, and methods of the parent object are available to the child objects. This means that if we tell a reasoner that object B is a sub class of object A, the reasoner knows what to do since the abstract prime KIND is part of its language.

For concepts which are not part of the language, we can describe all needed facts in the language itself. This is done in domain specific ontologies like, e.g., in models of MOF or ontologies in OWL. The argument here is that those languages already include some of the primes and it seems like the NSM theory is applicable, at least for this subset of primes. This means that a decomposition of concepts of an artificial language might be possible to.

With decompositions, we want to decompose every concept into those entities, which are known to the reasoner in a way that the meaning of the concept can be distilled from the meaning of the primes. One example of a decomposition could be the explanation of “to go”. We take the English dictionary [394] as a reference for this decomposition: the meaning of “to go” is defined as to move in time or space. Figure 6.8 illustrated this decomposition as a schematic depiction. Here the hexagons are semantic primes from the list of NSM primes, round cornered tetragons are intermediate concepts, and the rhombus is auxiliary words. The decomposition is done by taking the definition out of a dictionary, where such a complex concept is defined using less complex concepts. The intermediate concepts can provide different interpretations for different contexts.

The Algorithm 1 is structured after what we understand as explanations. If we look at humans and how they learn new concepts, explanations are a common tool to do so. Of course, not everything can be explained, and some concepts need experience or cognition to capture their meaning, like the notion of pain or the concept of red. Nevertheless, explanations are how most adult humans learn[5]. The second and perhaps more important part of this human studying process is the capability to self-explain. This is needed to integrate the newly learned information into one's beliefs. As a result, adjusting the world view to incorporate the new facts, update the reasoning about this new concept and their relations, and with that change the way one thinks about the world. For this reason, we look at the decomposition next, and then we glance at self-explanation.

Algorithm 1 A decompositional algorithm.

Name: Decompose **Input:** Concept *word*, Integer *depth* **Output:** Semantic Graph

```
 1: SemanticGraph decomposition = ∅
 2: function DECOMPOSE(Concept c, Integer depth, SemanticGraph decomposition)
 3:     if $depth \geq 0 \land c \notin$ decomposition then
 4:         concept ← Normalization(c)
 5:         Relations ← getRelations(c)
 6:         Definitions ← lookUpDefinitions(c)
 7:         AddConcept(c, decomposition)
 8:         if concept ∈ IsSynonymOfPrime then
 9:             return decomposition
10:         end if
11:         for all r ∈ Relations do
12:             AddConcept(r, decomposition)
13:             AddRelation(r, getTargets(r), decomposition)
14:             DECOMPOSE(r, $depth - 1$)
15:             for all target ∈ getTargets(r) do
16:                 DECOMPOSE(target, $depth - 1$)
17:             end for
18:         end for
19:         for all definition in Definitions do
20:             for all def in definition do
21:                 AddConcept(def, decomposition)
22:                 AddRelation(”definition”, {Concept , def}, decomposition)
23:                 DECOMPOSE(def, $depth - 1$)
24:             end for
25:         end for
26:         return decomposition
27:     else
28:         return decomposition
29:     end if
30: end function
31: DECOMPOSE(word, depth, decomposition)
32: return decomposition
```

The functions *AddRelation* and *AddConcept* are convenience methods for adding the relation and concepts into the semantic graph. The functions *AddConcept(concept, decomposition)* and *AddRelation(relation, getTargets(relation), decomposition)* add the concepts or relations to the graph which represents our decomposition. AddConcept adds the given concept to the graph

[5]To back up this claim, see how students in Universities learn by studying complex concepts and their relations. Where the most information is given to them by course consisting of lectures, which can be seen as explanations.

nodes and AddRelation adds the relation between the concept its targets to the relations of the graph.

We now have a look at how such decomposition can be created and how automatisms might help. We identified the steps for a decomposition as described in the recursive Algorithm 1. The algorithm takes as input the concept that is subject to the decomposition. As a successful decomposition will always build a graph, the semantic primes are the termination criterion for the recursion.

The Algorithm 1 reads as follows: Line 1 initializes the semantic graph which we will build up during this algorithm and which represents the result at the end.

Line 2 to 29 represents the recursive function which is called on all decomposed concepts. This function adds the decomposition to the semantic graph initialized in Line 1. Which is called until the decomposition depth is reached or all concepts have been decomposed into semantic primes. We will build a hierarchical structure made up of concepts also referred to as lexical units. Those concepts include a lexical representation, the textual representation of a lexeme and a decomposition.

Line 3 checks if the concepts have been already decomposed or if the decomposition depth is reached. The decomposition depth is a parameter of the decomposition, which restricts the decomposition to an amount of relations to which the decomposition extends. The second part stops the decomposition of decomposing the same concepts over and over again. Additionally, the decomposition sops here, if a synonym of the concept has been decomposed previously. This is because if a synonym has been decomposed previously, its synonyms are added to the decomposition as well. Thus this synonym, which is supposed to get decomposed now, is already part of the decomposition and is not decomposed again.

Line 4 takes the concepts to decompose and normalizes it. Here the inflection is removed revealing the stem of a concept. Furthermore a concept includes all its inflections (all concepts which can be created by applying grammatical forms to a concept like eating, ate, eaten), all lexical paradigms for this concept (all concepts rooting from the same word stem like to dine, dinner) and all sub-categorization frames (like the valence which is the amount of parameters like ask, ask X, ask X for Y). We remove this kind of inflection because we are interested in the concepts described by a word, not its relation to other words. We can integrate syntactic information into the graph, by adding syntax relations and nodes. For this reduction, we use the linguistic process of **Lemmatization**[6]. The function Normalization in Algorithm 1 Line 4 hides this normalization of a concept.

Line 5 gets all the relations of the concept from the used dictionaries. This means we are looking through all our dictionaries and look up all the semantic relations we can find and remember them for later processing.

Line 6 looks up the definitions of the concept in all available dictionaries.

Lines 8 to 10 check whether the concept itself is a prime. If this is the case, the prime is added to the decomposition, and the decomposition is finished for this concept finally the decomposition is returned. This hides technical optimizations like that we check for synonyms of primes as well to make the search a bit broader. At the same time, we simplified the stop word removal here. Stop word represents words which are ignored. Those are taken from natural language processing theory [412]. These are mostly words with less semantic meaning

[6]Sometimes Lemmatization is referred to a Stemming, where, e.g., the end of words is removed to remove something like a plural s.

like, e.g., “a”, “an” or “the”. Those nodes are removed and are not further decomposed.

Lines 11 to 18 handle the relation of the concept we are decomposing. Here all relations are added to the decomposition as a relation between concepts. Then all concepts which are connected by those relations are recursively decomposed.

Lines 19 to 25 decompose the definitions. Each definition is a list of concepts which get decomposed again. The definition is connected to the definiendum via a “definition” relations.

The main point to notice here is that the order of words of the definition is lost. The grammar is, as a result, untouched and does not yet influence the decomposition. Further, this means the decomposition is highly influenced by the quality of the dictionary.

Algorithm 2 Get the relations of a concept

Name: GetRelations **Input:** Concept *c* **Output:** List<Relations>

```
1: List<Relations> relations = ∅
2: for all dict in Dictionaries do
3:     relations.addAll(GETSYNONYMS(dict, concept, GetWordType(concept)))
4:     relations.addAll(GETANTONYMS(dict, concept, GetWordType(concept)))
5:     relations.addAll(GETHYPERNYMS(dict, concept, GetWordType(concept)))
6:     relations.addAll(GETHYPONYMS(dict, concept, GetWordType(concept)))
7:     relations.addAll(GETMERONYMS(dict, concept, GetWordType(concept)))
8: end for
9: return relations
```

Algorithm 2 describes the GetRelation function which looks up the known relations from the information sources. Each information sources is encapsulated behind the interface of a dictionary. The dictionary implements the signature of concept and definition from Section 3.5. Thus, this dictionary implements the functions GETSYNONYMS, GETANTONYMS,... as well as the GETDEFINITION function. Algorithm 2 represents an abstraction of how the information about a concept is collected from the information sources.

Algorithm 3 Get the definitions of a concept

Name: lookUpDefinitions **Input:** Concept *c* **Output:** List<Definitions>

```
1: List<Definitions> definitions = ∅
2: for all dict in Dictionaries do
3:     definitions.addAll(GETDEFINITION(dict, concept, GetWordType(concept)))
4: end for
5: return definitions
```

Algorithm 3 describes how the definitions are collected from the dictionaries. Each dictionary implements the signature describe in Section 6.3. Here the GetWordType() function determines the **Part of speech (POS)** of a concepts, so that the dictionary can determine which word sense to return. This parameter can be unspecified in which case all word senses are considered.

Now we will have a look at how the different semantic relations are integrated into the decomposition to argue why the above algorithm has been implemented the way it is. In the following discussions, we will look and additional information or ways the information could have been included into the decomposition. These discussions are held to explain how the decomposition in Algorithm 1 could be extended. Additionally, we will discuss simplification we have made in Algorithm 1 to keep clean from distractions and ease the understanding.

6.5.2. Decomposing using Synonyms

This section discusses our use of synonyms in our algorithm. We focus on things we have not explicitly integrated into the algorithm. This is done to show potential improvement points or our Decomposition.

Since synonyms are equivalent in meaning regarding the context of use, the synonyms can be replaced in an utterance, without changing the meaning of the utterance. Consequently synonyms play an important role in the decomposition. The decomposition could be stoped, if the meaning of synonyms is known. Imagine if one concept is not known in an utterance, but by providing a known synonym, the utterance can be understood completely. Synonyms are collected with the function $\textit{getSynonyms} : \textit{Concept}, \textit{WordType} \to \textit{Concept}*$ in Algorithm 2.

With that synonyms make the search for primes broader. Depending on the type of semantic similarity measure and the threshold, from which we call two concepts synonyms the breadth of the decomposition can be controlled. This again is defined by the information source which decided which two concepts are seen to be a synonym. Line eight of Algorithm 1 hides this implementation detail. Here we check if a concept is a prime, but at the same time, we check whether one of its synonyms is prime or if a synonym of primes contains the concept.

On improvement point of the Algorithm 1 could be the selection of the word sense before choosing the synonyms narrows the breadth but focuses on the right synonyms for the decomposition. In detail, there are two parts of the decomposition where synonyms can be used: Regarding the concept subject of decomposition and regarding the semantic primes. In the first case we can use the context of uses of the concept to select the word sense, and with that, the right synonyms, check if one of those synonyms is a prime and if so, we can stop decomposing since we could replace this concept with a prime without changing any meaning. Another improvement point is the caching of concepts we already have decomposed, which is done in our implementation but neglected in the Algorithm 1 for simplicity.

For further improvement, we could identify all synonyms of the primes, classified in their context and word sense. Then when decomposing, we can check if, for the given context and word sense, there is a prime or a synonym of a prime which leads us to again stop the decomposition. Analyzing, the synonyms of primes allows us to specify in which word sense the prime is used. Since there is a relatively small set of primes (ca. 65), we can do this for all of them in an insignificant amount of time. At the same time, defining synonyms for semantic primes allows us to downsize the effect of the neglecting of syntax during the decomposition because all those synonyms cause the decomposition to stop and do not need further analysis. In Table 6.5 this can be seen for example for the prime “I”. Here we can define syntactic variations as synonyms which spare us the effort to find ”yours truly” as one concept and its syntactic analysis.

Further Algorithm 1 neglects the fact that if a synonym is a replaceable concept in a given context, then the hyperonym and hyponyms can be treated as synonyms, e.g., almost all nouns could be replaced by SOMETHING/THING. If or when it makes sense to use specialization, and generalization can be looked at separately, which is done in Section 6.5.4 and Section 6.5.5.

Using a synonym is like explaining what a truck is by explaining something like: “A truck is like a bus but for transporting goods instead of people.” If the concept bus has already been decomposed, this can lead the reasoner to extract multiple facts for a truck. If the concept word type (POS) is known, then this gives some insight into its relations, because not all definitions have to be looked at, since the definitions with a different POS can be neglected. The implemen-

Prime	Context	Synonym
I	self reference	me, myself, yours truly
YOU	direct reference	thee, ye, ya
SOMEONE	general reference	individual, soul, person, anybody, anyone, mortal, somebody
SOMETHING	general reference	THING, sth, item, affair, matter
PEOPLE	general reference	populace, nation, mass, lede, race, commoners, community,tribe, folk, clan, folks
BODY	physics	dead body, torso, soma, organic structure, corpus, flesh, trunk, physical structure

Table 6.5.: Semantic primes in the category substantive in the English language [140].

tation of our decomposition includes the incorporation of a word POS which in Algorithm 2 but is neglected in Algorithm 1 for the sake of simplicity.

As primes are concepts as well, they have semantic relations like synonyms and antonyms like all other concepts. In this section, we look at the synonyms of primes and in which context they can be replaced. Since synonymy is a fuzzy concept as defined in Section 3.9. With this definition, it is possible to determine synonyms of primes. Those synonyms then are replaced by the lemma of the prime[7]. This means that the lemmatization is the removal of inflection caused by syntax. In simpler words: This means to remove the effect a grammar has on a word.

Semantic primes are seen as word senses, thus only have one meaning [142, p. 460]. If we look at the Table 3.2 we can see that there are two primes ”BE”. It is used in two senses: to be in a location and to be something. Those are separated in Table 3.2 by the specialization in parentheses (SOMEWHERE and SOMEONE/SOMETHING).

Some of the primes in Table 3.2 have already synonyms defined. If we look at MUCH and MANY, OTHER and ELSE and finally SOMETHING and THING there primes seem to have the same meaning. This leads to a decision which one of them should be used. Here we have selected arbitrary one prime and the other as a synonym.

In natural language, a prime can have multiple words senses in which it could be used. Consequently, different contexts in which it can be used and with that different synonyms. We now try to identify the synonyms of primes an order them by context. The following methodology is used to identify synonyms for the primes. All dictionaries of used in the decomposition are used to look up the synonyms known to the decomposition.

Here we analyze an example category: substantives. Here the primes are to reference others, one self or in general things. Here is a difference between SOMEONE, SOMETHING, and PEOPLE. The first semantic discrimination is between humans and the other things in the world. SOMEONE does address a person, as well as PEOPLE, addresses us as human kind. However, PEOPLE is more general then SOMEONE, since SOMEONE addresses one of the PEOPLE. The synonyms of primes have been selected by querying the data sources for synonyms and selecting the one which fits the word sense of the prime manually. Table 6.5 shows an example of the category of substantive primes.

[7]The lemma of a word is its base form. This means we are replacing “are”, “was”, “am”, “is” with its basic form “BE”

The rest of the tables can be found in Appendix A Section A.4. Next, we will look at the decomposition using antonyms explaining why antonyms are used in the decomposition and which theoretical role they play in the decomposition.

6.5.3. Decomposing using Antonyms

Using antonyms is similar to using synonyms during the decomposition. Here again, an antonym is not well defined, and there is a degree of antonymy to every concept pair. Choosing a threshold of which concepts are an antonym to each other is one parameter which defines the output of a decomposition.

Also, we will see that selecting how to use antonyms in a decomposition is not as simple as it is with synonyms. This is because there is a multitude of different antonyms.

From the discussion of antonymy in Section 3.9 we know that this only works for some primes. If we look at the change of evaluations a sentence to true or false statements, the replacement of an antonym again has to depend on the context the concept to be replaced is used in. If we take the example sentence: "Now it is sunny", and replace sunny with a negated antonym cloudy this changes the situations in which this sentence is evaluated to true or false. You can imagine that it can be not cloudy at night, but it is most of the time not sunny. This non-symmetrical antonyms special cases we will look at next. In those cases, the antonymy might not be a undirected edge, but rather has different roles. That antonyms are not symmetric can be seen on the example of to buy which is not equal to "not to sell". Where to buy and to sell are two parts of the same transaction. In consequence to buy and to sell might be antonyms in some lexical contexts, but they can not be handled by negating them.

Namely, the function $getAntonyms : Concept, WordType \rightarrow Concept*$ in Algorithm 2, gets the appropriate antonym relation for a given antonym. This is implemented by the signature of a dictionary described in Section 6.3.

As we have seen in Section 3.9 there are different types of antonyms. Not all antonyms can be used the same way. Therefore we can not like with synonyms (see Section 6.5.2) stop the decomposition when we find an antonym of a prime.

Next, we discuss the different types of antonyms and why they could not be used during the decomposition.

Gradable

Gradable antonyms are antonyms which can not be separated into two classes. A common example is a temperature: "hot" and "cold". Since we do not include fuzzy concepts into our decomposition, these antonyms are handled by a simple default antonym relationship.

Complementary

The complementary concepts are not in an antonym relation as in being the opposite, but rather to complete a set of states. For a machine to be "off" or "on" are examples of a complementary antonym. Another example is "dead" and "alive", where the one can not be without the other. In opposite to purely relational antonyms, complementary antonyms are not restricted to relationships. Because of the open world assumption, there is alway a third option of modeling not knowing if something is "off" or "on". Since an ontology can explicitly model union of classes, we can model state which exists only of “off” or “on”. In this case, the antonym is part of the standard antonymy relations, and the complement is explicitly defined.

Converse Relational

The converse relationship describes a relation from its different views. An example of such a relationship could be the “inheritance” which has two different views: parent and children. Special about the converse relationship as an antonym is the form the existence of one part; the other has to exist. As an illustration, in the example of inheritance there is no child without parents, and if there are no children, then the nodes are just nodes and not parents because we do not know if the next node in the graph will be inserted as its parent or its child. Converse antonyms are neglected concerning our decomposition.

Reverse Relational

The reverse relations describes antonyms on relations. Thus if concepts are in one relation of the relations being in a reverse relationship, they are in the other relationship as well. As the name implies, the relations in these cases are directed. In particular, if the one relationship is in one direction, then the other one is in the other direction. An example here is the Hyponym- and Hypernym-Relation. We neglect reverse relationships in our decomposition because they can be explicitly described in the ontology by relating them to a reverse relationship if needed.

Incompatible

Incompatibility as an antonym relation is a standard interpretation of classes in OWL. Therefore if not specified otherwise if to concepts inherit from two independent classes, they are incompatible. Thus incompatibility is included in our antonym relations.

Semantically this does not describe all the antonyms we have looked at in this section. Sadly the information sources unify those types of antonyms in one relation. This unified relation will be used in our approach until the identification of the different types of antonyms is possible. Until then we might have an occasional mix in of some other antonym from our dictionaries since, e.g., Wikipedia is described by humans, which works in such inconsistent ways.

Besides antonyms and synonyms, the most used relation in ontologies is the inheritance relations. Those can be split into specialization and generalization. The next section will discuss the generalization and its role in the decomposition.

6.5.4. Decomposing using Hypernyms

Using a more general concept to explain a specialized one allows the explainer to be abstract during an explanation. This is done with function $getHypernyms : Concept, WordType \rightarrow Concept*$ in Algorithm 2. Explanations like: “A human is an animal” are useful since now the reasoner can infer properties of the more general concept. The hypernyms can lead to primes or can be a prime, which again stops the decomposition in the direction of a generalization.

The recursive level of which Hypernyms are used during the decomposition effects the abstraction the decomposition will have. Abstracting concept will eventually lead to the prime THING/SOMETHING which can lead to decompositions like: “Someone does a thing to something.” These decompositions are not useless; a reasoner can now determine the different actors and the action. The interpretation of such a decomposition is that we describe an event which has two agents, an active and a passive one. This is different too: “A thing is done to something” since here the active agent is missing. This leads to smaller detailed differentiations in a decomposition like active or passive, which we neglect in our decomposition since they are syntactical.

6.5.5. Decomposing using Hyponym

To use hyponyms in the decomposition is a bit like using examples during an explanation. Hyponyms can be collected with function $\mathit{getHyponyms} : \mathit{Concept}, \mathit{WordType} \rightarrow \mathit{Concept}*$ in Algorithm 2. It eases the access to a reference. By describing abstract entities, the reasoner might lack the ability to find fitting references. Since some semantic theories like the prototype semantic 4.1.4 insist on references to describe the meaning of something, we can argue that with a hyponym these references can be included in the decomposition.

A hyponym describes different manifestations of the decomposed concept. Hence a hyponym can be added to the decomposition by noting that some of the entities described by this concepts are of the hyponym type. Therefore in the decomposition, we can add the hyponym by adding: SOME to it.

This can be used by the reasoner to test examples of the abstract description. If the reasoner finds an inconsistency with a meronym then he has found a counter example in his beliefs, which makes additional reasoning necessary for deciding to neglect the property of the hyponym in favor of integrating new information or if the new fact described by the concept changes fundamental beliefs which the reasoner is not willing to change. In this case, the decomposition can not be integrated into its beliefs, but the contradicting part can be extracted and explained, via an example (the hyponym).

For example, if someone with pointy ears is going to tell you that feelings are bad since he wants you to make a rational decision you might reason the following: Feelings are bad. Love is a feeling. Consequently, love is bad. Love is the precipitation of my body of some hormones. This leads to new entities of my species. This ensures the existing of my species. This can not be bad for me. Hence love is not bad. Thus not all feelings are bad. If we can create reasoner which can reason such implications, we can check the decomposition for plausibility. In contrast to the hypernym, where we check if its logical structure is valid for more abstract or general statements.

6.5.6. Decomposing using Meronyms

Decomposing a concept by describing its parts is a top-down approach to a definition. Meronyms can be collected with function $\mathit{getMeronyms} : \mathit{Concept}, \mathit{WordType} \rightarrow \mathit{Concept}*$ in Algorithm 2. Since a part of a system is always less or equally as complex as the system itself, the decomposition can break down a concept into its parts and decompose them to reach the goal of describing the concept with simpler concepts.

Meronyms describe parts of relations (see section 3.9 for more details). The meronyms are integrated into the decomposition with an own relation. Showing that a part of the concept decomposed is the meronym. This can be thought of as an explanation which includes describing all the parts of a concept if it has some. For example, a computer consists of a hardware and software part. The hardware part in the von Neumann architecture contains memory and a computational unit. In fact, these parts again can be decomposed and explained. Since each of these parts is maximal as complex as the whole process, decomposing the parts can be seen as splitting the complexity and with that reaching simpler concepts.

The part of a concept is not only defined by its meronyms but also by its definition. The definition contains more or less all of the relations above, depending on how the concept is explained. Definitions are mostly formed in natural language sentences, like an article in Wikipedia, or an explanation in a dictionary. These explanations consist of two part, the concepts they are made of and the syntax cluing them together. How the concepts are handled was subject of the last sections, the next section will discuss the use of syntax.

6.5.7. Decomposing neglecting syntax

The syntax of sentences allows us to discriminate details of language which are needed to express precise utterances. In order to formalize the meaning of concepts, this detail can be seen as the next level, for describing semantically close concepts like past tense and imperfect. Rogers and Hodges [333] model knowledge representations into two categories: structural description system and semantic knowledge. Syntactics is part of structured knowledge and as a result, falls out of the semantic part.

To describe syntactical details, we have to describe each concept which is part of syntax in primes. Then we can use them to describe the syntactical differences of concepts. This is left to work after the general feasibility has been shown.

This work concentrates on creating the meaning of concepts which might lead to syntactical incorrect decompositions. This can be imagined as a three-year-old human child: not grasping the whole expressiveness of his mother tongue, but able to discriminate itself from others, and pronouncing simple utterances. For example, an utterance like: “I hunger, need power.” would be acceptable in the goal of this thesis.

One argument for the neglecting of syntax is that we want to build an artificial intelligence which can create meaning for concepts not write elaborated syntactically correct phrases. Especially find references, connect concepts to events in memory, connect similar concepts, select word senses and use abstraction to producing multi-word utterances. Since concerning the broad meaning of concepts without syntactical details, we reduce the complexity of the problem by moving syntax out of the scope of this dissertational thesis.

The tasks of object recognition can be seen as the phase of one-word utterances of a human child. As we humans learn a language we start out with single words after 12 months of language development (mostly nouns) and continue to full sentences after about 36 month [211, 319]. During the first 12 month connecting references to concepts and connecting them to events in memory is done.

During the decomposition, every word is reduced to its stem or paradigm before it is processed further. For that reason, before searching for synonyms, antonyms or looking up the definition, the paradigm is searched. This means that sentences like “I am being normalized” reduce to “I BE normalize”. Here we loose the syntactical information but reduce the complexity for the decomposition.

Even with the syntactic complexity out of our concern, the decomposition can become too complex, if the graph does not stop growing. This can be imagined as one explanation leading to another until a concept is well described. To reduce this size complexity, the next section will discuss our termination conditions of the decomposition.

6.5.8. Termination conditions

The termination of the Algorithm 1 has been chosen in a way that the decompositions are kept as short and precise as possible. However, the theoretical termination condition is the total decomposition of the given concept. For some tasks, this might be disadvantageous, and the order of steps in the algorithm needs to be changed appropriately.

The manner in which a decomposition is created depends – as always in ontology engineering – on the skills of the engineer, his domain and cultural background in which the information sources have been described. Although, with this strict framework of decomposition we might get different results for the same concepts depending on the parameters of the Decomposition and the included Information Sources. We postulate that those results differ in such a way that an artificial reasoner can create equivalence classes from those decompositions. This means that the decompositions might vary in different domains, by syntax, which is irrelevant to the meaning the decomposition represents. The selection of information sources and the selection of the decomposition depth thus dictate the termination of the decomposition.

The Algorithm 1 does not have to terminate with the total decomposition of a concept. We rather define a parameter describing the decomposition depth to terminate earlier. This means we can decide how often the main recursion is run through and with that shorten the span of the created graph.

We could think of having multiple termination conditions, e.g., with multiple decomposition depths for different relations, for the different "directions" of the decomposition. In this way, we could parametrize the use of synonyms differently than the use of the other relevant concepts. However, new problems arise, e.g., with the combination of different relations types. Further, we are not analyzing this kind of threshold and leave research in this direction to future work.

We have mentioned that the result of the decomposition is a graph. To ease the understanding, we are going to look at a handcrafted example next. For more complex examples the reader is referred to Figure 9.3 and Figure 9.4.

6.5.9. Example decomposition

Now we will have a look at an example of a decomposition done manually to illustrate how the result of a decomposition could look like from an abstract view. In Figure 6.9 we illustrate an example of a decomposition of the concept to use in the sense of putting something to a purpose. We did remove the formal description we introduced above to restrict the example to the core concept and show an abstract view on the result of the decomposition. Figure 6.9 even reduces the edges to only one unlabeled type. In this example, we decomposed the concept used as a verb and selected a definition out of the Cambridge advanced learner's dictionary [394]. Here we neglect the modal particles like to. The decomposition has been created by recursively applying the decomposition algorithm. The result is a relative short decomposition because we did select the definitions which are used for the decomposition. Further, we selected the words to use from the definition by hand. Here the graph reads as follows: To put is to move something inside a place.

The example from Figure 6.9 shows a theoretic result of the decomposition. In a real decomposition the decomposition becomes more complex, since, e.g., the same concept is not added

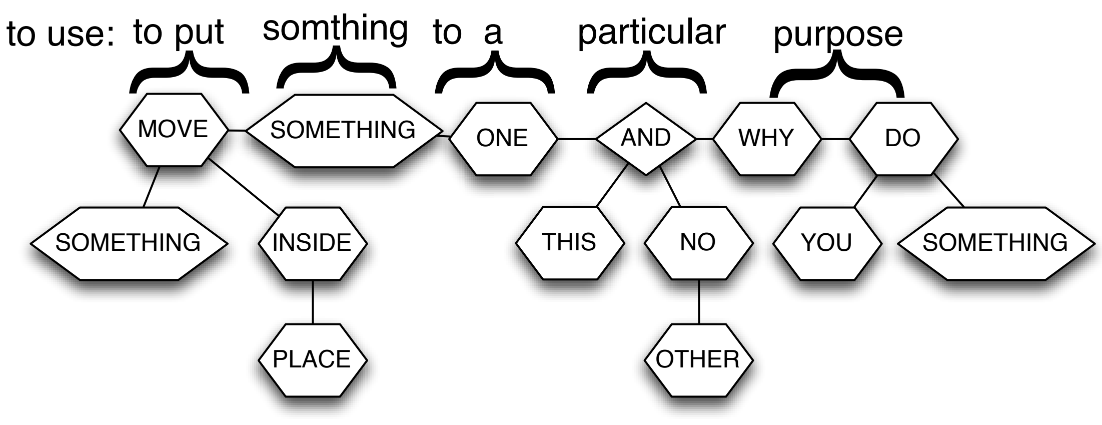


Figure 6.9.: Example of decomposition using a definition out of the Cambridge Dictionary [394].

twice like the "SOMETHING" node. The example in Figure 6.9 has been simplified from our formalism to describe semantic graphs, to ease the readability. However, there are further challenges, which need to be addressed. Next, we will have a look at the remaining challenges: The selection of the definition to be used, for the decomposition influences the meaning which is decomposed. Here the different information sources must use the same sense of the word for the decomposition. In this way, ambiguities are neglected.

There are many parameters, which need specification reaching the depth of the search for, i.e., synonyms and the recursion depth for crawling through the dictionary. Further manual decompositions are needed in case of a deadlock in the recursion of the algorithm. Here we need to find a way on how a human can be queried for his input. This has been subject to research in the work of Ghadah Altaiari [9]. Integration of context-dependent meaning is not yet considered by our algorithm. If context-dependent meaning needs to be described, or if the decomposition is context dependent, then context needs to be integrated into the recursion, which could lead to a different termination condition, e.g., if we consider the beliefs of an agent, we could terminate earlier if some concepts are already known.

6.6. Conclusion

In this section, we have built an algorithm which can create semantic graphs automatically by using existing ontologies and information sources like Wikipedia or WordNet. This has been done by others like Alani et al. [7] and has been subject to research until today [63, 212, 307]. However, none of those approaches used a theoretic grounding in a linguistic theory like NSM. This makes this approach unique and helps us understand more about the structure of semantic graphs, which leads to more insight on how meaning can be represented for AI. Also, this approach gives us the ability to define only ca. 65 concepts, to enable the agent to build its meaning of concepts.

For the goal we have set, we fulfill the first part of our abstract approach shown in Figure 5.1 with this algorithm. We create a semantic graph with includes the collection of definitions we gather from the information sources and all semantic relations we can extract from them. We have defined a formal representation of our graph and decided on a mechanism to terminate the decomposition namely the decomposition depth.

7. Marker Passing

The second part of my approach is to use the automatically created semantic graph representation of the connectionist meaning for reasoning. This section is structured in the following way: We start our by describing the pragmatic way of representing meaning in Section 7.1. Then we look at the marker passing algorithm in Section 7.2. Section 7.3 then details the parameters of my marker passing algorithm. Finally we conclude the pragmatics of my symbolic approach in Section 7.4.

7.1. Pragmatic Meaning Representation

Building up on the semantic graph representation discussed in Section 6.4 and the automatic creation of such a semantic graph described in Section 6.5 we now look at an example of what pragmatics means in natural language and carry this idea over to our representation of meaning. The goal of this section is to show how pragmatic meaning can be described using Marker Passing. Furthermore, it describes the theoretic basis for the interpretation of the internal mechanisms and the result of the Marker Passing and gives the theoretic background to understand why the algorithm has been created the way it is.

In this thought experiment, I will utter: “I am cold.” moreover, you are sitting next to an open window which lets cold air into the room. The implicature which you could reason out of my speech act could be: ‘Would you please close the window because I am cold.” It is easy to see that such a reasoning is not possible without contextual knowledge, in this case, the knowledge that the window is open, that is is colder outside and that it would get warmer if the window is closed. This chapter will look into this kind of contextual reasoning of meaning called pragmatics.

In the agent paradigm, we model two minds: one who utters, the other one understands. If something is known about the context of the utterance, then the understanding (interpreting) entity can use this context to create pragmatic meaning. Here the communication can be seen as: “the successful interpretation by an addressee of a speaker’s intent in performing a linguistic act.” [149] To clarify this we can adapt the conceptualization of a communication act from Marta Lenartowicz [228] where communication is seen as a single act, where the uttering uses semantic information to encode information in symbols (signifiers) and the listening agent interprets those symbols. Both agents use a context in which they concretize the semantics to pragmatic information. However, this context might differ.

Hence the observing agent can have access to the same contextual information as the uttering agent, which eases the interpretation, or the current contextual information of the understanding agent needs to suffice. Depicted in Figure 7.1 the two agents might use the same context, symbols and semantic to communicate. Then there can be three factors to differentiate the communication:

Different Symbols is the fact that different symbols (signifier) are used for the communication. This can be seen as in different languages where different words reference the same

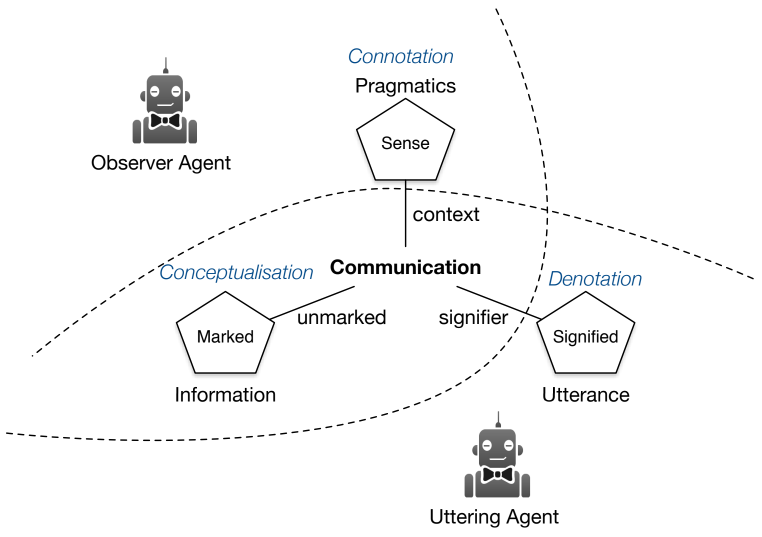


Figure 7.1.: Abstract communication act adapted from [228].

object, e.g., English: Apple, German: Apfel, French: Pomme. All reference the same object but use other symbols to describe the concept. Without a common context and a common semantic, the words can not be mapped. One area of research where this effect is wanted is cryptography. Here the communication is translated into languages only sender and receiver can decipher. Even in the same context and with the same semantics, the communication becomes subject to interpretation. We will ignore this kind of communication problem and leave this challenge to other researchers.

Different Semantics If the semantics are different, then the conceptualization of an object, might be different. This means in the abstraction of conceptualization, different properties of an object are emphasized. This leads to the problem of resolving the object reference, which is eased by using the same context and signifiers. This can be seen as the ontology matching problem [358]. This problem can be tackled with the here presented approach. Besides the structural comparison of ontologies, a semantic similarity between the concepts used as a basis for ontology matching. We will see that Marker Passing can provide insight into ontologies, their structure and the similarity of concepts used.

Different Context In a different context the same signifier with the same semantic might have a different meaning. This means by not sharing a context; the communication becomes subject to interpretation as well. The problem of integrating context[1] can be tackled with Marker Passing. Integrating contextual information in the interpretation of communication is the main goal of this work. For that reason, we will analyze in this section, how the semantic meaning represented in our semantic graph can be interpreted contextually.

We will neglect the utterance side during our analysis since we postulate the willingness of agents to communicate and leave the analysis of symbolic difference to other researchers. The semantic decomposition creates a conceptualization depending on the information sources given to it, which allows a semantic analysis to tackle the ontology matching problem. The

[1]Here context can be more than the word surrounding a target word, e.g., the special or temporal context of the utterance.

interpretation done by the listening agent to the utterance done by the uttering agent should yield to be context dependent. So that our artificial meaning is represented by the context-dependent interpretation of an utterance. For that, we need an approach to integrate contextual information, to be able to tackle a problem like WSD. We are looking for two types of usage of our semantic graph: Semantic similarity[2] and WSD.

We start out by describing the result of the semantic decomposition and the use of Marker Passing and how this can be used to describe a representation of meaning. However, first, we look at how the context-independent meaning is transferred into a pragmatic one.

Since semantics is the theory on how meaning is transferred, a semantic transference and interpretation process is required. There are four parts for the meaning of a word which are of concern to an AI:

Denotation The denotation represents the primary or basic meaning of a word. It can be seen as the definition of a word that is represented in some mental lexicon (or a dictionary). Denotation is the explicit meaning of a word, the shared explanation of the what is referenced. A dictionary holds denotations for the words described in it. In our representation of meaning the denotation is described by the definitions in the semantic graph.

Connotation The connotation is the abstract idea presented by the word. It can be seen as the conceptual representation of the meaning of a word. The connotation is the idea or quality; the associations brought to mind. It is private, depending on memory and experience and most of all context dependent. This includes the connectionist interpretation of meaning since here the meaning is interpreted as the unity of its relations to other concepts. The connotation in our approach is the result of a Marker Passing of one concept. It consists of the marked graph which is subject to interpretation.

Conceptualization To be able to come up with a conceptual representation of the meaning of a word, one needs to abstract from the word to a specific concept (*i.e.*, one needs to connect the word with a known concept). This process is named Conceptualization and helps to clarify a word within a language. In our approach, the conceptualization is the result of the decomposition of one word. It consists of a semantic graph holding all known information about this word, building its conception.

Pragmatics The meaning of words is not independent of the context the words are used in. In particular, a context-dependent representation of meaning (a pragmatic one) has to be created (e.g., mouse (computer) vs. mouse (pet)). The Pragmatics in our approach is described as the result of a Marker Passing where a concept and its context have been activated. It consists of the marked graph which is subject to interpretation.

Meaning itself needs to be represented appropriately (in a formal manner) to be handled by an agent. Since meaning is not precisely defined, this is subject to research. We will look at meaning in the linguistic sense, which can be defined as follows: Meaning is what the source of an expression (message) wanted the observer to infer from the expression [240]. This view is shown in Figure 7.1. Here the inference an observer can make about the communicated information is seen as meaning. If the observer has access to the context the uttering has been made in, then the interpretation becomes simpler. This context involves all relevant information

[2]Here we argue that semantic similarity measures are the core problem of ontology matching.

to the communication. From the perspective of the observing agent and in the worst case: if all uttering context is lost, then the observed utterance can only be seen as denotation.

One example problem in such an utterance is that the similarity of words needs to be established, by comparing their meaning. In this case, markers passed from one concept to another might interact. Thus a node should be able to react to activation to different markers.

Another example is the problem of WSD. In the WSD problem, we want to select a word sense of a target word, given a linguistic context it is used in. The linguistic context might, e.g., be the surrounding sentence. After the decomposition of the target word with all its definitions, we select one of them via Marker Passing fitting the best to the linguistic context. Doing this Marker Passing, we want to control which edges pass markers in which way.

The third example is a semantic sentence similarity measure, where multiple concepts might pass markers. To establish the similarity of the meaning of a sentence, we might not know when to stop passing markers since we do not know if the meaning of the sentence changes if another word meaning of one of the words in the sentence changes. This means we want our algorithm to have a variable and problem specific termination condition.

The fourth example of such an abstract communication is the proposition of a service. The service providing agent communicates the service interface to other agents. The interpretation and use of this service could then be seen as our listening agent. The listening agent then can interpret the concepts of the service description in its context. When looking at services, we might want to activate some concepts and not others like the input of a service, but not its precondition to see if we have fitting input arguments. To do so, our Marker Passing should be able to select which nodes pass markers.

The last example is the creation of a heuristic, where concepts are not compared to their similarity, but a rather complex construct, like a precondition of an effect or a service, is analyzed for their usefulness. This includes more symbolic reasoning on nodes and edges, which can only be done if the information is passed to them. This means we want the liberty to write any information needed onto the marker.

These five examples lead us to the description of the Marker Passing algorithm we designed in Section 7.2.

Conclusion

In conclusion, we describe pragmatic meaning as a result of the Marker Passing algorithm, where, e.g., the COI, as well as its context, has been activated. For the Marker Passing to be pragmatic, we need to pass the additional concepts to the Marker Passing, and the different concepts should influence the result. Thus our algorithm needs to provide parameters in a way that the markers can influence each other. The pragmatic representation of meaning is created through the influence the markers from the different concepts have on each other. In consequence, we want nodes and edges to be able to react to different markers. Depending on the parameterization of the Marker Passing algorithm, this might yield different results depending on when we stop passing markers leading to the need to specify specific termination conditions. Thus the pragmatic meaning not only depends on the concepts which have initial markers but also on the Marker Passing algorithm and its parameters, as well as the interpretation of the result. With this interpretation of pragmatic meaning in mind, we now build our Marker Passing algorithm. We describe the abstract Marker Passing algorithm in the next section and will analyze his parameters and its performance to represent meaning later in Section 9.

7.2. Marker Passing Algorithm

After creating the graph representing the semantic knowledge of the agent, this section will describe how this graph is used to select relevant concepts in a given context. These relevant concepts are specified as a subgraph of the original graph including the markers passed to them. We will first describe the Marker Passing algorithm at a higher level to name the different parts and get an overview of the needed components in Section 7.2.1. Then we dive into the implementation and all the details of this algorithm in Section 7.2.2.

7.2.1. High Level Marker Passing Algorithm

To start the algorithm, we have to create the underlying graph and prepare it with a set of start markers, which then are propagated through the graph. Figure 7.2 shows an abstract representation of our algorithm. Here the idea is to propose a naming of the parts of the algorithm to be able to discuss those parts later on in more detail.

This algorithm is a generalization of the algorithm described by *F. Crestani* [61][Figure 5, p. 461]. Crestani describes the Marker Passing in four steps: Pre-adjustment, spreading, post-adjustment and termination condition evaluation. This is quite general and can result in inaccurate interpretations of the algorithm. Consequently, we introduce a more precise description of the algorithm by breaking the activation down into multiple steps without losing generality.

Crestani's algorithm is based on the following principle: Starting from a start activation, a concept has a threshold (seen as an upper limit of activation in a node to decide if the node is activated), with each incoming activation the activation level of the node builds up. If the threshold is reached the node is selected as activated and is spreading in the next spreading step. This means that the node passes all its markers on to its neighbors. This is done until a termination condition is reached [61].

Figure 7.2 shows the input and output of our algorithm[3], like documents, named with the content for the input. At first glance we look at the algorithm from the outside, specifying its interface.

Marked Graph (input) This is the input representing the graph, which is used to determine concepts and edges which are used to hold and conduct markers. This graph has to contain a set of markers to start from. These markers are called the **start markers**. The graph specifies which concept can pass markers to which concept. This is the basic restriction of this kind of algorithm: A concept can never pass markers to a concept when it has no connection to this concept.

Node Interpretation The underlying graph can contain different concepts and edges. Each concept or edge can have a different Marker Passing behavior. To specify this behavior, the algorithm gets a set of concept interpretations as input. Here each concept type specifies its interpretation by defining what it does with markers passed to it, with that when it is activated and how it passes marker to other concepts. These two functions allow each concept (or concept type) to react to different markers or the same markers differently.

Termination condition For our Marker Passing algorithm to terminate the algorithm needs a specification when it has reached its goal. The goal is to mark the sub graph which

[3]Our algorithm is based upon Crestani's work but details some of the design discussions in the algorithm.

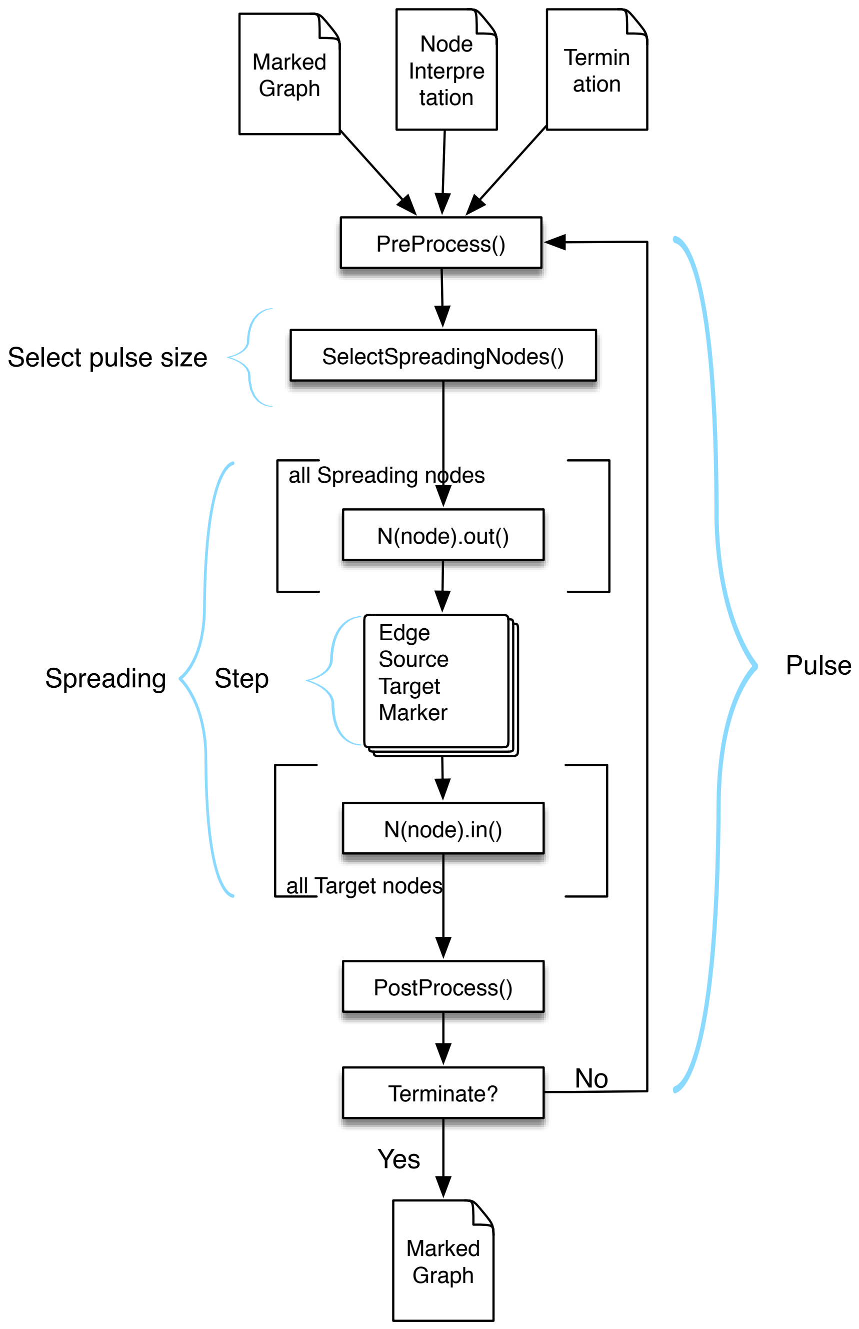


Figure 7.2.: Abstract description of our Marker Passing algorithm.

is relevant to the given start activation. If no termination condition is given, the activation spreads infinitely until no concept is activated anymore. This can be specified in a termination condition but does only in some cases accomplish the goal of the Marker Passing [26].

Marked Graph (output) The output of the Marker Passing algorithm is again a marked graph, with the updated markers. This graph now can be interpreted to determine the answer to a question which has been encoded in the markers and the graph, to begin with. By looking at this graph and the markers on it, we can, e.g., decide which other relevant concepts contain markers on them.

All in all, the algorithm described in this section can be seen to move around markers in a graph regarding given rules. How the start markers are placed, in which way the markers are passed over the network and how the result is interpreted is problem specific. Five example applications can be found in Section 10, e.g., for the application of this algorithm to create a semantic similarity measure, which has been published in [104].

Now that we have looked at the Marker Passing algorithm from its interface, we can have a look at the algorithm itself. We start out by looking at the different phases the algorithm can be separated into.

Pulse The pulse is defined as one iteration from the preprocessing to the check of the termination condition. All other phases are sub-phases of the Marker Passing pulse. This term has been coined by *F. Crestani* in his survey on activation spreading approaches [61]. The pulse can be imagined as one activation of a set of concepts, where each concept is imaged to light up if it activates. In this visualization within each pulse, all the activated concepts light up at the same time, creating a pulsing light which "wander" through the graph. This visualization gives the Marker Passing pulse its name. The pulse has two intermediary steps: the **Pre-** and **Post-Processing** here the algorithm can integrate nonactivation specific tasks like cleaning up the graph or normalizing the activation.

Pulse Size Selection Since during one pulse the termination condition is not checked, we need the ability to change the pulse size. This means selecting the concepts which activate during the next pulse. If all concepts which are active, activate then it could be that the conditions of the termination condition are met without noticing, because further concepts are activated, where the result does not meet the termination condition anymore when it is checked, after the pulse. The smallest pulse size is to activate each concept in its pulse, meaning that after each activation the termination condition is checked. This option has the problem that the activated concepts are needed to be brought into an order, where the order influences the result.

Spreading The spreading describes the Marker Passing of all activated concepts. During the spreading, all active concepts calculate their out-function and specify which markers are passed to which concepts concurrently. Afterwards, all concepts which have markers passed to them (these concepts are called **targets**) activate their in-function and collect their markers and update their internal activation level.

Spreading Step The spreading consists of one spreading step. The spreading step contains all markers which were passed by the out-function of all active concepts during this pulse.

The spreading step contains all needed information for the in-function to process the incoming markers. Here each marker is mapped to an edge (the edge represents an edge between two concepts in the underlying graph) over which the marker has been passed and the target it is destined to reach. The link contains sources and targets of an edge, but since there can be multiple of them, the step needs to specify which target should be the destination of the marker and from which source it has been spread.

After having an informal overview of the functioning principle of the abstract Marker Passing algorithm in our extension of Crestani's algorithm, we now formalize the Marker Passing in the next section.

7.2.2. Implementation of our Marker Passing Algorithm

To start with the formalization, we start by defining the information stored on a nodes *Data* in the following signature:

Signature $Information =$
sorts:
$Data$
$Marker$
opns:
$getMarker : Data \rightarrow Marker*$
$setMarker : Data \times Marker* \rightarrow Data$

At the same time, we define how we describe a marker. This is done by defining *Marker* as the sort of the marker object since the marker is defined regarding the problem specific implementation of the information encoded on the *Marker*.

This *Marker* as a sort can be implemented with any data type which is needed. To implement a spreading activation, this sort could be a decimal value. This decimal value then can be interpreted as the activation level passed from one concept to another. For another example, the marker might have an integer type which could be interpreted as markers for a Petri net.

The sort *NodeData* maps the information encoded in markers (*Data*) to the concepts. The *Data* sort specifies the information encoded on a marker. The method *getMarkers* allows us to get the markers of a type of a concept. Equivalent the method *setMarkers* allows us to set the markers of a concept.

The marker is typically represented by information objects from the marker signature and usually assigned to the concepts of the graph structure and passed on via the relations. This *Graph* represents the ontology or semantic graph which is the output of the decomposition. We have defined this graph formally in Section 6.4.

Here the signature *Graph* is the result of the semantic decomposition and is represented in the formalism described in Section 6.4. From this graph the two sorts *Concept* and *Relation* are taken as specified in the Figure 6.3 in our minimal type graph.

To enable the assignment of markers to concepts of a graph, we define a signature *MarkedGraph* which extends *Graph* and *Marker* and is defined as follows:

The function *getNodeData* gets the *Data* of a given node.

The Marker Passing algorithm then transforms markers on a concept in a way specified by, e.g., the implementation of the marker, the concept-interpretation, the in- and out-functions

Signature $MarkedGraph = Graph + Information$
    sorts:
        $NodeData$
    opns:
        $setNodeData : NodeData \times Concept \times Data \to NodeData$
        $getNodeData : NodeData \times Concept \to Data$
, $getMarkers : NodeData \times Concept \to Marker^*$
        $setMarkers : NodeData \times Concept \times Marker^* \to NodeData$
    vars:
        $M : NodeData$
        $N_1, N_2 : Concept$
        $D : Data$
    eqns:
        getNodeData(setNodeData(M,N_1,D),N_1) = D
        $N_1 \neq N_2 \Rightarrow getNodeData(M, N_1) = getNodeData(setNodeData(M, N_2, D), N_1)$
        getMarkers(M,N_1) = getMarkers(getNodeData(M,N_1))

of the Marker Passing algorithm. The passing of markers is called spreading. Moreover, the signature *MarkerPassing* describes the functions which implement the Marker Passing behavior.

Signature $MarkerPassing = MarkedGraph + Boolean$
    opns:
        $preAdjustment : NodeData \to NodeData$
        $postAdjustment : NodeData \to NodeData$
        $getActiveConcepts : NodeData \to Concept^*$
        $getPassingConcepts : NodeData \to Concept^*$
        $outFunction : NodeData \times Concept \to (Relation \times Edge \times Markers)^*$
        $edgeFunction : NodeData \times Relation \times (Edge \times Markers)^* \to (Concept \times Edge \times Markers)^*$
        $inFunction : NodeData \times Concept \times (Edge \times Markers)^* \to NodeData$
        $afterSend : NodeData \times Concept \to NodeData$
        $terminationCondition : NodeData \to Boolean$
    vars:
        $N_1, N_2 : Concept$
        $Ns : Concept^*$
        $M : NodeData$
        $M_1 : Data$
        $R : Relation$
        $E : Edge$
    eqns:
        $getPassingConcepts(M) \subseteq getActiveConcepts(M)$
        $N_1 \neq N_2 \Rightarrow getNodeData(M, N_2) = getNodeData(inFunction(M, N_1), N_2)$
        $N_1 \neq N_2 \Rightarrow getNodeData(M, N_2) = getNodeData(afterSend(M, N_1), N_2)$
        $(R, E, M) \in outFunction(M, N_1) \Rightarrow source(E) = R \wedge target(E) = N_s$
        $(N_1, E, M_1) \in edgeFunction(M, R, (E, M_1)) \Rightarrow source(E) = R \wedge target(E) = N_1$

The set *Relations* contains all the concepts which have a type inherited from a relation. The function *setNodeData* sets a *Data* on the given node returning the updated *NodeData*.

The function *preAdjustment* is called in each pulse and enables us to adapt the *NodeData* of the graph if needed. This can be used to implement general marker behavior which is concept independent like a normalization of *NodeData* over the graph. The *preAdjustment* uses the current markers of the graph and produces an updated *NodeData*. The *postAdjustment* is similar to the *preAdjustment* but is called after the spreading, since here again; we can change the markers of the graph before the termination condition is checked. For more details see Section 7.3.7.

The *getActiveConcepts* function filters the concepts which have been activated from the *MarkedGraph*. All these concepts have reached their spreading threshold and can potentially spread in the next pulse.

The *getPassingConcepts* function then selects from all active concepts the ones which will pass markers in the next pulse. The passing concepts are always a subset of the active concepts, which is guaranteed by the first equation.

The *outFunction* implements the concept behavior on how markers are passed on to its neighbors if the concept is selected as passing concept. For more details see Section 7.3.4. The fourth equation of this signature describes that the edges always have a source in a relation R and target the concepts which are related with the relation R.

The *edgeFunction* describes the behavior of the edge when markers are passed over it. This allows us to implement, e.g., edge weights. For more details see Section 7.3.5. The fifth equation of this signature enforces that markers are only passed from an edge to its targets over the edges of the relation R.

The *inFunction* and *outFunction* is the place where we can implement the behavior of the concept when markers are passed to it. This in-function takes the concept, the markers passed to it and the edge the marker is passed over as input and converts the incoming markers into a form which lets the algorithm determine the activation of the node. For more details see e.g. Section 7.3.3.

The function *afterSend* can be used to update the markers of a concept after its out function has been called. It is called with the current markers and respective concept and updates the markers accordingly. This could be used, e.g., if markers are rejected by the destination and should be given back to the passing concept.

The *terminationCondition* defines when the spreading process has finished. It is based on the current markers. For more details see Section 7.3.6.

The first function enforces that the passing concepts are a subset of the active concepts so that all passing concepts need to be activated first. The second and third equation of this signature ensures the compatibility of the *inFunction* and *afterSend* functions with the *getNodeData* function. The fourth function ensures that markers are passed by the out function which have their source in the edge and are passed to the target of this edge. The fifth equation ensures that *edgeFunction* only passes markers to targets of their edges.

In conclusion, we can say, the Marker Passing is a way of marking concepts in a graph, regarding some rules. These rules are problem specific and change depending on the implementation of the different points of the algorithm. There is one rule which has to be always true: Markers can only be passed between concepts connected with a relation.

The pseudo-code of Crestani's algorithm can be found in Appendix A with Algorithm 22 and our extension is described in Algorithm 5. The implementation of the algorithm is available[4].

With this formalization, we can describe a wide range of algorithms ranging from Petri-Nets or Artificial Neural Networks to Spreading Activation mechanisms like PageRank-Algorithms.

[4] `git•gitlab.tubit.tu-berlin.de:johannes_faehndrich/semantic-decomposition.git`
Please contact the author to gain access to the source code.

Next, we will look at the different parameters as variation points of the algorithm.

The Marker Passing algorithm is distinguished by four phases. These are depicted in Figure 7.2 (see page 120): (1) the *pre-processing* for preparing (priming) the graph; (2) the *selection of the pulse size*, which defines which concepts will pass information within the pulse; (3) the *pulse* itself, where all spearing concepts are activated and pass on their markers (each pulse consists of an *activation step* for each active concept); and (4) the *post-processing* step, were the results i.e. are normalized.

We start out with the algorithm proposed by Crestani [61]. Crestani's three step algorithm is abstract enough that many activation spreading interpretations are subsumed by it. Hence starting from Algorithm 4 we dig down into more details, to foster a more precise but still abstract algorithm.

Algorithm 4 Marker Passing- adapted from Crestani's Spreading Activation Algorithm

```
Name: Marker Passing main loop Input: NodeData $M$ Output:NodeData
1: while ¬ terminationCondition($M$) do
2:     $M$ = preAdjustment($M$);
3:     $M$ = MarkerPassing($M$);
4:     $M$ = postAdjustment($M$);
5: end while
6: return $M$
```

The Algorithm 4 takes a NodeData M as input.

This is a graph consisting of concepts and directed links connecting the concepts as specified by the minimal type graph depicted in Figure 6.3. This defines that everything is a concept and with that relations are special concepts as well.

Each concept might carry information in the form of markers. M contains the start markers. Also, Figure 7.2 describes the Node interpretation and the Termination condition as input. Those are seen as design discussions on how the algorithm is implementing the spreading functions (in-, out- and edge-function), its threshold and the information given by the markers. The additional termination condition is a meta parameter of the algorithm, which is specified separately. For simplicity reasons, we take those parameters to be given and use them in the pseudocode here.

Algorithm 5 describes our extension of the spreading activation algorithm of Crestani [61]. Active concepts are defined by getActiveConcepts().

The Algorithm 5 holds two maps $pulse_{out}$ and $pulse_{in}$, which hold the markers passed during a pulse. The function "addAll()" is a simplification of an iterative get() and add() and the insertion of the remaining tuple into the appropriate set of, e.g., all markers of the current pulse. We use a map as a key-value store with such an adapted addAll() function in e.g. the $pulse_{out}$ variable. The addAll() function gets the values of the given keys and adds the markers on top of the old value. This is in contrast to replacing them. In line 3 of Algorithm 5 we add the result of the *outFunction* in the form of $(Relation \times Edge \times Markers)^*$ to the map $pulse_{out}$, where for each edge the markers are sorted. We separate the Algorithm 5 into four blocks each consisting of one loop:

Line 2 - 4: All passing concepts activate their out-function and the result to the current pulse stored in the variable $pulse_{out}$. This is the input for the edge functions of the appropriate relations of the next step.

Algorithm 5 Marker Passing Algorithm

Name: MarkerPassing **Input:** NodeData M **Output**:NodeData

```
1: $pulse_{out}$ = new Map<Concept,$(Edge, Markers)^*$ >();
2: for all sourceConcept $\in$ getPassingConcepts($M$) do
3:     $pulse_{out}$.addAll(outFuntion($M$, sourceConcept));
4: end for
5: $pulse_{in}$ = new Map<Concept,$(Edge, Markers)^*$ >();
6: for all $e \in pulse_{out}.keyset()$ do
7:     $pulse_{in}$.addAll(edgeFunction($M$,$e$,$pulse_{out}.get(e)$));
8: end for
9: for all targetConcept $\in pulse_{in}.keyset()$ do
10:    M = inFunction($M$, targetConcept, $pulse_{in}$.get(targetConcept));
11: end for
12: for all sourceConcept $\in$ getPassingConcepts(M) do
13:    $M$ = afterSend($M$, sourceConcept);
14: end for
15: return $M$
```

Line 5 - 9: Each marker passed by the current pulse is given to the appropriate relation it is passed to, and this relation activates its edge-function. The result of the edge-function is added to the pulse which is used as input for the in-functions of the targets of this relations.

Line 10 - 12: The concepts which are targets of the relations passing markers are given the markers passed to them and activate their in-function.

Line 13 - 15: The after-send-function is activated to fix the markers on the source concepts if needed.

In the applied type of Marker Passing the concepts and edges of the underlying graph do not have to be disjoint. This has the benefit, that if needed an edge can be interpreted as a concept and pass markers to other edges. Since the part of our Marker Passing algorithm is executed in a loop displayed in Algorithm 4 we do run through those four loops until the termination condition returns true. In each iteration of this loop, we have the methods pre- and post-Adjustment to change the Markers on the graph outside Algorithm 5.

The here described algorithms have variation points, which are not specified in this abstract algorithm and need specification for the algorithm to work for a specific problem. We call those variation points the parameters of the algorithm, and we will analyze them in the next section.

7.3. Parameters of Marker Passing

The generic Marker Passing algorithm introduced in Section 7.2 has variation points which allow a specialization for different areas of application. These parameters are dependent on each other. In this section, we will discuss possible implementations of algebras of the signatures introduced in Section 7.2.2. Simple examples of such variation points are the *selectActiveConcepts* function, the *terminationCondition* and the *markers*: Here the *selectActiveConcepts* function needs to interpret the *markers* to decide if a concept is active or not. These examples show that the variation points can inter-depend. The needed variation points of the Marker Passing described in Section 7.1 are the following:

Data: describes the marker and with that the information available to marked node.

Pulse size: selecting which nodes pass markers.

In-Function: describes how the node handles incoming markers.

Out-Function and After-Send: describe how the nodes pass markers and what happens on the node after passing them.

Edge-Function: describes how the edges handle markers passed over them.

Termination Condition: describes when to stop the passing of markers.

Pre- and Post-Adjustment: describes what happens before and after a pulse.

The selection on how those parameters are implemented influences the behavior of the Marker Passing, regarding the problem at hand.

Regarding the variety of the parameter, which could be selected, we restrict our discussion here to general rules and dependencies which are necessary to understand to make informed design decisions for the application of this algorithm on a domain specific problem.

This section is structured in the following way: We start out with the discussion on the type of information encoded on the markers in Section 7.3.1. Section 7.3.2 discusses the pulse size and in Sections 7.3.3 to 7.3.5 we then analyze the In-, Out- and Edge-Function. Afterwards we analyze the termination condition in Section 7.3.6 and the Pre- and Post-Adjustment in Section 7.3.7. We then look at an example of how the Marker Passing can be used in Section 7.3.8. After the example we discuss the interpretation of marker, concept and edge in Section 7.3.9, Section 7.3.10 and Section 7.3.11.

7.3.1. Data

The data sort described in the signature *NodeData* in Section 7.2.2 contains the data sort which formalizes the information encoded on a marker. The information modeled here influences the whole Marker Passing algorithm in the spreading functions, the termination condition or the interpretation of the result. One of the design decisions during the implementation of a Marker Passing algorithm is, therefore, which information should be encoded in which way.

Examples of data on the markers are a boolean value of tokens assigned to places in Petri-Nets, or double values like in Artificial Neural Networks (ANN) which describe the activation of a neuron. We can not describe which information should be implemented on the marker since this is highly problem-specific. But we will describe an example in Section 7.3.8.

7.3.2. Pulse size

Selecting which nodes pass markers influences which nodes could be active next. Consequently, it might make a difference if two concepts are activated concurrently or sequentially. Selecting which nodes are counted as active, meaning they could pass markers and selecting which one do are two steps towards deciding the pulse size.

A standard way, e.g., in ANN or Petri-Nets is to give a threshold when a node is active. This threshold is calculated by the function *getActiveConcepts*. The result of the *getActiveConcepts* function is a list of concepts which could pass markers in the next pulse. Which one of them are passing markers is determined by the function *getPassingConcepts*. The function *getPassingConcepts* has the active nodes as input and returns those of them, which should pass markers.

7.3.3. In-Function

The *in-function* of a concept describes how the markers are processed once they have been passed to the concept. The in-function is dependent on the concept interpretation. This means every concept type can have his own implementation of an in-function.

Here, we have to decide how the incoming markers influence the selection of active or passing concepts. Depending on how these two functions (*getActiveConcepts* and *getPassingConcepts*) are implemented, the in-function can prepare the markers on the node to reflect the new activation level of the node, to be used in the next selection of the active passing concepts.

Some of the design decisions which can be made in the in-function are discussed next.

Edge interpretation The in-function gets the information over which edge a marker has been forwarded. This can be interesting information if different edge types exist. The in-function is the part of the Marker Passing, where this information is available, and where the edge type might influence how markers are handled on the node. An example would be that the role of the edges of a relation might make a difference like in the example in Figure 6.2 (p. 98) where we can interpret in which role of the relation “give” we pass marker to, e.g., source and target.

Marker interpretation The feature which is unique to the in-function is that it can influence the activation level of the concept by interpreting the symbolic information passed by the marker. In particular, depending on the marker types and the information encoded on it, the implementation of the in-function is responsible for interpreting the markers and recalculating the activation level for the next pulse. One example decision is whether the markers of an earlier spreading step are combined with the current markers, or whether the activation level of a concept depends only on the current markers. We call this behavior cumulative vs. replacing. If the markers are replaced, then the old markers are ignored, and the activation level of a concept only depends on the current markers. If the markers are aggregated with old ones, then the marker interpretation is called cumulative.

Symbolic Information The in-function is one of the places where the symbolic information on the markers can be changed. Here we could, e.g., write the current marker in a concept history of markers, to detect loops. The markers passed to a concept do not need to be interpreted in the same way for each concept. Depending on the information passed by the marker, the concept the marker was passed over and the concept itself, different markers can be integrated differently in the activation level of the concept. An example could be that if a loop is detected with one of the markers, the marker can be marked for the after-send function to give it back to its previous concept.

The in-function hence depends on the marker, edges and the concept interpretation. The in-function can only interpret the markers passed to it by the out-function. Both of these functions are complementary concepts of influencing the marker distribution on the graph. Since between the out-function and the in-function, there is a potential edge-function, one can not predict the marker flow from either perspective without knowing how all those functions are implemented.

7.3.4. Out-Function and After-Send

The out-function implements the behavior of a concept when it is passing markers. This means this function is only executed if the concept is active and has been selected as a passing concept for this pulse. The out-function takes the current activation level (the set of markers at the concept in question at pulse *t*) and specifies how markers are passed over connecting edges of the concept.

Similar to the in-function, the out-function depends on the concept interpretation as well. The out-function is one of the places in the algorithm where the symbolic information encoded on the markers can be interpreted. There the implementation of the concept specific out-function gives us the possibility to adapt to the markers on the concept and the outgoing edges of the concept. If for example, the edges connecting concepts specify a weight for a maximal capacity of the edge, the out-function can distribute the markers of the concept proportional to these weights.

Some of the design decisions which can be made in the out-function are discussed next.

Output to which edge The out-function decides which edge is used to pass markers. This could depend on the markers available on the concept. This enables us, e.g., to propagate "antonym-Markers" which only pass over antonym relations between concepts to only those edges.

Marker creation We need to decide if the out-function is allowed to create new markers or if it can only modify the way existing markers are passed to the neighbors. This can be done by copying existing markers or by creating a new kind of marker. Creating new markers allows us to react to the existing marker combinations. This could, e.g., be helpful if a common ancestor in a tree structure is searched. When markers are passed to other concepts, one design decision is whether the markers are removed from the passing concept, or if they are copied to be passed on.

Edge interpretation The out-function can depend on the edge type it passes markers to. This allows us to pass different markers over different edges types. Like the in-function which interprets markers depending on the edge they passed over, the out-function can pass different markers or different amount of markers to different edges. This enables us to, e.g., pass markers to outgoing edges and not back to the incoming edges.

Discrete vs. Continuous In the out-function, we have to decide if we pass on markers as a whole, or if we can split markers up, e.g., into new smaller markers. This is correlated with the design decision if the out-function can create new markers. Additionally, this depends on how the symbolic information on the markers is specified.

Marker interpretation To be able to react to different markers on the concept, the symbolic information on a marker needs to be analyzed in the out-function. If for example we want to implement a negation-concept used to integrate syntactic information into the Marker Passing, the out-function can look through the concept history of the marker, to see if a negation concept has been passed right before this concept and with that change the output of the out-function depending the wished behavior of nested negations. This means that markers passed to the edges can be modified before they are passed to the neighbors. This enables us to create something like a decay of markers by, e.g., reducing the activation of the markers each time it is passed to another concept.

Relative vs. Absolute When a node is activated, the out-function can pass an absolute amount of markers, e.g., proportional to the edge weights meaning that a fixed number of markers are passed, independent of the markers on the concept. A second design decision could be to pass a relative amount of markers depending on the markers available on the concept. With this propagation mode, the weights of the edges can be seen as a distribution of the existing markers.

After the out-function has given the markers to spreading step, the edge functions distribute them to the in-functions of the targets of the edges. Now after the in-functions have completed their work, there could be the case that we want to manipulate the markers of the passing nodes again. This could be the case, e.g., if a transition is not fired in a Petri-Net, the markers in the input places are not consumed but could be given back to where they came from. Another example could be to remove markers from concepts which have been passed on but have not been deleted by the out-function.

In such cases, the function *afterSend* is used to update the markers of the passing concepts, again, after the spreading step has been executed.

7.3.5. Edge-Function

The edge-function implements the behavior of edges when markers are passed over them. This means the edge-function is invoked during the spreading and passes markers from its sources to its targets. To decide how the markers are passed, the edge has the information which markers are passed to it, which concepts passed the markers to it, which concepts are its target concepts and its own type. Depending on this information the edge-function can change how the markers are passed to its target concepts.

The edge-function is activated after the out-function of all spreading concepts is activated, and before all in-function of the target concepts are activated. Here we have to emphasize that the edge-function of one edge does not have access to all the information encoded in the spreading step. The edge function has only access to the spreading steps, where it is the declared edge of the step. Some of the questions which could be answered by the edge function are: How are multiple markers aggregated when they pass over an edge? How does the aggregated marking change when it passes over an edge? Which concepts to forward the changed marking to? How to distribute the changed marking among those edges? Because there can be no answer to those question without a specific problem which should be solved, we now look at the available information which could be used to answer such questions.

Dynamic behavior An edge-function can implement dynamic or static behavior. Static means the edge-function reacts in the same way regarding the same spreading step (e.g., this is how most Artificial Neural Networks (ANN) work). Dynamic behavior could mean that the edge-function becomes stateful and passes markers depending on, e.g., when it has been activated the last time (e.g., this is how ANN with inhibitors work). Inhibitors in ANN reduce the probability of a neuron to be activated depending on the time of the last activation.

Marker propagation The edge-function controls only the markers placed on it by the out-function of its source concepts. Here we can decide how the markers are passed to its referenced concepts. This allows us to, e.g., implement something like an edge weight in the edge, not the out-function.

Roles The edge-function has to interpret the type of which the different concepts are connected to it. In a simple example, these types are source and target of the edge. However, in examples like described in Figure 6.4, where the edge “give” of type “Relation” has three different references to concepts, the edge-function implements the marker behavior on such an edge.

The edge-function depends on the implementation of the underlying graph, and its edges. In our Knowledge Graph defined in Section 6.4 we can have multiple sources, over which markers are passed to an edge and multiple targets to which the edge passes markers depending on the edge type on the type of relation the concepts are connected to the edge.

7.3.6. Termination Condition

The termination condition encodes when the Marker Passing algorithm should stop propagating markers. The type of termination condition depends on the use case of the Marker Passing, e.g., if relevant concepts in a semantic graph should be identified, we could choose a termination condition which terminates after a few spreading steps to localize the markers around the context of the initial concepts. We can define the termination condition as a function which maps the marked graph to boolean values which indicate if the termination condition is satisfied.

If a desired goal condition is reached, regarding the markers in the graph, and the termination condition is amiss, the markers can change with the next pulse, and the goal might never be reached. The same is true for a termination condition which stops at pulses too early. This means the termination condition depends on the pulse size (here the selection of active and passing concepts).

Furthermore, the termination condition should identify the goal we want to achieve with the Marker Passing. If for example, the goal is to calculate the length of the shortest path in a graph, the algorithm should stop if markers of the two starting concepts meet.

During the calculation of the termination condition, all information on the marked graph is available. In addition to the markers on the graph and the graph itself, the termination condition might be stateful, for example counting the activation steps. This gives us the ability to react to any change done in one pulse.

One basic decision here is which information is taken into account by the termination condition. Depending on this, we have to decide under which condition the termination condition evaluates to true. The termination condition than might have the following properties:

Dynamic behavior The termination condition might not be a fixed kind of threshold but might develop with the changes of markers in the graph. It might, e.g., be dependent on the start activation, depend on the markers distribution velocity or simply be probabilistic.

Interactive The termination condition might be interactive so that a user might stop the Marker Passing by choice. This might be of interest if the problem is computational complex and a tradeoff between precision and resources has to be made.

Time based In some problems, the termination condition is simply a time interval which determines how long markers are allowed to spread.

The termination condition is highly problem-specific. Therefore it might be canny to have multiple termination conditions, which guarantees a termination of the algorithm.

7.3.7. Pre- and Post-Adjustment

The pre- and post- adjustment are like the termination condition two phases of the algorithm where the entire graph and its markers are available for analysis. Here we can implement changes we want to happen to the marking in between pulses. The design decision to make here is how to change the markers of the graph before and after the result. The pre-adjustment can be used to change the markers to influence the active and with that the spreading concepts. The post-adjustment, on the other hand, might influence when the termination condition is fulfilled.

When using these adjustment phases, we have to be careful to not contradict the termination condition or the thresholds of the concepts. Since with the “right” pre- and post-adjustment we can render each termination condition useless and at the same time make the spreading pointless by creating a condition where the marker is passing only consumes resources without producing results.

7.3.8. Example Marker Passing

Now we are looking at a simple example of this Marker Passing algorithm. This is done so that the more complicated discussions are founded on a concrete idea on how this algorithm works, what the effect on the parameters are and how the result could look like. In this example, we want to find similar concepts to a pair of concepts given. For this reason, we want to answer the question: Given two concepts, which concepts are similar to those two? We selected this example because we want to show that markers can carry more than decimal values like in standard activation spreading algorithms or neural networks. The example is based on the example decomposition shown in Figure 6.5 on page 100.

We start out by describing the parameters of this example:

Graph The graph is given by the decomposition of the two target concepts, e.g., ”midday” and ”noon” and has the concepts and relations like shown in Figure 6.2 on page 98.

Type-Graph We have an Attributed Type Graph with Inheritance $ATGI = (TG, Z, I, A)$ like specified by our example in Figure 6.6 on Page 101.

Data The markers encode the concept they have started from as color. The marker count is represented by a decimal value. We implement the signature *Data* as a tuple: $M = (Origin, Activation)$ where *Origin* is the concept where the marker is created upon which is depicted by the color of the marker.

Pulse size The pulse size is selected to be all active concepts. This means that all active concepts are passing concepts in one pulse. Here we have selected an activation function which uses a threshold for each concept which, if reached, activates the concept. The activation threshold is $\tau(n) = 5.0$. This means concepts with more than a sum of five activation of a color are considered active. Here we use the threshold in the absolute value of activation. For instance, if a concept is activated with -7.5 then it reaches the threshold as well.

In-function The in-function collects all markers passed to the concept and sorts them by color and adds the activation to the activation of the respective colored markers already placed at the concept.

Out-function The out-function spreads all markers equally distributed over all relations of the concept. The markers of each origin (color) are spread separately. This means in one pulse one concept is a passing concept which spreads the color of markers it has been activated with.

Edge-function The edge function changes the markers with a multiplicative weight. In particular, the edge function is defined by the factor depending on the edge type:

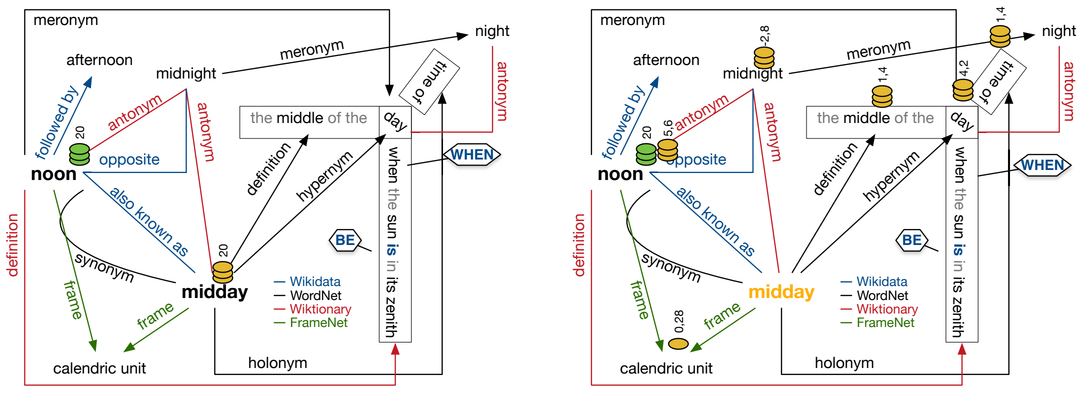
Edge type	Factor
Antonym Relation	-1
Definition	0.5
Semantic Relations	0.5
Named Relation	0.1
Synonym Relation	1

The edge function, therefore, changes only the activation of a marker. The weights have been chosen to demonstrate the effect of the edge function and show in our example how a use case of a semantic similarity can basically be implemented.

Termination Condition: The Marker Passing terminates in our example after no concept is activated anymore.

Pre/Post-Adjustment: is not needed in this example, so we leave the function to be the identity function.

With this example we can start out at the beginning with a start marking of 20 green markers of concept “noon” and 20 orange markers on concept “midday” on the graph shown in Figure 6.2 on page 98. This leads to the following graph shown in Figure 7.3a.



(a) Start marking. (b) Marked graph after first spreading step of midday.

Figure 7.3.: The example graph (based on the example graph shown in Figure 6.2) with its start marking is shown in Figure 7.3a. The result of the first spreading step after the concept midday as been activated in shown in Figure 7.3b.

In Figure 7.3 the markers are depicted by three piles of chips with the “color” of the origin concept. Here the concept noon is described by the color green, and the concept midday is described with the color orange. In each step, we denote the active concept with the color it has been activated in. We describe every activation step in one figure, to describe how this algorithm works.

The gray words in the two example definitions of “midday” and “noon” depicted in boxes in Figure 6.2 are stop words. Stop words are ignored in this example. Each concept in a definition gets all the markers passed to the definition. The concept “day” has a special purpose in this example: It shows what happens if the same concept is used in multiple definitions or is referenced multiple times by different relations. The concept “day” is first used in the concept “time of day” which is a holonym of “midday”. Secondly, “noon” is a meronym of a day and third and fourthly it is used in the definition of midday and noon: “The middle of the day.” and “Time of the day when the sun is in ints zenith”. This explains why the concept “day” has been activated twice: First, from the definition relationship to “midday” and second by the hypernym relation with “midday” as well.

Other special concepts are the concepts “is” and “when”: During the decomposition, the concepts are reduced to their lemma via lemmatization or stemming. This leads to the basic form of the word, without its inflections. If this basic form corresponds to a semantic prime or a synonym of a semantic prime as we have described them in Section 6.5.2, then this concept is marked as semantic prime and is not further decomposed. This is done with the concept “is” and “when” here. The lemmatization stems the concept “is” to the semantic prime BE and then concept “when” to the prime WHEN (time). This is denoted with a hexagon enclosing the prime colored in dark blue. Primes do not activate and consequently just collect markers over pulses. For simplicity, we do not include those markers in our example.

We start out by selecting a first spreading concept. We do so by selecting all active concepts and selecting one of them. Since there are two active concepts: “noon” and “midday” we select one of them (here we select midday at random). Then we calculate the out-function of midday by counting the outgoing edges which are seven. Then we distribute the start marking of 20 markers over those seven edges, which leaves us with ca. 2.86 markers for each edge. Depending on the edge function of each marker we reach the result depicted in Figure 7.3b.

In Figure 7.3b, the markers have been passed to all concepts connected to the concept “midday”. The concept “noon, ” e.g., is connected to midday by two relations (”also known as” and “synonym”) which leave us with orange 5.6 markers on the concept “noon” [5].

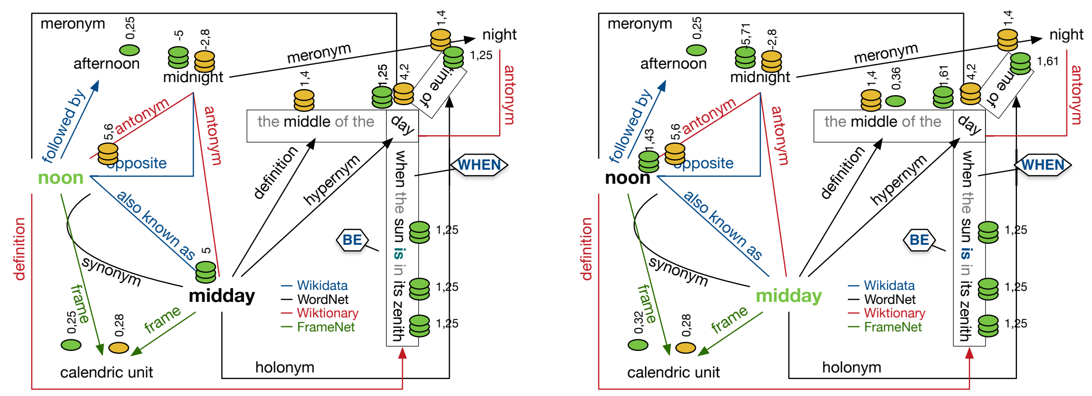
Here, all relations which have a name but are not specially listed in the Table 7.3.8 on Page 133 are subsumed in the category “Named Relation” and have an edge-function with a weight of 0.1. Therefore the concept “calendric unit” in Figure 7.3b becomes marked with orange markers and the activation level of $0.1 * 2.8 = 0.28$.

Now that we have dealt with the first spreading step taking us from Figure 7.3a to Figure 7.3b we can proceed to the next concept. This leaves us with one active concept: “noon”. Since we are still in the first pulse, we activate “noon” next with the green markers. Here again, we count the outgoing edges of the concept “noon” and divide the markers equally among those relations. This leaves us with 2.5 outgoing markers for each relationship. Depending on the edge-function this leaves us with the marked graph like depicted in Figure 7.4a.

[5]The size of the chips does not reflect its activation value in detail. Concepts with less than 1.0 markers of one color are depicted with one marker, to have an intuitive view on where the activation has gone in the graph.

Here again, concepts with multiple relationships are activated multiple times like the concept "midday". With Figure 7.4a we have reached the end of our first pulse. This means we have to reselect all active concepts. This time there are three active concepts: midday, noon and midnight. Since "noon" and "midday" share a non-directed relationship, the two concepts are activating each other. However, since we divide the markers up among all relationships, the activation becomes less and less. The relations between "noon" and "midday", make's up about $1/4$ of their relationships, we have roughly $1/4$ of markers left on these to the concept after the second pulse.

The concept "midnight" which is connected to both starting concepts via an antonym relation, is activated further negative. We have to remember, that the weights we have chosen for the edge-function, multiply the markers passed to it by the weight. To illustrate, if a positive marker is passed over an antonym edge, the edge-function negates its value.

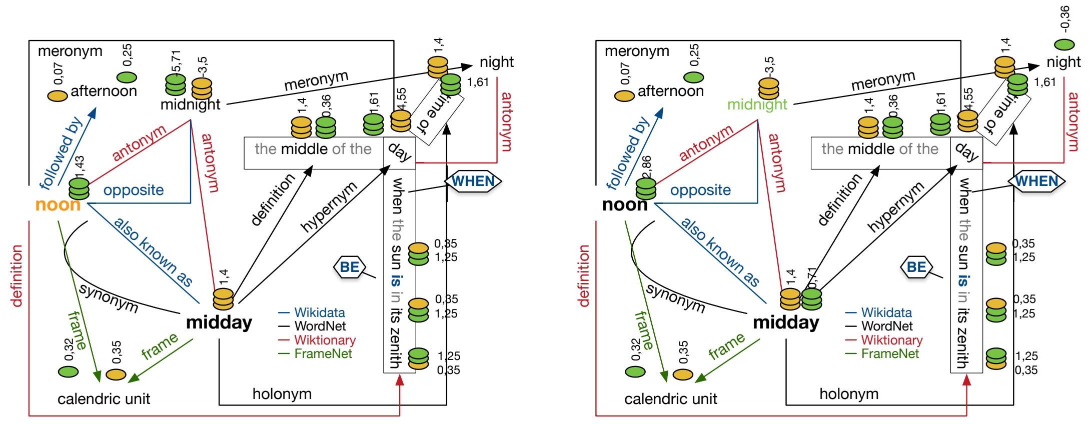


(a) Marked graph after second spreading step of noon. (b) Marked graph after third spreading step of midday.

Figure 7.4.: The example graph (based on the example graph shown in Figure 6.2) describing the second and third step of the Marker Passing example. The result of the third spreading step after the concept midday has been activated is shown in Figure 7.4b.

Now selecting from the passing concepts, we select "midday" to spread first. This leads to the marked graph depicted in Figure 7.4b. This step reduces the green activation of midnight even further to the activation value of -5.71. Here we can see, that the activation order of the markers make a difference, since the activation of midday" before "midnight" leaves us with more negative activation on "midnight" which, when activate next, well spread appropriately to the neighbors of "midnight". If the pulse size is increased, this effect can be opposed. Let us imagine we have selected the pulse size to be all active concepts; then all concepts activate their out-function before any concept activates their in-function. We can imagine that in a first step as if all concepts put their markers on the relations, and in a second step, all concepts take the markers meant for them from the relations.

Next, we activate "noon" since it has 5.6 markers of the color orange. This leads us to the marked graph depicted in Figure 7.5a. Here, only one concept is still over the markers threshold. Which leaves us no choice but to activate the concept "midnight". This is possible since the activation threshold can be reached in absolute values. Since the concept "midnight" passed negative markers, we can observe an interesting effect of negative weights: concepts semantically similar to midnight are activated negatively as well. Furthermore, concepts connected to the antonym might have passed negative markers to "midnight" , but if "midnight" passes markers, they receive positive markers back. This is because our antonym relation is not directed so markers can be passed to it in any direction and because the edge-function does multiply the weight on the markers. This leads to the effect that after the fourth activation the concept “midday” is activated positively again.



(a) Marked graph after third spreading step of noon.

(b) Marked graph after fourth spreading step of midday.

Figure 7.5.: The example graph (based on the example graph shown in Figure 6.2) describing the second and third step of the Marker Passing example. The result of the third spreading step after the concept midday has been activated is shown in Figure 7.4b.

The result of our fourth Marker Passing step can be seen in Figure 7.5b. With the edge-functions being all smaller or equal to one, the active concepts become less and less until we do not find any active concepts anymore. So that we can stop since our termination condition is reached.

Now it is time to interpret our result. The result of our example execution of the Marker Passing algorithm is shown in Figure 7.5b. Therefore our next objective is to see how we can use this marked graph to answer questions we want to ask. First, let’s look at the things we decomposed: First, we decomposed two words: “noon” and “midday”. Then we placed markers on those concepts and propagated them in a way that similar concepts get more markers than opposite ones. In fact, the question we have been asking for this example is: What are similar concepts to noon and midday?

Depending on the questions we ask, the interpretation of the resulting marked graph differs. For our questions, we wanted to know similar concepts to noon and midday. Then our interpretation of the result could be to select the concepts with a maximum total amount of all colors of markers as similar and the concept with a minimal amount of markers as not similar. This would lead to the result that concepts like the day, time, sun and zenith are similar concepts and that midnight and night are un-similar concepts. This interpretation is just one example of how the decomposition and the Marker Passing (and their selection of parameters) can work together to represent meaning. With the result that different decompositions, different marker-passing algorithms, and different interpretations can be found.

Since we are not the first ones to use Marker Passing on semantic graphs [21, 59, 286, 301, 314, 329, 333, 362, 421], we now look at the contribution of this work in more detail. Starting out by looking at the difference of our approach to others, we will look at the combination of connectionist models of meaning with symbolic ones. We do that by looking at the symbolic information encoded in the markers in Section 7.3.9 and continue by looking at the interpretation of concepts in Section 7.3.10 to finally extend this connections model through an edge interpretation in Section 7.3.11.

7.3.9. Markers Interpretation in Marker Passing

We have seen above that the definition of the information encoded on the markers is problem specific and influences further design decisions. Some examples of markers could be: boolean tokens which are passed over a graph like it is done in Petri Nets [306], a decimal activation like in Artificial Neural Networks [347] or the information passed on by a Marker Passing can become more complex like proposed by Minsky [273]. Depending on the use case, different information can be encoded on our marker. In our example in Section 7.3.8, we have used two types of symbolic information: the information of an amount of activation, and the origin as a start concept the markers were placed on. Depending on this information, the rest of our Marker Passing algorithm has been developed. This means that for example, the threshold of a concept uses both parts, the amount of activation collected by all markers received from one origin. The concept interpretation does not stop there: We also used the origin in the in- and out-function.

The main difference to the state-of-the-art here is that this information on the markers is flexible and if we can have a dynamic interpretation, this interpretation of the markers by the Marker Passing algorithm enables more expressiveness. We now take a look at which information about and from markers is available for the Marker Passing algorithm and how this information can be used to interpret the resulting marked graph.

We separate the information into four parts: First, the symbolic information encoded in the markers which enable us, e.g., to create specific markers of one or more sorts. Second, the meta information about markers, like their count. Thirdly, the information about the markers distribution over the semantic graph, which lets us analyze active areas. Moreover, fourth, the change in markers over the pulses or time. Each of that information about and from markers can be used independently on what the markers hold as a sort of symbolic information.

In the simple example, where markers only carry the boolean information if it is present or not (like in a Petri-Net), there are still all other three information types available regarding the markers. With more complex marker sorts the symbolic information becomes more complex and with that its meta information. Looking at the information available we can analyze the following four groups:

1. **Symbolic Information** The sort of the markers is the most complex and domain dependent part of the marker's information. Choosing which information to model in the marker, and how to interpret it depends on the questions asked with a query and the answer we want from the interpretation of the Marker Passing result. The symbolic information of the markers is the only information available at all time during the Marker Passing algorithm. Hence we can use it from the pre-adjustment phase to the check of the termination condition. In our example in Section 7.3.8 the origin (depicted as color) and the activation level (depicted as a decimal number above the markers) are the symbolic information encoded by a marker.

2. **Meta Information** Meta information about the markers and the graph include information like the total amount of start markers, the number of different types of markers, the total

number of edges or concepts in the graph, the pulse size and so on. Depending on this information, the Marker Passing algorithm can change its behavior. However, there are some restrictions rooted in the way we have defined the Marker Passing algorithm. Since we have defined the in-, out- and edge-function (see Section 7.2.2) in a way that their result depends only on their local information. This means the meta information available here depends on the current concept which is activated. Thus only some meta information is available during those functions. Let's call this information local meta information. Consequently, we need to define the difference in the meta information:

Global meta information is information about the graph or its marking which take into account the whole graph and its markers not encoded as symbolic information on the marker.

With global meta information, we can express things like the total amount of markers currently marking the graph, the number of concepts or edges in the graph or the number of the current pulse. Global meta information is available in the pre- and post- adjustment as well as the selection of active and spreading concepts and the termination condition.

Not all meta information needs to be global. A concept can collect meta information as well. The concept in its in-function could, for example, keep track of the markers types and react differently the first time it is activated with a new markers type. For this reason, we need to define local meta information as well:

Local meta information is information about the concept or its marking which take into account this concept and its markers not encoded as symbolic information on the marker.

The local meta information is available to in-, out- and edge-function and can be calculated for each of those functions and each concept and edge independently. An example of a global meta information which can not be calculated locally is the total amount of pulses passed. Since the local concept can only act in the pulses, it has been active. All meta information can be made available to all phases of the Marker Passing algorithm by modeling them as symbolic information on the markers. Depending on the information modeled, this can be done in the pre- and post-adjustment Marker Passing phases.

3. **Distribution** As markers pass through the graph, the markers form a distribution. This distribution can foster further information and can be used to influence the Marker Passing. Distributional information can be seen as a type of global meta information. It describes the markers about other meta information. Enabling us to define something like a hot-spot[6] or a focus point during the Marker Passing or to express relationships of markers to concept types. This information is available during the pre- and post-adjustment and the termination condition.

4. **Change** The information about markers' change is another meta information which we want to discuss separately. We define meta information change as:

   Meta information about change is global or local meta information about concepts and markers which change over time or pulses.

[6]A hot-spot can be seen as a subgraph which exceeds an activation limit.

We analyze this information separately from other meta information since it takes into account the change of meta information as well. This enables us to define something like velocity of markers. This lets us analyze information about the markers change which leads to further meta information like attention points where concepts are more often marked as active than others. Because this information needs to be collected pulse overarching this information can only be collected during the pre- and post-adjustment and the termination condition phases.

Under changing information, we can also include the change in markers' interpretation. Here the markers include information on how to react to other markers. An example application of this use of meta information about markers can be found in the work of Fahlman [92] where the interpretation of the markers is dependent on the query, and with that, if modeled in our Marker Passing paradigm, the interpreting information of a marker is passed on the markers itself.

One could imagine a Marker Passing where markers are added and removed dynamically. In this case, new marker types can be introduced during the Marker Passing. One use case for such a scenario is the distributed query[7] of one graph by multiple users, where the queries influence each other. This would enable a conversational behavior of the answers given by the algorithm which leaves us with an implementation of abstract markers interpretation[8], which is left to future research efforts.

We can conclude that the markers' interpretation takes place in multiple parts of the Marker Passing algorithm and that the information passed on with markers is specific to the problem. The problem inclines the question we ask, which is about to be answered by the interpretation of the marked graph as a result of the Marker Passing algorithm.

Where Fahlman [92] can encode the presence of a marker type for each node, the novelty in our extension is, that more information than the presence of a marker type at a node can be used for the Marker Passing. This includes but is not limited to the information discussed in this section.

Next we look at how the state-of-the-art marker-passing algorithms is extended by the introduction of concept (see Section 7.3.10) and edge types (see Section 7.3.11).

7.3.10. Concept Interpretation in Marker Passing

In difference to the approach of Fahlman [92] where the symbolic information on the markers is used to enable reasoning on an entire knowledge graph, we introduce concept types, where each concept can react differently to markers. In the approach of Fahlman, every concept reacts to the present markers in the same way. With that, different markers passing through the knowledge, a graph can cause different marked concepts to react differently to a new marker. We extend this approach with a concept type which has its own "interpretation" of its marking. We define interpretation as follows:

[7] A query can be distributed in parallel or over time. Having one graph queried by multiple queries enables us to model something like attention or focus to analyze which parts of the knowledge graph or the semantic graph are of interest, form correlations, topics or causal relations.

[8] The markers could need to be typed by a hierarchy, and this hierarchy needs interpretation, or the interpretation of the markers is encoded as symbolic information in the marker.

An concept interpretation is the effect of in- and out-function of a concept, given its current marking, the getActiveConcepts- and getPassingConcepts function and a set of markers.

The interpretation of a concept, therefore, depends on the implementation of the in- and out-function. The effect of the in- and out-function is again a marking on the graph but with the restriction of the local context of the concept under study.

To formalize the concept interpretation, we introduce concept types. A concept type is a mapping of the type graph discussed in Section 6.4 to the concepts of the knowledge graph.

Here we introduced the example concept types: Stop Words, Semantic Primes, Logical Concepts and Not. Additionally, we added the relevant properties of a concept type[9] in Figure 6.6 by giving a Concept it's in- and out-function and the active- and passing-ConceptFunction. These conclude the concept interpretation with respect to the definition above.

In our example of "midday" and "noon" from Section 7.3.8 on Page 133 a Semantic Primes concept has an in-function where all incoming markers are ignored. In particular a Semantic Prime concept is never activated and therefore never passes markers with its out-function.

Like the concept interpretation, we introduce an edge interpretation where different edges can react differently to markers. How this is done and what can be modeled with that is subject to the next section.

7.3.11. Edge Interpretation in Marker Passing

Derived forms of the Marker Passing like artificial neural networks, use weights for each edge to change the behavior of, e.g., the activation spreading over the network to adapt to a given training set [347]. Like with the concept interpretation we introduce an edge interpretation giving each edge type the possibility to define its behavior exceeding the setting of weights. In this section, we will analyze how this edge interpretation influences the Marker Passing.

A Edge interpretation is the effect of edge-function of an edge given an edge and its markers.

Like with the concept interpretation, the edge interpretation is given by the edge types. The edge type is defined in the type graph shown in Figure 6.6 on page 101. Here each edge is assigned to exactly one edge type.

Since the edges are defined in an inheritance hierarchy, like with concepts, the inherited edge-function can be overwritten in a specialization. An example of such relations is shown in Figure 6.6 on page 101 where we have a general relation which inherits the edge function to its children i.e. the semantic relation which can be further specialized like shown in Figure 6.7 on page 102.

The edge interpretation function allows us to modify the markers passed to the edge by an out function. It allows each edge to change the symbolic information passed by the markers or change its connectionist information by choosing to which target of the edge the markers are passed to. Here the restriction is that only markers passed to the realization can be modified, and the edge can only pass markers to concepts which are part of its target function codomain.

In our example Marker Passing in Section 7.3.8 the edge-function of, e.g., the antonym relations changes the symbolic information on the markers into a negative activation. Here we

[9]In our example of a semantic graph the concepts are made up of concepts, and the different concept types are reflected by different concepts.

selected all activation of all source concepts and then passed the modified information on to the target concepts.

As we have described in Section 6.4 an edge of our graph can be seen as a concept as well. Therefore an edge can act as an edge when it is activated by a concept, and it can act as a concept if it is activated as a concept (if markers are passed to it by an edge).

Also, the edge-function might choose to activate another edge which is part of its targets depending on the markers passed to it and the implementation of the edge function. In our example relation in Figure 6.2 the "give" relation inherits from a "transfer" relation. If the "give" relation is now activated, it can activate the transfer relation by sending markers over the is-a-relation to the transfer relation. The transfer relation is then handled like every other relation which has markers passed to it by an out-function in this spreading step.

As the activation of concept cycles, the information about a passed markers in the history of markers can help to detect loops in edge cycles. This means that for the Marker Passing algorithm cycles in concepts or edges do not have to influence the termination behavior.

The effect of edge interpretations can be seen in the Marker Passing example shown in Figures 7.3 to 7.5. In this example, one case of an edge interpretation is the antonym relation. Here the edge function negates the activation value of the marker, leading to the effect that antonyms of concepts with decreasing activation.

With this Marker Passing algorithm over the here defined graph structure, we can model many problems. We will focus our analysis on one kind of problem which veins this thesis: The modeling of meaning. This is done next, where the graph and the Marker Passing are specialized to the use case of modeling pragmatic meaning after a short conclusion of this section.

7.4. Conclusion

In this section, we have proposed an abstract Marker Passing algorithm, which uses a semantic graph to guide the Marker Passing. This extends the state-of-the-art of marker-passing algorithms by having defined four functions (In- Out- Edge- and After Send-Function) which allow a specialization of the algorithm to a given problem. Also, we have added the marker data type which allows us to encode symbolic information onto the markers and use this information in the spreading functions.

This Marker Passing algorithm together with the semantic decomposition present the two parts of our artificial meaning representation: The semantic decomposition automatically creates a semantic graph encoding connectionist information. The Marker Passing allows symbolic information to be encoded on the markers and the spreading functions to use this information regarding the structure of the semantic graph. Together they form two parts of a thought process like learning new knowledge (extending the graph) and thinking of this knowledge (Marker Passing).

Because of its general nature, the algorithm leaves many parameters to the developer for specification. Selecting the right information on the marker, the spreading functions and how the result is interpreted are just a few of them. This fosters broader applicability of the algorithm but generates more effort on using it in special applications.

Next, we will evaluate our approach in the different experiments described in Part IV.

8. Implementation

This section describes the implementation of the approach which was used to evaluate the hypothesis in the here performed experiments. For each component, we look at how it can be used and at the main classes used in the component. With that, we want to give an overview on the application interfaces and enable other scientists to reuse the results created here. The implementation has been realized using Java 1.8. The semantic graph is implemented using JGraph [14]. All dependencies have been managed using Apache Maven[1].

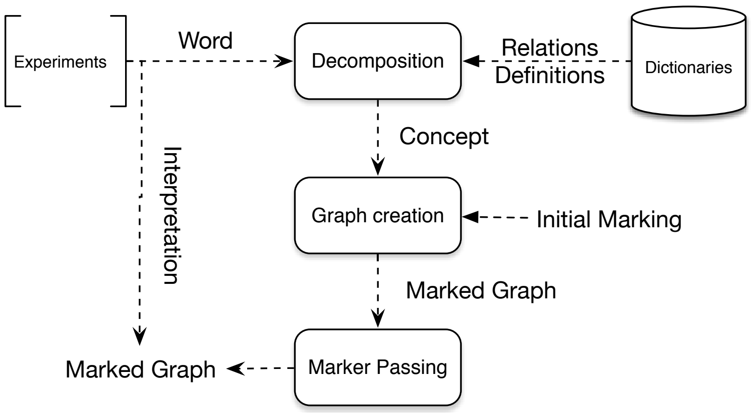


Figure 8.1.: Architecture of the implementation.

Figure 8.1 shows the architecture of our implementation. We start ours with an experiment, which needs to decompose some words. Those words are given to the decomposition. At the same time, the experiment defines how the result of the Marker Passing is interpreted. The words were given by the experiment then are decomposed by the decomposition component, using the attached dictionaries. The result of the decomposition is a graph containing all information gathered during the decomposition. This concept can be transformed into a graph, which then is passed to the Marker Passing. The result of the Marker Passing is a graph marked with the appropriate markers. This result then can be interpreted by the experiment.

We first look at the implementation of the Decomposition and how it can be used in Section 8.1. We then look at the Implementation of the Marker Passing algorithm in Section 8.2.

8.1. Semantic Decomposition

This section describes the implementation of the semantic decomposition. All components used in this thesis can be found online[2]. We will look at the API for the different components and explain how they are used. An Overview of the classes used by the decomposition can be found in Figure A.2 and Section A.

[1]see: https://maven.apache.org/
[2]To get access to the GIT repositories, please contact the author.

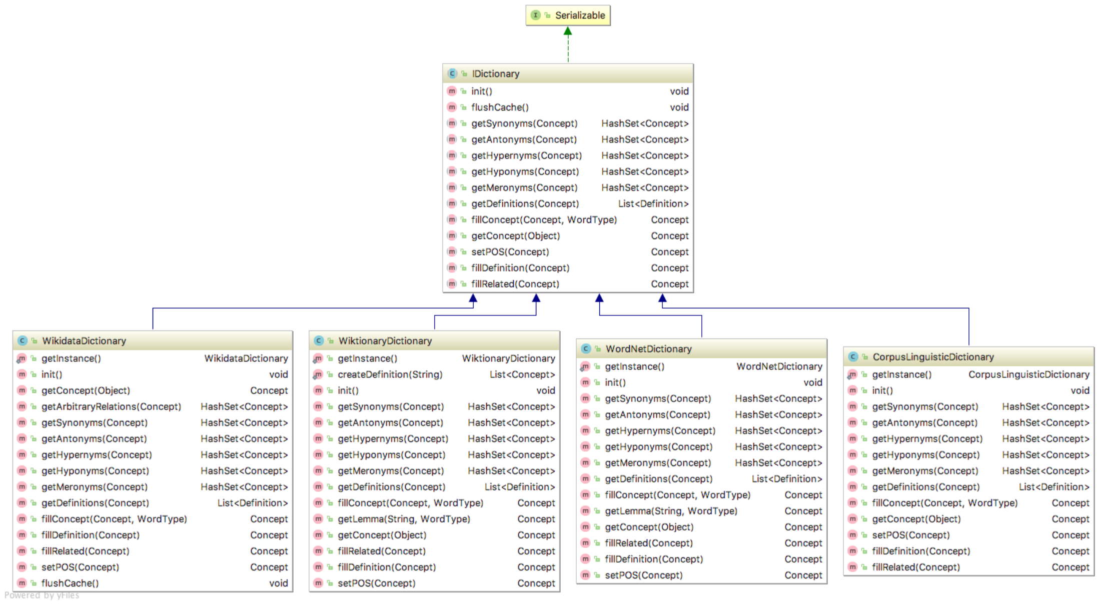

Serializable
IDictionary
init() void
flushCache() void
getSynonyms(Concept) HashSet<Concept>
getAntonyms(Concept) HashSet<Concept>
getHypernyms(Concept) HashSet<Concept>
getHyponyms(Concept) HashSet<Concept>
getMeronyms(Concept) HashSet<Concept>
getDefinitions(Concept) List<Definition>
fillConcept(Concept, WordType) Concept
getConcept(Object) Concept
setPOS(Concept) Concept
fillDefinition(Concept) Concept
fillRelated(Concept) Concept
WikidataDictionary
getInstance() WikidataDictionary
init() void
getConcept(Object) Concept
getArbitraryRelations(Concept) HashSet<Concept>
getSynonyms(Concept) HashSet<Concept>
getAntonyms(Concept) HashSet<Concept>
getHypernyms(Concept) HashSet<Concept>
getHyponyms(Concept) HashSet<Concept>
getMeronyms(Concept) HashSet<Concept>
getDefinitions(Concept) List<Definition>
fillConcept(Concept, WordType) Concept
fillDefinition(Concept) Concept
fillRelated(Concept) Concept
setPOS(Concept) Concept
flushCache() void
WiktionaryDictionary
getInstance() WiktionaryDictionary
createDefinition(String) List<Concept>
init() void
getSynonyms(Concept) HashSet<Concept>
getAntonyms(Concept) HashSet<Concept>
getHypernyms(Concept) HashSet<Concept>
getHyponyms(Concept) HashSet<Concept>
getMeronyms(Concept) HashSet<Concept>
getDefinitions(Concept) List<Definition>
fillConcept(Concept, WordType) Concept
getLemma(String, WordType) Concept
getConcept(Object) Concept
fillRelated(Concept) Concept
fillDefinition(Concept) Concept
setPOS(Concept) Concept
WordNetDictionary
getInstance() WordNetDictionary
init() void
getSynonyms(Concept) HashSet<Concept>
getAntonyms(Concept) HashSet<Concept>
getHypernyms(Concept) HashSet<Concept>
getHyponyms(Concept) HashSet<Concept>
getMeronyms(Concept) HashSet<Concept>
getDefinitions(Concept) List<Definition>
fillConcept(Concept, WordType) Concept
getLemma(String, WordType) Concept
getConcept(Object) Concept
fillRelated(Concept) Concept
fillDefinition(Concept) Concept
setPOS(Concept) Concept
CorpusLinguisticDictionary
getInstance() CorpusLinguisticDictionary
init() void
getSynonyms(Concept) HashSet<Concept>
getAntonyms(Concept) HashSet<Concept>
getHypernyms(Concept) HashSet<Concept>
getHyponyms(Concept) HashSet<Concept>
getMeronyms(Concept) HashSet<Concept>
getDefinitions(Concept) List<Definition>
fillConcept(Concept, WordType) Concept
getConcept(Object) Concept
setPOS(Concept) Concept
fillDefinition(Concept) Concept
fillRelated(Concept) Concept
Powered by yFiles

Figure 8.2.: Our IDictionary interface.

The semantic graph is enriched with all available information from the sources: *WordNet*[3], *Wiktionary*[4], *Wikidata*[5] and *BabelNET*[6]. Here the graph is created by a semantic decomposition[7] that breaks each concept down semantically until a set of semantic primes is reached [328, 98, 144, 407], the decomposition depth has been reached or no new information can be found in the information sources. To add another information source as dictionary to be used in the decomposition an Interface called IDictionary needs to be implemented like shown in Figure 8.2.

The dictionaries therefor all provide a function to get the concepts which are in the different semantic relation of a concept or its definitions. The main function used in the decomposition is the function *fillConcept(Concept, WordType)* which calls all the other function and adds their result to the given concept. Here the WordType restricts the dictionary to only look up definitions and relations for the given POS. The word type can be null, which lets the dictionary return the relations and definitions for all POS.

The decomposition component will download all necessary data sources and extract them into a ".decomposition" directory in the users home folder. Since this includes downloading and extracting e.g. Wikipedia, this might take several hours and will need disc space of more than 60 Gigabytes. This download is done in during the first initialization of the decomposition, e.g. during the first use of the decomposition.

Next, we will describe how to create a decomposition. Creating a decomposition is done like shown in Listing 8.1. The decomposition expects the word it has to decompose and the depth to which the concept should be decomposed.

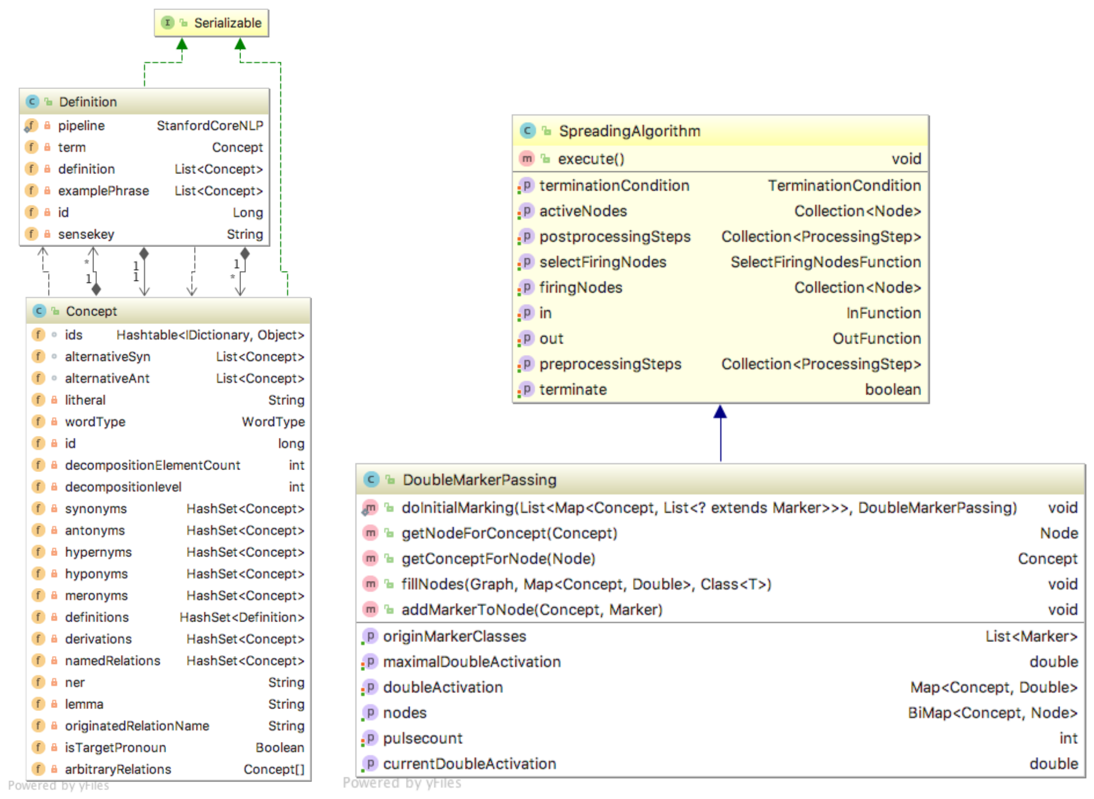
Listing 8.1: Example decomposition creation.

[3]https://wordnet.princeton.edu/
[4]https://en.wiktionary.org/
[5]https://www.wikidata.org
[6]http://babelnet.org/
[7]`https://gitlab.tubit.tu-berlin.de/johannes_faehndrich/semantic-decomposition`

```
Decomposition decomposition = new Decomposition();
Concept dog = decomposition.decompose("dog",2);
```

In the example shown in Listing 8.1 we decompose the word "dog" with depth two. A class diagram of the decomposition can be found in the appendix in Section A.2 on Page 260.

The decomposition returns a concept. A concept implemented as shown in Figure 8.3a incapsulates all relations the concept has with other concepts. This implements the signature described in Section 3.5. The concept holds the literal (the word) it represents, as well as its lemma (the base form of the word) and its named entity id (NER). The class additionally hold organizational fields like a list of IDs of the word in the different dictionaries, the highest decomposition level it has been decomposed with and the element count of the decomposition. The "namedRelation" is a relation extracted from Wikidata, where the relation itself is not a semantic one, but a named relation like "is-president-of-country".


Serializable
Definition
pipeline StanfordCoreNLP
term Concept
definition List<Concept>
examplePhrase List<Concept>
id Long
sensekey String
Concept
ids Hashtable<IDictionary, Object>
alternativeSyn List<Concept>
alternativeAnt List<Concept>
litheral String
wordType WordType
id long
decompositionElementCount int
decompositionlevel int
synonyms HashSet<Concept>
antonyms HashSet<Concept>
hypernyms HashSet<Concept>
hyponyms HashSet<Concept>
meronyms HashSet<Concept>
definitions HashSet<Definition>
derivations HashSet<Concept>
namedRelations HashSet<Concept>
ner String
lemma String
originatedRelationName String
isTargetPronoun Boolean
arbitraryRelations Concept[]
Powered by yFiles
SpreadingAlgorithm
execute() void
terminationCondition TerminationCondition
activeNodes Collection<Node>
postprocessingSteps Collection<ProcessingStep>
selectFiringNodes SelectFiringNodesFunction
firingNodes Collection<Node>
in InFunction
out OutFunction
preprocessingSteps Collection<ProcessingStep>
terminate boolean
DoubleMarkerPassing
doInitialMarking(List<Map<Concept, List<? extends Marker>>>, DoubleMarkerPassing) void
getNodeForConcept(Concept) Node
getConceptForNode(Node) Concept
fillNodes(Graph, Map<Concept, Double>, Class<T>) void
addMarkerToNode(Concept, Marker) void
originMarkerClasses List<Marker>
maximalDoubleActivation double
doubleActivation Map<Concept, Double>
nodes BiMap<Concept, Node>
pulsecount int
currentDoubleActivation double
Powered by yFiles

(a) Class diagram of a concept.

(b) Class diagram of a definition.

Figure 8.3.: Class diagrams of concept and definition.

In addition, a concept has a word type which encapsulates the Part-of-Speech (POS) of the concept. Here the word type includes the POS representations of the different data sources. The word type class proposes functions to convert one POS data structure into another.

The concept includes its definitions. A definition is implemented like shown in Figure 8.3a the concept, it defines is held here as "term" and a list of concepts making up the definition. The definitions implement the signature described in Section 3.5. A definition additionally includes a sense key which references the WordNet send key. This is needed if a definition should be selected in a Word Sense Disambiguation task.

8.2. Marker Passing

This section will describe the implementation of the Marker Passing component and how it can be used. The Marker Passing algorithm in its basic form is kept abstract in its implementation as spreading activation algorithm[8]. In this implementation the basic methods like execute() are implemented to run the Marker Passing. The interface defines which functions have to be implemented by a specific implementation of the Marker Passing algorithm, like its in- and out-function. This is shown as an example in Figure 8.3b. This algorithm is implemented as an example with the “DoubleMarkerPassing” specialization. This implementation is used as a basis for all our experiments and is listed in Figure 8.3b.

Figure 8.3b shows the abstract algorithm contains only the functions described in Section 7.2. A Marker Passing algorithm can be used in the way shown in Listing 8.2. In Figure 8.3b we create a concrete implementation of the abstract Marker Passing algorithm. The Marker Passing takes a graph as input, which represents the semantic graph on which the markers are to be passed on.

Listing 8.2: Example Marker Passing.

```
DoubleMarkerPassing doubleMarkerPassing =
new DoubleMarkerPassing(mergedGraph,
threshold, DoubleNodeWithMultipleThresholds.class);
DoubleMarkerPassing.doInitialMarking(startActivation,
doubleMarkerPassing);
doubleMarkerPassing.execute();
```

The second parameter of the Marker Passing represents the thresholds for the node type passed in the third argument. The node type has to be passed to the Marker Passing since different node types of the different experiment have different behavior. Here we created a node which cumulates double activation and has multiple thresholds for the markers encoding origin information.

To implement an instance of the Marker Passing, the (purple) properties shown in Figure 8.3b, beneath the line, have to be implemented. One implementation of these abstract Marker Passing functions is shown in the “DoubleMarkerPassing”. The implementation of the specific functions is explained for the example of a semantic similarity measure in Section 7.3.

[8] `gitlab.tubit.tu-berlin.de:johannes_faehndrich/general-spreading-activation.git`

Part IV.

Evaluation

9. Experiments with the Decomposition

This chapter will describe the first part of our evaluation of my approach: This first part will experiment with the parameters of the Decomposition (Section 9). Those parameters are fundamental for all other experiments including the decomposition. Since the decomposition is the basis for all other experiments, its parameters and their effect on the resulting semantic graph should be analyzed to be able to select the parameter later in the other experiments.

In this chapter we will first look at the parameters of the decomposition in Section 9.1 . We then analyze in more detail the selection of synonyms in Section 9.2 and the decomposition depth in Section 9.3.

9.1. Parameters of the Decomposition

In this section, we will describe our experiments with the parameters of the decomposition. Thes experiments are done before our semantic experiments because these parameters influence all of those experiments through the output of the decomposition: the semantic graph. Here we want to analyze how the parameters of the decomposition change resulting semantic graph. The parameters which are analyzed are the recursion of the semantic relations on the example of synonyms and which influence the decomposition depth has on the output graph. The analysis of all semantic relations could further gain insight into the creation of artificial meaning but is left to future work.

With this experiments, we want to foster more understanding of our decomposition. The results should enable us to improve its performance and find drawbacks which could be removed in future specializations. The parameters of the decomposition (e.g., synonyms and decomposition depth) are general properties and hence are not tied to a specific application.

The theoretic semantic decomposition introduced in Section 6 is not usable in practice because of two reasons: it takes a long time to create a total decomposition into semantic primes or even worse, it can not be done with the given information sources. The information source is unusable for the decomposition, e.g., if no definitions or explanations for concepts are given in them. In this case, the information source will only replace words with other words without describing their meaning. If words are only replaced by an agent, meaning can not be established which is argued in the philosophical ”Chinese Room” argument [353]. For the decomposition to become more usable we have restricted the theoretic decomposition with a decomposition depth shown in Algorithm 1. This experiment analysis the effect of the decomposition depth on the resulting semantic graph.

We start out by looking at the different relations used in the decomposition to identify the scope of the decomposition depth. Here for the termination of the decomposition we look for semantic primes or their synonyms because synonyms of primes are handled by the decomposition like primes themselves.

Finding synonyms depends on the used information sources. The information sources like

Wikipedia and WordNet describe different relations between concepts. Relations can be of syntactical nature like Part Of Speech (POS) relations or of semantic nature like the “is-part-of” relation of Synonymy. The selection of the relations used during decomposition makes up the semantic graph. Using this graph with the Marker Passing influences the result of the Marker Passing and the information it can encode. Therefore, the selection of the relations during decomposition and when to stop decomposing influences the result of the Marker Passing. As an example leaving out the Hypernym (generalization) and the Hyponym (specialization) relations has the consequence that generalizations like the similarity between “bird” and “cock” is harder to find and the markers take other routes between the two concepts. Bringing about that the same Marker Passing algorithm creates a different marker distribution on the graph.

Since we can configure the Marker Passing in a way to ignore relations[1] We will first add all relations to the decomposition and filter them out later if not needed. This is done since we might not know all application contexts the decomposition is used in. Consequently, we can not decide which relation will be needed and which one can be left out. One example for this argument is the Marker Passing done by Fahlman [92] where the edge “eats” connects “bird” and “worm” in the sense of birds eat worms. If this edge had not been added during the decomposition, the connection between worms and birds would have been lost, and the reasoning that birds eat worms but penguins eat fish would not be possible.

The remainder of this section discusses the use of the synonym relation in Section 9.2 and the decomposition depth in Section 9.3 since both influence the output of our decomposition. The synonymy relation analyzed here is a surrogate for other semantic relations, which need further analysis as well but are out of scope for this work. First, we analyze how synonyms are chosen, since the synonyms of primes are also seen as primes and as a result terminate the decomposition.

9.2. Selecting Synonyms

The first parameter of the decomposition algorithm we look at is the properties of relations in the example of the synonym relation. Selecting words which have an equivalent meaning is not precise enough for a semantic decomposition. The goal of this section is to determine if synonyms have certain properties with the information sources we use in the Decomposition. Most concepts have different meanings in different contexts, consequently, synonyms of a concept do too. A synonym might be equivalent in one context of use but not in another one, as discussed in Section 3.9. As a result, we use the Definition 23 to define a synonym.

Nevertheless, there is a set of agreed on synonyms which are held in different databases like the Roget’s Thesauri on which WordNet has build its synonym relations [192]. These synonyms can be used to bootstrap other synonyms from unstructured text like the definitions used in the semantic decomposition. Kennedy and Szpakowictz [192] suggest adding new word into the Roget’s Thesaurus by using word context and already known words and their synonyms. The major drawback of this approach is that the synonymy relation is not dependent on the context in which the words are used. Because we first have to create our semantic similarity measure to use Definition 23 as synonyms, we will start out by using the pre defined synonyms by the information source. In this section, we will discuss the properties of synonyms and analyze if

[1]This can be done, e.g., by setting the weight for a relation to zero in the out-function of the Marker Passing.

those properties are given by the synonyms we extract from our information sources.

We consider partial synonymy as defined by Löbner [240] since total synonymy is either trivial, i.e., abbreviations (e.g., AI is a synonym for Artificial Intelligence) or nonexistent if we consider if two concepts have the same meaning. If two concepts have the same meaning they are equivalent hence again a trivial case: each concept is a synonym to itself. To words, on the other hand, might be equivalent and not be synonyms as this is the case, e.g., for ambiguous words. In particular, if we talk about synonyms we talk about partial synonyms as defined in Definition 23 with a fixed context. This means we can look at the following properties to analyze if they are given by our data sources:

Reflexive: Synonyms are reflexive since a concept can be replaced with it self without changing the meaning of its word sense. A concept, in consequence, is a synonym to itself since the semantic similarity of a concept c and itself should be one. This is of cause only the case if the same word sense is selected.

Symmetry: Replacing one concept with another in a given context without changing its meaning should be symmetric if the context is fixed. Hence if two concepts c_1 and c_2 are synonyms in a given context, replacing the one with the other does not change the meaning of the sentence, neither does changing it back to the original concept as long as we select the appropriate word sense. We can see in Figure 9.1 and Figure 9.2 that both concepts (midday and noon) are synonyms since they are connected by a synonym edge, but a speciality of the information sources used for our decomposition is that midnight is an antonym of midday, but not of noon, even though they are synonyms. This means the information sources we used do not see synonyms as semantically equal because if they would, noon should have the same antonyms as midday.

Transitivity: Synonyms are transitive in a fixed context. Because of the symmetry of synonyms, if a concept c_1 is synonym to a concept c_2 in a given context and c_2 is synonym to c_3 in the same context then c_1 is synonym to c_3 as well. This can be proven by the Hypothetical syllogism [237].

Closure: Has the set of synonym closure under the synonym relations. This means: does the synonym relation always add new elements or is the set at some point complete and finite. With our information sources, the synonym sets are complete. This can be shown adding all synonyms of synonyms to a set of concepts. After running several tests, this recursive lookup of synonyms terminates and results in a finite set of concepts. E.g., the word “dog” (as a noun) as 193 recursive synonyms, the word “fly” (as a verb) has 62, and “nice” (as an adjective) has 150. There could be only two outcomes of this experiment: First, if the synonyms build a recursive chain and are not complete, then all words of the information sources become part of the synonym. Second, the recursion at one point does not add any new synonyms.

Having a measure of semantic similarity, we need to decide which recursion depth of a synonym can still be counted as a synonym. If our definition of a synonym is symmetric, a number of synonyms regarding one threshold of Definition 23 discussed in Section 3.9 should be finite. Never the less infinite chains of similarity could be created.

Concluding we can summarize: Until now synonym relations are given by ontologies like WordNet. These relations have been modeled by humans like in thesauri where for each concept there is a set of synonym concepts. If done well, the so defined synonym relation is symmetric. But with the generalization from context this representation is error-prone: As described in Section 3.9 two concepts might be synonyms in one context but not in another [176]. In reference to a financial institution, one would perhaps utter: "This counting house was robbed!" using "counting house" as a synonym for a bank. But in reference to a bench in the park or a slope of the earth, this would probably confuse the audience.

The goal of this evaluation is to show the usefulness of our representation of meaning by finding semantic similarity between concepts e.g. of a goal state and some concepts of a service; we analyze concepts, sentences, word senses and semantic service descriptions for their similarity. We are still using all of the relations available, but do not define their semantics. With that the analysis of Meronyms, Hypernyms und Hyponyms is left for others to be researched. The question if a synonym of a synonym is still a synonym of the original concept can be asked for all semantic relations. The restriction on when we stop following a semantic relation in the decomposition is discussed in the next section.

9.3. Selecting Decomposition Depth

The decomposition as described in Algorithm 1 is a recursion which ends the decomposition if a concept is a semantic prime or the decomposition depth is reached. The total decomposition of a concept includes all concepts found on the path to the prime. The resulting decomposition graph might become huge depending on the information sources available. Therefore we specify a parameter describing how often the recursive decomposition should be called. This parameter is called decomposition depth.

The restriction on the recursion depth allows us to narrow the recursion to exclude concepts which are too far from the concept which should be decomposed. Including more concepts leads to the effect similar to the "curse of dimensionality" described by Bellman [22], where with more dimensions valid data (for us the right concept) becomes sparse since more abstract concepts are included in the decomposition which adds more and more unwanted concepts. The inclusion of more abstract or further away concepts leads to additional concepts, which might not be part of the meaning of the original decomposed concept, hence since the decomposition should represent the meaning of a concept, adding noise to the decomposition.

To see the effect of the decomposition depth, we will now look at two examples shown in Table 9.1. Here "Nodes" is the number of nodes and "Edges" the number of edges in the graph. "Depth" is the parameter of the decomposition depth limit and "Diameter" is the diameter of the so constructed graph. The "avg Degree" is the average degree of incoming and outgoing edges per node and "avg Path Length" is the average path length between nodes.

As we can see in Table 9.1 the number of concepts grows fast with the increase of decomposition depth. The average degree of on the other hand grows only slowly. This lets us conclude that a number of leave nodes grow faster than a number of inner nodes.

Figure 9.1 shows the decomposition of the word noon in a graph representation with depth one. Here edges in orange are definitions, green are synonyms, red are antonyms, and blue are hyper- and hyponyms. Figure 9.2 shows an example of the decomposition of the word midday. We can see in Table 9.1 that for depth of one the degree is smaller than one since the avg degree

Table 9.1.: Example decomposition properties.

Concept	Nodes	Edges	Depth	Diameter	avg Degree	avg Path Length
noon	38	37	1	2	0.97	1
midday	39	50	1	2	1.28	1.34
noon	1105	1392	2	4	1.26	2,25
midday	1108	1412	2	4	1.2	2.29
noon	6701	13058	3	13	1.95	4.58
midday	6701	13071	3	13	1.95	4.58

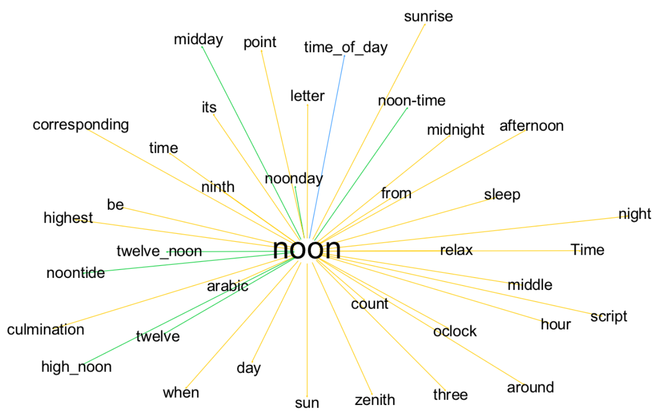


Figure 9.1.: Example graph of an example decomposition of “noon” with depth 1.

measured here is an average of in- and out-degree of edges per node. Since most of the nodes have only one incoming edge and no outgoing edges, the average drops under one. In our example of “noon”, we have one node with 37 outgoing and 37 nodes with one incoming node.

The size of the graph at depth two is too big to be displayed here. But we can show a part of the graph in Figure 9.3 and Figure 9.4. The nodes have grown by ca. 29 times and the edge by ca. 37 times. With the increase of depth, the growth of the nodes becomes smaller, but the interlinking becomes denser. From depth two to depth three the nodes grow only by a factor of ca. 6 and the edges by a factor of ca. 10. The observation can explain this that at some point the same concepts are used more often in multiple definitions, and the average path length is less than five. This explains as well why the two concepts have almost the same amount of nodes and

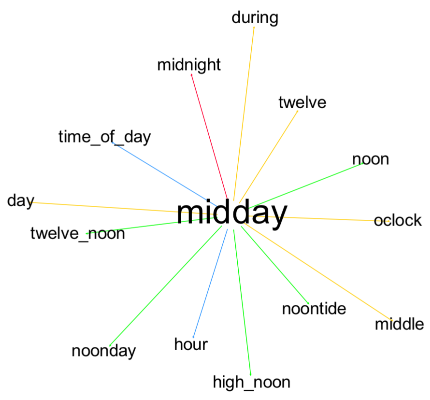


Figure 9.2.: Example graph of an example decomposition of “midday” with depth 1.

Figure 9.3.: Example graph of an example decomposition of “noon” with depth 2.

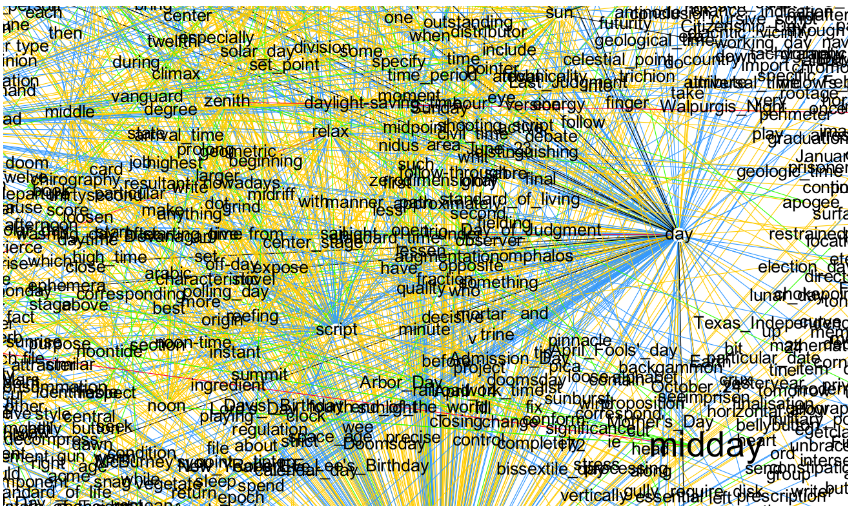


Figure 9.4.: Example graph of an example decomposition of “midday” with depth 2.

edges the higher the decomposition depth. Since the diameter is the longest of the shortest path between two nodes, it grows with the size of the graph and the branching factor. This shows that not all nodes are closely interconnected. The large diameter is created because the stop words have been removed. The diameter shows that sometimes we have to take long paths through the graph to get from one concept to another. The average path length, on the other hand, shows that most paths are still within an interval around the decomposition depth.

We were not able to produce decompositions, with no decomposition limit since the decomposition then ended in a resources error because our test system did not have enough memory.

In conclusion with this experiment, we have gained the knowledge on how fast and in which way a decomposition of a concept grows. This insight gives us a basis for the later selection of the parameter for specific problems and explains why in some problems, the decomposition depth can not get larger than three since the size of the resulting semantic graph becomes too big.

10. Experiments with the Marker Passing

This section will evaluate the here proposed representation of meaning in five experiments. Starting with a Semantic Similarity Measure in Section 10.1. Then we look at an experiment regarding Word Sense Disambiguation in Section 10.2. We then evaluate our approach with a Sentence Similarity Measure in Section 10.3. Then we apply our approach to Semantic Service Matchmaking in Section 10.4. Finally, we evaluate our approach in an experiment of creating a goal oriented semantic heuristic in Section 10.5.

The second part will consist of experiments using our approach in different research challenges (Section 10). Since the main goal is to create a formal representation of artificial meaning, we are facing the problem of proving that this representation constructs meaning in an agent. Even more complex is the question if the "correct" meaning is represented. We have seen that there is no correct meaning, only context-dependent meaning which differs from thinker to thinker. An example is the meaning of colors, which in a context do carry meaning, but there is no general meaning of, e.g., red. Therefore, we are not evaluating the correctness of meaning of words, but rather six example problems which determine if the artificial representation of meaning makes sense regarding human thinking. The first experiment is to learn about the parameters of the decomposition and how they can be selected in the following experiments. For the evaluation of our representation of meaning we carry out the other five experiments:

Word Similarity experiment: If words carry similar meaning and this meaning can not be measured, we can try to measure the distance of meaning represented by words as a less complex problem. Here we can try to find common ground on which is more similar, e.g., banana to apple or banana to anger. In this way, we do not measure the meaning of the concepts used, but only the effect this meaning has on certain problem-solving techniques. The idea is to take data sets where humans assess the semantic similarity of a list of concept pairs and then try to get a similar result by using our formalization of meaning. In our example of midday and noon, we might find that midday is closer to noon than to banana.

Word Sense Disambiguation: Another experiment to analyze the quality of a meaning representation is to assess if ambiguous concepts can be disambiguated in a context of use. Ambiguous words are words which can carry different meaning and therefore need to be disambiguated. A known example of the ambiguity of the word "rose" is: *Rose rose to put rose roes on her rows of roses*[1]. In contrast to our first experiment, the second one will establish if our artificial meaning can represent pragmatic (contextual) meaning.

Semantic Sentence Similarity: If words carry meaning, and they are used in syntactical structures like a sentence, the understanding of the meaning of a sentence is at least as

[1]Rose [a person] rose [stood] to put rose [pink-colored] roes [fish eggs as fertilizer] on her rows of roses [flower].

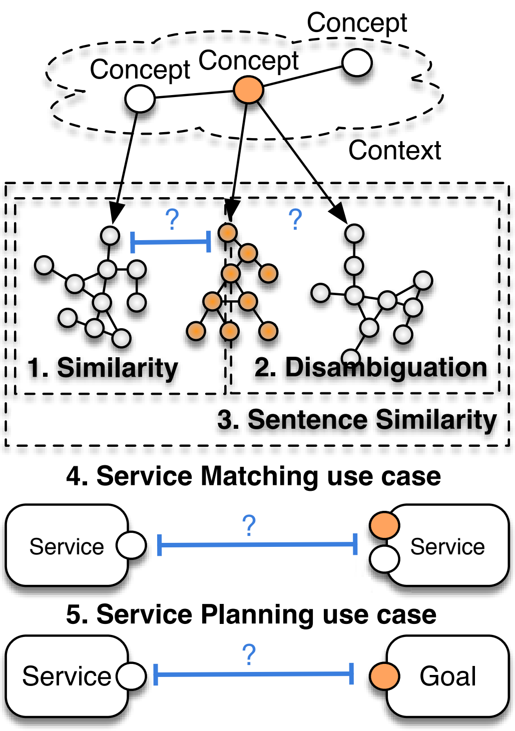


Figure 10.1.: Overview of the experiments making up the evaluation of our representation of meaning.

hard as the understanding of the meaning of the words making up this sentence. In addition, similar to word similarity the meaning of a sentence is hard to measure, leading us to our experiment of comparing the similarity of sentences. Calculating sentence similarity is a common task in AI research, and standard datasets exist. Our experiment will, therefore, evaluate our approach on such a data set to see if we can represent the meaning of sentences as well.

Semantic Service Match Making: The words used to describe the functionality in a service connect humans with machines. In a service description, concepts in ontologies, logical axioms and natural language sentences are combined to describe the functionality of a service. In service descriptions, the position of concepts influences their interpretation, e.g., the same concept in a precondition vs. an effect. Our first use case explores our approach applied to service description and its use to find a fitting service given a service request.

Heuristic for service planning: The assessment, if a service is helpful for a given goal includes the understanding of the goal, the descriptions of the services and then a reasoning on how well the described service can “help” to reach the goal. This includes finding similar concepts (with a semantic similarity measure) and to select the right word sense for the concepts describing the services or the goal. Further sentence similarity is used to compare axioms like ”IsBookedFor(Flight, Perter)” reformulated as sentence ”The flight is booked for Peter.”. Finally, parts of the service matching experiments are used to assess the services available, e.g., on their preconditions and effects. Consequently, this experiment is more an application of the results of the first four experiments to a real world problem: The general purpose service planning.

The evaluation section is structured as depicted in Figure 10.1. First, we describe the parameters of the decomposition. Second and third, we describe how a similarity measure can be created and third we describe the experiment of word sense disambiguation. Fourthly, we describe how we extended the word similarity measure to a sentence similarity measure. Finally, we dive into the applications of those results and use the insight gained from those experiments to improve our service matching and a semantic heuristic for service planning.

In more detail, the evaluation in our five experiments has each time a set of parameters which need specification, a data set, and an evaluation goal. Here parameters are selected with our use cases in mind. As the parameters have been discussed in Chapter 6 the evaluation keeps this structure, and we will elaborate the best parameters for each part of the approach through an experiment.

10.1. Experiment 1: Semantic Similarity Measure

In this section, we will build a semantic similarity measure based on our decomposition and Marker Passing algorithm. This has been published in Fähndrich et al. [104][2]. A semantic similarity measure returns how similar two concepts are regarding their meaning [336]. Part of this task is selecting the best word sense fitting the context first discussed by Waver in 1949 in his essay "Translation" in the Cambridge collection "Machine Translation of Languages: Fourteen Essays". We divide the problem from Waver into two experiments where the first part - a semantic similarity measure - is subject to this section and the second part - selection of word senses - is the experiment in Section 10.2.

To use our Marker Passing in this experiment, we need to choose the information modeled on the markers, the node, and edge interpretation to guide its markers and the termination condition as well as the final interpretation of the markers. In this experiment, we base the node and edge interpretation which models the symbolic influence of certain concepts on the idea of Charniak [48].

First, we define the signature of the semantic similarity measure we want to create:

$$\mathit{SemanticSimilarity} : \mathit{Concept} \times \mathit{Concept} \to \mathit{Double}$$

Here the semantic similarity measure is context independent because the input allows only two concepts, therefore, the entire measure is only based on the two concepts for which we want to calculate the semantic similarity. Next, we describe how we modeled the different parts of the Marker Passing algorithm, starting with the information encoded in the marker. With that we describe all the parameters specified in Table 10.2.

Figure 10.2 depicts our abstract approach, from the input of two concepts, which are decomposed using the approach described in Section 6. We call these two concepts for which we want to calculate a semantic similarity between the string concepts. They are the two concepts given to the algorithm as input parameters. The result is a semantic graph. The only parameter we selected from the decomposition is its depth. This influences the size of the graph, used for the Marker Passing.

[2]This section is based on the thesis of König [203] and Weber [402].

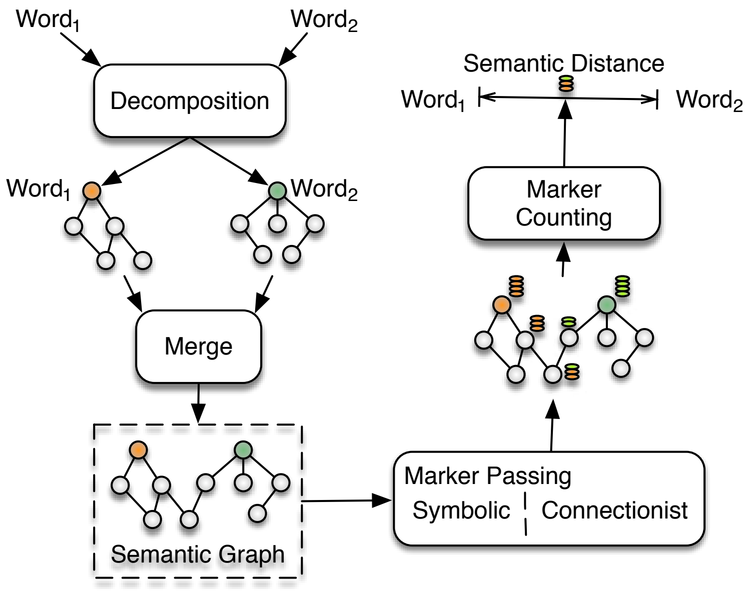


Figure 10.2.: Overview of the algorithm to measure semantic similarity of two concepts.

The state-of-the-art analysis in Section 10.1.1 about semantic similarity measures revealed that the most successful approaches can be clustered in three groups: Approaches using Thesauri approaches using ontologies like WordNet or Wikipedia or similar online dictionaries and corpus based approaches. The here proposed approach fits in the first two categories.

Algorithm 6 Semantic Similarity measure.

```
Name: Semantic Similarity Measure Input: Concept $c_1$, Concept $c_2$ Output: Double
1: Graph $g_1$ = DECOMPOSE($c_1$,Null)
2: Graph $g_2$ = DECOMPOSE($c_2$,Null)
3: Graph $g$ = MERGE($g_1$, $g_2$)
4: NodeData $ND_1$ = SETINITIALMARKERS($g$,$c_1$)
5: NodeData $ND_2$ = SETINITIALMARKERS($g$, $c_1$)
6: NodeData $init = ND_1 + ND_1$
7: NodeData $result$ =MARKERPASSING($init$)
8: return $d_{sim}(result, c_1, c_2)$
```

Algorithm 6 describes the overall calculation of the semantic similarity measure. Here line one and two decompose the given concepts into semantic graphs. The Merge-function in line three implements a standard merge of the two graphs. Here all nodes and edges of the two graphs are combined in a new graph. Then, in lines four to six the initial markers are set and merged into one NodeData which is given in line seven to the Marker Passing. The function for setting the initial Marking is specified in Algorithm 7 on Page 168. Line eight interprets the result of the Marker Passing and returns the semantic similarity. This interpretation is described in Section 10.1.13.

This section is organized in the following oder: We first look at the state-of-the-art in Section 10.1.1. Then we define the decomposition depth for this experiment in Section 10.1.2. Then we define the information modeled on the marker (see Section 10.1.3), how how we interpret a node (see Section 10.1.4) and how relationships are interpreted (see Section 10.1.5). Then, we describe how we selected the different parameters described in Table 10.2 in the Sections 10.1.6 and 10.1.7. The functions of our abstract Marker Passing signature defined in Section 7.2.2

will be specialized in Sections 10.1.8 to 10.1.12. Then, we use Marker Passing to identify relevant sub-graphs and describe the parameters use to interpret the marker information (Section 10.1.13). We then look at the test data sets available in Section 10.1.14. How the parameters of the Marker Passing are learned will be described in Section 10.1.15. The whole experiment is used to create a semantic similarity measure that is experimentally evaluated in Section 10.1.16. The results are then critically discussed in Section 10.1.17.

10.1.1. State-of-the-Art

This section will give a literature overview of existing semantic similarity or distance measures. This analysis is done to find the best approaches we can compare our approach against. There are many definitions of semantic similarity. In this section, we will choose the once we compare our results against, and we will take a deeper dive into some of their details to gain insight into how the work. The deeper analysis of the results is done to gain insight into the shortcomings and benefits of the different approaches, in contrast to a single value correlation value over the whole data set. This insight is needed later on to create a problem specific Marker Passing algorithm for a semantic similarity measure.

We do not agree with the simplification done by Ballatore [16] who reduces semantic similarity measures to hypernym and hyponym relations. This simplification reduces a semantic similarity measure to a taxonomic measure, contradicting the ontological principle of **identity of indiscernibles** which states that as if entities have all properties in common, they are the same entity. Without taking Meronyms (the part-of relation) into account, a taxonomical similarity measure based on “is-a” relations can not identify if two entities are similar with all their properties.

There are many surveys on semantic similarity or semantic distance measure like [72] who describes multiple semantic distance measures or [424] who survey on semantic relatedness. Others describe specific approaches using a certain information source like [130, 420] who describe WordNet based semantic similarity measure. We surveyed all of those approaches, categorized them to their basic approach and selected the best performing of each category. These three similarity measures are explained in more detail and are selected for the comparison against our approach.

Other works describe semantic distance measures like weighted randomized heuristic semantic walks over Wikipedia articles [123] or survey such approaches in a specific application like done in [155]. Work like those have been reviewed but are not further discussed in the section because their topic is too far of from our approach.

The research literature on word similarity metrics ranges from thesaurus based approaches (*cf.* [183]) to neuronal networks (*cf.* [269]). These approaches can be classified based on the used data structure. Most commonly the differentiation is made between *knowledge-based* and *corpus-based* approaches [268]. Zhang et al. [427] use the same classification in their survey, further subdividing the domain of knowledge-based approaches into *taxonomies* and *ontologies*. Both work on a graph representation of the domain knowledge but taxonomies are organized by the generalization-specialization relationship only, while ontologies are taxonomic structures enriched with other semantic relationships [342]. Taxonomical approaches can be specialized to the part-of (mereological view) relation as well. Based on this classification, we will look at three state-of-the-art approaches to understand how they work. The insight we gain by understanding

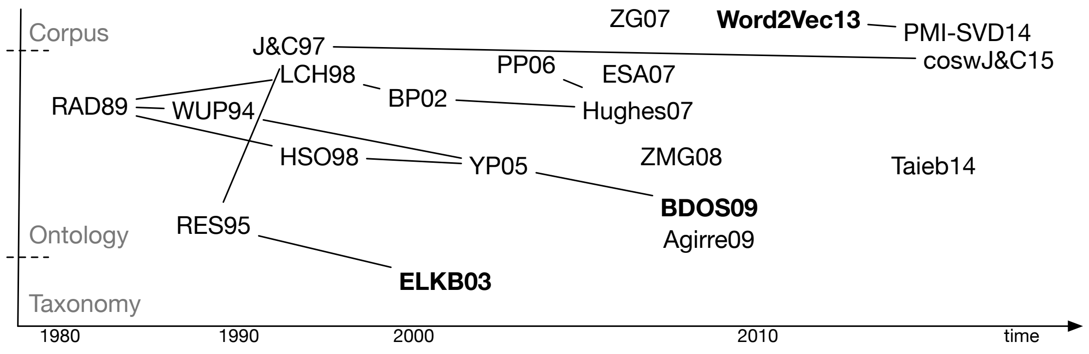


Figure 10.3.: Overview of the state-of-the-art of semantic similarity measures.

how other similarity measures work will help us to make the right design discussions in our approach later on. We selected the *Electronic Lexical Knowledge Base* [183] as a taxonomy-based approach because its use of the Roget's Thesaurus and its simple functioning principle, which shows how most of the taxonomical approach work.

Afterwards, we will shortly look at the *Bidirectional One-Step* approach [49] as a candidate of the ontology-based based approaches. Here not all types of relations are used. This approach has been selected because it shows the difference to a taxonomical approach but at the same time shows how the distance measure in a graph is almost the same.

Last we look at the *Word2Vec* [269] as a corpus-based approach as typical representatives of its group using neural networks (aka deep learning). *Lastra-Diaz et al.* [221] has not been selected as it mixes corpus-based and ontology-based approaches. The approach of Mikolov [269] has been selected as it provides better performance than the algorithm of Baroni [17]. Bringing the categories and approaches together, Figure 10.3 shows the development of the most prominent word similarity metrics on a time scale, starting with the work of Rada et al. [317] and ranging to the latest work, which was presented by Lastra-Diaz et al. [221].

Next, we will look at three different semantic similarity measures, one of each of the categories. This is done to explain the different principles of how the measures work and so that we can better comprehend the following design discussions. For further state-of-the-art on semantic similarity measures the interested reader is revered to the surveys of Pilehvar et al. [308], Lastra-Díaz, García-Serrano [221], and Zesch and Gurevych [424].

Taxonomy-based: Electronic Lexical Knowledge Base (ELKB)

The ELKB approach uses the digitized version of *Roget's Thesaurus* as a basis for its semantic distance measure. The structure of the thesaurus is similar to a taxonomy and represents words in a tree-like network, where most connections are parent-child relations. Here ELKB counts to the best performing algorithms on this kind of structure was introduced by Jarmasz et al. [183] in 2003 and named ***E**lectronic **L**exical **K**nowledge **B**ase*. The semantic similarity between two words is the length of the shortest path found between the two concepts. That is to say, if r_1 and r_2 are the sets of references for the words, then the similarity (sim_{ELKB}) between the words c_1 and c_2 is calculated as:

$$sim_{ELKB}(c_1, c_2) = 16 - min_{distance}(r_1, r_2), \tag{10.1}$$

Where 16 is the maximum distance in Roget's Thesaurus and with that defines the maximal distance of this approach. Here $min_{distance}$ is the minimal distance from one word r_1 to another word r_2 in the thesaurus, following the network in the Thesaurus. Later, the authors compared their approach to other state-of-the-art approaches like Hirst-St. Onge [168], Jiang-Conrath, Leacock-Chororow [223], Lin [233] and Resnik [324] to show that their approach yields better results.

If a word is not contained in the Thesaurus, the approach can not find a semantic similarity for it. This is why Kennedy and Szpakowicz [192] introduce an approach to integrate new words into the Roget's thesaurus with the help of Wikipedia. This would lead to a broader thesaurus but does not extend its height.

Ontology-based: Bidirectional One-Step (BDOS)

Another data base used to find semantic similarity between words is WordNet[3], a lexical data base developed at Princeton University. Similar to Roget's Thesaurus words in WordNet are connected through their relations to each other, creating a graph. Here the relations of concepts are semantic relations like "synonym" or "antonym" relations. Because of the graph representation of concepts and their relations, most approaches are similar to taxonomy based approaches: The distance of the words in the graph is proportional to their semantic similarity. The approaches differ in how they use the edges of the graph to calculate the distance between words.

One of these approaches is the ***Bid**irectional **O**ne-**S**tep* algorithm introduced by Chen et al. [49] in 2009. It uses the hypernym, hyponym and synonym relations and expands up and down the hierarchy simultaneously. The algorithm starts with two sets each containing one of the start words. Then in an iteration, each step adds adjacent nodes to the respective sets. After each iteration, the sets are compared and if an overlap exists the algorithm terminates. The overall step count is then the distance which defines the semantic similarity. Chen et al. compare their results to the results achieved byYang et al..

Corpus-based: Word2Vec

With an ever growing amount of digital information (aka corpora), statistical methods become more popular in linguistics. Thereupon a way to obtain knowledge about the human perception of semantic similarity refrains from the use of predefined networks but utilizes the vast amounts of language data produced by humans. Corpus-based approaches are based on the assumption that two words are similar if they are used in similar contexts [156].

Most of the corpus-based approaches create a vector representation or a bag of words for each concept which represents those words occurring in similar contexts. This vectors can be compared, to define a similarity among words.

We will look at the approach of Mikolov et al. [269] where a neuronal network is trained to learn the vector space representation of words occurring in the training set. Here different neural network architectures are proposed like probabilistic feedforward neural or Recurrent Neural networks. These vectors are trained to learn the co-occurrence probability of the words in the corpus. Word similarity is then found by taking a distance measure in the vector space embedding the vectors.

[3]http://wordnet.princeton.edu/wordnet/

Such vector approaches work well on tasks where the relatedness of words is well represented in the corpora. One such example set of tasks is the Sentence Completion Challenge [434] which searches for semantic relation by example. The tasks include examples like "France - Paris, where an equivalent relation would be "Berlin - Germany". Other relations are, e.g. "big - bigger" where "quick - quicker" would be an equivalent. The dataset contains other examples like "Japan - Sushi" and "Germany - Bratwurst" where now equivalent relations are searched without naming the relations.

Here Mikolov et al. [269] propose vector arithmetics to find such semantic similarities. Hence to answer the question: What relates to small like big to bigger? They formulate the calculation:

$$X = vector(bigger) - vector(big) + vector(small) \tag{10.2}$$

A semantic similarity measure can then be created, by using a norm to measure the distance between two vectors. Mikolov et al. use the cosine distance measure between the word vectors.

$$sim_{\cos}(c_1, c_2) = \frac{\vec{c_1}\vec{c_2}}{\|\vec{c_1}\|\|\vec{c_2}\|} \tag{10.3}$$

The cosine distance measure describes the cosine of the angel between the vectors.

Not all semantic distance measures can be categorized in only one of these three categories. One mixture of used information sources is so-called *Information based approaches*. Three of them are described next.

Information based similarity measures

Resnik [324] proposes that the semantic similarity of two concepts is the overlap or intersection of the information which is contained in the concept. In this measure, external information is used to estimate the similarity. In this context the probability $p(c)$ denoting the likelihood of a concept c occurring in a text. The entropy or information of a concept is defined as the negative log likelihood of the concept $-log(p(c))$. The measure of Resnik uses the inheritance hierarchy for example of WordNet to find the first common ancestors denoted in set $S(c_1, c_c)$. He postulates that the further up this common ancestor is found in the hierarchy the less similar two concepts are.

$$sim_{resnik}(c_1, c_c) = max_{c \in S(c_1, c_c)}[-\log(p(c))] \tag{10.4}$$

Practically the measure is elaborated by counting the occurrences of a concept in a corpus. To calculate this occurrence probability Resnik uses the BrownCorpus of American English. Every time an occurrence is encountered the counter of this concept, and all hypernym concepts are incremented. The likelihood $p(c)$ then can be calculated as follows: $p(c) = \frac{count(c)}{\sum\limits_{c \in C} count(c)}$ where C is the set of all concepts in the corpus. Subsequently if the first common ancestor of to notes is the root note the similarity is $-\log(\frac{\sum\limits_{c \in C} count(c)}{\sum\limits_{c \in C} count(c)}) = -\log(1) = 0$. Since the root note in WordNet subsumes every word in the corpus. This is interpreted as an similarity of 0 therefore no similarity at all.

Lin [232] introduces a similarity measure which uses external information as well. Lin defines the term similarity in an information theoretic sense, meaning that we need a probabilistic model of the entropy of a concept link with the similarity measure of Resnik [324]. Here the similarity measure is defined by using assumptions about similarity, rather the formulas, which "if the assumptions seems reasonable the similarity measure necessarily follows" [232]. Lin calls this a "Theoretical Justification" of a measure. The assumptions of Lin are

related The similarity between concept A and concept B is related to their commonality. The more commonality they share, the more similar they are.

opposite The similarity between concept A and concept B is related to the differences between them. The more differences they have, the less similar they are.

identity The maximum similarity between concept A and concept B is reached when A and B are identical, no matter how much commonality they share.

Lin then proposes a similarity measure by:

$$sim_{lin}(c_1, c_c) = \frac{\log(p(common(c_1, c_c)))}{\log(p(description(c_1)) + p(description(c_c)))} \tag{10.5}$$

Lin's theory of similarity is only applicable if the domain is described with a theoretic model. Furthermore, the similarity measure is defined with the notion of "commonality" stating what two concepts have in common, ergo needing a similarity measure for those commonalities. The function $description(c_1)$ describes the concepts which describe our concept c_1.

Ziegler et al. [429] describes an approach in which named entities are included in the taxonomy used for the calculation of a similarity measure. The measure builds upon the results of a query of different taxonomies like the google Directory and a manually created directory created out of the 'Open Directory Project' called DMOZ (http://www.dmoz.org/). A concept is queried, and the returned results are used to build a profile of the concept. The profile is formalized as a vector v which contains the ranked results of the query. Then the 'is-a' relation of the taxonomies used are traversed and used for the calculation of a similarity. Here the "symbolic" and "related" relationships are neglected so far. Spreading activation is used to activate related concepts to create a ranking for the profile of a concept. The resulting ranked vector $\vec{v}$ is used to calculate a similarity by using the Pearson's correlation coefficient in the following way:

$$d(v_x, v_y) = \frac{\sum_{k=0}^{|v|} (\vec{v}_{x,k} - \bar{v}_x) \cdot (\vec{v}_{y,k} - \bar{v}_y)}{\sqrt{\sum_{k=0}^{|v|} (\vec{v}_{x,k} - \bar{v}_x)^2 \cdot \sum_{k=0}^{|v|} (\vec{v}_{y,k} - \bar{v}_y)^2}} \tag{10.6}$$

Where $\mid v_x \mid$ is the length of the profile vector and $\bar{v}_x$ is the mean values of the profile vector of concept x. Using the ranking of search engines like wwww.google.de to calculate similarities makes the similarity measure dependable of this ranking. The foundation of the similarity measure is consequently a black box and can only be empirically evaluated. This leads to a similarity measure but not into more insight into how such a measure works.

We have only found one approach which uses activation spreading: Thiel and Berthold [379]. They use activation spreading like we described in Section 4.5. They use a double valued activation like in an ANN, to propagate through a semantic graph. This activation is then used to calculate the cosine similarity in the following way [379, Eq. (8)]:

$$d_{act}(u, v) = \cos(\hat{a}_i^*(v), \hat{a}_i^*(u)) = \frac{\langle \hat{a}_i^*(v), \hat{a}_i^*(u) \rangle_H}{\parallel \hat{a}_i^*(v) \parallel_{L^2} \parallel \hat{a}_i^*(u) \parallel_{L^2}} \tag{10.7}$$

With $(\Omega, A, a_i^t(v))$ being the separable L^2-space over the weights $v \in W$ with the created Hilbert-space $(H, \langle .,. \rangle_H)$.

Thiel and Berthold [379] then use the l_2 norm to represent the amount of change of one activation step towards the eigen vector and call this "velocity vector norm". Here the convergence speed is measured in:

$$\tau_t(v) = \| \delta_t(v) \|_{L^2} = \cap_\Omega \langle \delta_t(v), \delta_t(v) \rangle_H \, \mathrm{d}a_i^t(v) \tag{10.8}$$

This normalized activation then is used to calculate the cosine similarity or the Jaccard index [181], which then is used in a cosine similarity [379, eq. (9)]:

$$d_{sig}(u,v) = cos(\tau_t(u), \tau_t(v)) = \frac{\langle \tau_t(u), \tau_t(v) \rangle_H}{\| \tau_t(u) \| \; \| \tau_t(v) \|} \tag{10.9}$$

Which allows detecting if the activation of two concepts results in a similar activation of the activated nodes. Since Thiel and Berthold [379] use undirected graphs, this similarity is symmetric.

Consequently, two parts of the activation are analyzed: The activation vector of the activated concepts are similar (activation similarity), or the change of activation is similar (signature similarity).

Conclusion

In conclusion, we have analyzed the state-of-the-art of semantic similarity measures, categorized them and selected three approaches - one of each category - to learn how they calculate the similarity. Those three approaches have been explained in more detail to get insight into their functioning to be able to create our semantic similarity measure similarly. All these approaches are not context-dependent and, in consequence, cannot select a semantic similarity specific to a given context. Based on this insight, we now can configure our Marker Passing approach starting with Section 10.1.3.

10.1.2. Decomposition Depth

The decomposition depth has been evaluated in depth one and two. With the insight of Section 9.3 we are not able to decompose with hight decomposition depth, due to resource restrictions. Decomposing the RG65 test data set [336] yielding 48 graphs. Those 48 graphs have an element count like shown in Table 10.1.

Table 10.1.: Node and edge count for different decomposition depth.

Depth	Nodes	Edges
1	6195	6704
2	1288873	4515376

The growth of node and edge count restricts the decomposition depth to three due to computations resources available. To connect the so created graphs, we merged them into one graph by building the union of the set of nodes and edges.

10.1.3. Marker information

With the decision to test our approach with a semantic similarity measure, one type of design decision is to select the data type of a Marker. This means we have to think about which information we want to use at the end of the Marker Passing, to interpret the result as a semantic similarity measure. We select the symbolic information carried by the marker as two fold: First, we modeled a token activation level as a double value. Second, we defined a token origin which contains the concepts, from which the token has first started. The datatype of the marker *Marker* then is defined as tuple (Activation:Double, Origin:Concept)

These markers have been chosen because most of the state-of-the-art approaches use some distance of two concepts in an ontology or a thesaurus. Modeling activation, which can decay for every concept it passes, allows us to determine a sense of path length the marker has traveled. This is not to be confused with a simple step count done in other approaches like [129, 183, 374, 423]. This is because, the markers can be changed with the in-, out- and edge-function. This means the path length can be modeled as one of the information of the activation, but that's not all: The importance of the edge, the abstraction level, the degree of connectedness, and a number of nodes all can play their part in a number of markers passed from one node to another.

The second part of symbolic information "the origin" lets us track which marker has started from which initial concept. This lets us see which markers of the two starting concepts meet where and how many markers of which origin have passed over a specific concept. Furthermore, the origin of the maker gives us the information which concept this marker has started from. This gives us the possibility to activate multiple different concepts and finding out if their markers meet somewhere in the graph. With that, we can analyze how fast they meet or if some specific concept gets in contact with a marker of a specific origin.

Enabling the analysis of which markers have passed over which node, we have to model the node- and edge-interpretation with an implementation of the *NodeData* signature. The algebra implementing *NodeData* is defined in the Algebra *NodeData*.

Here the symbol $|$ denotes the list concatenation operation. The symbol $\varnothing$ denotes an empty list. Additionally, we did shortcut extending the NodeData signature by introducing new operation signatures directly in the algebra.

The *Data* describes the symbolic information modeled on a concept. Consequently, it is part of the node-interpretation. In addition, we model a history, which keeps all markers ever passed to the concept as a copy of the marker passed.

An object of type *Markers* is a function that for each origin specifies the activation of all markers for this origin. This encapsulates what we have called "Markers" is a function which allows a simpler description of the formalization of the algorithm below. Never the less, this is merely a shortcut to modeling the information on a marker data type. The *Markers* function can be applied to the current markers or the history. This is a convenience method which allows us to access the activation sorted by origin without having to iterate over all markers and origins.

The function *getMarkers*, *setMarkers*, *getHistory* and *setHistory* are the convenience functions to set the *Data*" of a node. Here for each function, we can get or set a *Markers* data type which includes the appropriate markers. For the *current* marking of a concept, this includes all markers currently marking this concept.

Algebra $NodeData =$ implements NodeData

sorts:

$$Data : \{(current, history) \mid current, history \in Markers\}$$
$$Markers : \{ofOrigin \mid ofOrigin : Concept \to Double^*\}$$
$$NodeData : \{ofConcept \mid ofConcept : Concept \to Data\}$$

opns:

$$getMarkers((current, history)) = current$$
$$setMarkers((current, history), newCurrent) = (newCurrent, history)$$
$$setNodeData(nd, concept, data)(x) = \begin{cases} data & \text{, if } x = concept \\ nd(concept) & \text{, else} \end{cases}$$
$$getNodeData(nd, concept) = nd(concept)$$
$$getHistory : Data \to Markers$$
$$getHistory((current, history)) = history$$
$$setHistory : Data \times Markers \to Data$$
$$setHistory((current, history), newHistory) = (current, newHistory)$$
$$\_ + \_ : Markers \times Markers \to Markers$$
$$\_ + \_ : NodeData \times NodeData \to NodeData$$
$$m_1 + m_2(origin) = m_1(origin) \oplus m_2(origin)$$
$$getSumFromOrigin : Markers \times Concept \to Double$$
$$getSumFromOrigin(markers, origin) = \sum_{x \in markers(origin)} x$$
$$multiply : Markers \times Double \to Markers$$
$$multiply(markers, x)(origin) = multiply_D(markers(origin), x)$$
$$multiply_D : Double^* \times Double \to Double^*$$
$$multiply_D(v_1 \mid rest, x) = v_1 * x \mid multiply_D(rest, x)$$
$$multiply_D(\varnothing, x) = \varnothing$$
$$emptyNodeData : \to NodeData$$
$$emptyNodeData(concept) = \{emptyMarkers, emptyMarkers\}$$
$$emptyMarkers : \to Markers$$
$$emptyMarkers(origin) = \varnothing$$

The functions *setNodeData* and *getNodeData* set the date of one concept in a NodeData or get the data for a special concept.

The addition operation $\_+\_$ on the data type *Markers* implements the addition of two markers by adding up the markers by origin. Therefore the addition operation adds *Markers* by concatenating the respective lists of *Markers*.

The addition operation $\_ + \_$ on the data type *NodeData* implements the addition of two NodeData by adding up the markers and *data* by equivalent concepts. Therefore the addition operation adds *NodeData* by concatenating the respective lists of *marker* and *data*.

The function *getSumFromOrigin* gets the sum of activation for one origin. Depending on which Markers it is applied, this function gets the current activation of a given origin, or the total activation by one origin if applied to the history.

The *multiply* lets us multiply a marker with a double scalar by multiplying each element of the list with the scalar. Here we use the helper function $multiply_D$ to extract the activation values of a marker and multiply it by the given scalar.

The *emptyNodeData* and *emptyMarkers* are constructors which initialize a new empty NodeData and a new and empty Marker.

This concludes the algebra describing the formal basis of our first experiment. Now we can describe the needed function of the Marker Passing algorithm formally.

10.1.4. Relationship Interpretation Function

The relationship- or edge-interpretation influences how markers are passed from one node to another. This influence is formalized with the weights shown in Table 10.2 on Page 172. Those weights are not chosen but learned in the experiment. This is done because it is difficult to predict the influence of the weights to the performance of the similarity measure. Therefore we can not describe the concrete values of the weights here. They will be selected by a learning mechanism in Section 10.1.16.

For the experiment of building a semantic similarity measure, the following relationships weights have been modeled:

Definition edge weight describes how markers are passed over definition relationships. Each concept in a definition of a concept has a relation to the definiendum. For this reason, all defining concepts are activated independently of which definition they belong to.

Synonym edge weight describe how relevant synonyms are to measure semantic similarity. Because of the transitivity of the synonym relation, there is the possibility of the two concepts we measure the semantic similarity for are connected by a synonym relation chain. Since synonyms are mostly not complete synonyms, but rather are only replaceable in some word meanings and context, the meaning might change with each synonym step. For more details see Section 3.9. This difference in meaning should be reflected through those synonym weights.

Antonym edges weights are similar to synonym edge weights since the antonym relationship is negatively proportional to the synonym relations. It is transitive as well, and it is changing the meaning of an antonym depending on context since there are rarely any true antonyms. See Section 3.9.

Hypernym edges weight reflect how the abstraction is used in the similarity measure. In most ontologies, there is one root concept, which if two concepts are not related at all, a connection can be found over this root node. In particular, if markers pass over to abstract concepts, the weight should reflect this abstraction.

Hyponym edges weight are similar to hypernym weights. Except that they are the opposite relation. This means they are reflecting the specialization in a semantic graph.

All weights are of type double and are multiplied with the activation level of each marker passed by them. We do not differentiate the origin of the markers. How the actual weights are selected, we will discover in Section 10.1.16.

10.1.5. Node Interpretation Function

The node interpretation influences how markers are handled by the nodes. Given a problem we have to decide which node types are relevant for this problem. In this experiment we have two types of node interpretations:

Concept nodes represent a concept and hold the markers passed to it. The threshold of a concept node is reached if one of the marker origin reaches the numeric activation threshold τ. Furthermore, the markers passed to concept nodes are added to the activation value of the last pulse. The markers are sorted after their origin. All the functions implemented here in the following sections will be implemented for concept nodes.

Prime nodes represent semantic primes from the NSM, which act as leave nodes and collect markers without passing them. They collect markers of the different origins, and like a concept, node sort them by origin and add new markers to the old once. This is reflected in Algorithm 9 where the concepts are only checked for a threshold if they do not belong to the set of NSM primes.

As a result, the concept markers can tell us from which origin they have been activated and how much activation in summation they carried. There is no information about time (in which pulse the markers arrived) nor about over which edge(s) the markers arrived. Since there are only nouns in the RG65 data set, we did not include Part-of-Speech nodes in our node interpretation. The edge types are used in the edge function, where each edge type has an own weight determining how markers are passed over it. How many markers are passed over the edge also depends on the amount of markers which are placed on the node. Starting with the initial marking given to the Marker Passing algorithm as start markers.

10.1.6. Start Marker

Because the semantic Decomposition produces an unmarked graph, we need to define initial markers. For the calculation of a semantic similarity measure, our algorithm has two start markers. The parameter called **start activation** defines how much activation is placed on those two markers. The origin of the initial markers is set to the concept they are placed on.

Algorithm 7 Setting initial Markers.

```
Name: InitialMarking
Input: Concept $c_1$
Output: NodeData
1: function SETINITIALMARKERS($g$,$c_1$)
2:     $ND$ = emptyNodeData
3:     $m_1 = \lambda(x).\begin{cases}\textbf{startActivation} & \text{, if } x = c_1 \wedge x \in g \\ \varnothing & \text{, else}\end{cases}$
4:     $ND$ = setMarking $(ND, c_1, m_1)$
5:     $ND$ = setHistory $(ND, c_1, m_1)$
6:     return $ND$
7: end function
```

Algorithm 7 creates the initial markers and places them on the given concept which is to be analyzed via our semantic similarity measure. Here the parameter **startActivation** is the amount of initial activation of these concepts. This parameter is one which is subject to our analysis and therefore can vary during our experiment. The value of the variable **startActivation** is selected by the learning mechanism discussed later in Section 10.1.16.

10.1.7. Activation Thresholds

The activation threshold describes when a node is activated. We select our activation threshold to be origin specific. Meaning that for every origin the markers of this origin can reach the threshold. Furthermore, the activation threshold for all concept nodes has been set to the same value. The NSM prime nodes can be seen to have a threshold of infinity since they never activate. This is the reason for which line two in Algorithm 9 neglects NSM primes.

Next we have to define which concepts are active. This is done in Algorithm 8.

Algorithm 8 Function to select the active concepts.

```
Name: GetActiveConcepts Input: NodeData ND Output: List<Concepts>
1: return $\{x \mid x \in Concept \land CalculateThreshold(ND, x) == true\}$
```

The particular value of the **threshold** is selected as a parameter, like all other parameters of this algorithm through a learning method we will discuss in Section 10.1.16. In Algorithm 8 we used a method called *CalculateThreshold* to evaluate if a certain concept has reached the threshold and with that is active. Algorithm 9 describes how this is done:

Algorithm 9 Threshold calculation of concept nodes for our semantic similarity measure.

```
Name: CalculateThreshold Input: NodeData ND, Concept c Output: Boolean t
1: Markers markerOfConcept = getMarkers(ND,c)
2: for all Concept $origin \in Concept \land c \notin NSMPrimes$ do
3:     if getSumFromOrigin(markersOfConcept, origin) $\geq$ theshold then
4:         return true
5:     end if
6: end for
7: return false
```

The Algorithm 9 describes how each concept node checks whether it is part of the active concepts or not. It consists of one checking if for one origin we have enough activation on markers to reach the threshold value given by the configuration parameter **threshold**. If this is the case, the node is marked as active. The function *getPassingConcepts* then is implemented trivially: All active concepts become passing concepts.

After being activated and selected as a passing node the node gives all its markers away to nodes connected to it. How markers are passed to other concepts is defined next in the out-function.

10.1.8. Out-function

The out-function specifies how the markers of a passing concept are passed to the neighboring concepts. We have looked at out-functions closer in Section 7.3.4. For our semantic similarity measure, we selected to pass markers out over all outgoing edges of a node. The activation of the markers is split up equally amongst all edges.

Algorithm 10 describes how the out-function has been implemented. Here in line 5, the markers are equally distributed over the outgoing edges of the concepts.

Algorithm 10 Out-Function of concept nodes.

```
Name: Out-Function
Input: NodeData $ND$, Concept $c$
Output: List<Relation × Edge × Markers>
1: List<Relation × Edge × Markers> $result$
2: Set<Edge> edges = $\{e \mid c \in targets(e)\}$
3: for all Edge $e \in$ edges do
4:     Relation r = source(e)
5:     Markers $m_{new}$ = multiply(getMarkers($ND$,$c$), $\frac{1}{|edges|}$)
6:     result.add(r,e,$m_{new}$)
7: end for
8: return $result$
```

10.1.9. Edge-Function

The edge function determines how markers are passed from one concept to another. For the semantic similarity measure, we have modeled a list of relation types described in Table 10.2. Those relations are weighted with a double value which is a parameter of this experiment. Each marker passed over a relation has its activation value multiplied by the relations weight. The different weights are part of the algorithm parameters like specified in Table 10.2. Algorithm 11 describes how to calculate the edge function in our case for a semantic similarity measure.

Algorithm 11 Edge-Function.

```
Name: EdgeFunction
Input: Marking $ND$, Relation $r$, List<Edge × Markers> $passedMarkers$
Output: Map<Concept, List<Edge, Markers>>
1: Map<Concept, List<Edge × Markers>> result = new Map<Concept, List<Edge × Markers>> ()
2: for all $(e, m) \in$ passedMarkers do
3:     Markers $m_{new}$ = multiply(m, getWeightOfRelation(r))
4:     for all Concept target $\in \{c \in Concept \mid \exists e_2 \in Edge : e_2 \neq e \wedge source(e_2) = r \wedge target(e_2) = c\}$ do
5:         result.get(target).add(e × $m_{new}$)
6:     end for
7: end for
8: return result
```

The *getWeightOfRelation* function is specified in Algorithm 12. Here the appropriate weight for the given relation is selected. In line 4 of Algorithm 11 this weight is multiplied with the markers passed over this relation. The weights are limited to be chosen from the interval $\{-1, 1\}$.

10.1.10. In-function

The in-function of a node collects the markers in the Marker Passing **step**, for which the current node is the target of. With the node interpretation of the **concept nodes** and the **prime nodes** incoming markers of all edges are sorted regarding their origin, and their activation is summed up in regards to the origin and added to the previous activation of this origin.

In Algorithm 13 the calculation of the in-function is defined in a recursive manner. Here the

Algorithm 12 GetWeightOfRelation.

Name: getWeightOfRelation
Input: Relation *r*
Output: Double

```
if r instanceOf Synonym Relation then
    return synonymRelationWeight
end if
if r instanceOf Antonym Relation then
    return antonymRelationWeight
end if
if r instanceOf Hypernym Relation then
    return hypernymRelationWeight
end if
if r instanceOf Hyponym Relation then
    return hyponymRelationWeight
end if
if r instanceOf Definition Relation then
    return definitionRelationWeight
end if
return 0.99
```

Algorithm 13 In-Function of concept nodes.

Name: In-Function
Input: NodeData *ND*, Concept *c*, List<Edge, Markers> *incoming*
Output: NodeData *ND*

```
function IN-FUNCTION(NodeData ND, Concept c, List<Edge, Markers> marker)
    if marker.isEmpty() then
        return ND
    end if
    SetHistory(ND, c,getHistory(ND, c) + getMarkers(ND, c))
    Marker currentMarkers = marker.first().getMarkers()
    marker = marker.removeFirst()
    newNodeData = setMarkers(ND, c, getMarkers(ND, c) + currentMarkers)
    return IN-FUNCTION( newNodeData, c, marker)
end function
return IN-FUNCTION( newNodeData, c, incoming)
```

Table 10.2.: The parameters of the Marker Passing approach with replacement in the in-function.

Parameter	RG-65		
	min	max	best
startActivation	0	1000	184.26
threshold	0	1	0.76
definitionRelationWeight	-1	1	-0.25
synonymRelationWeight	-1	1	0.53
antonymRealtionWeight	-1	0	-0.95
hypernymRelationWeight	-1	1	-0.86
hyponymRelationWeight	-1	1	0.40
terminationPulsCount	1	100	97
doubleActivationLimit	0	2000	21.61
decompositionDepth	1	3	3

set of markers given to the node is encoded in the parameter List<Edge, Markers> *incoming*. If multiple markers are passed over the same edge, then the list contains multiple entries for this edge with different markers. The incoming markers on the node are added to the history of the node. The history holds all markers which have ever been passed to the node. Then the markers are added to the markers of the node recursively.

Since there is an infinite number of in-functions, and not all of them can be discussed here, we selected two general classes for further analysis: The first class bases its marking on the previous marking of the node. The second class replaces the old markers. Next, we are looking at a side experiment on which of these two types of in-functions might be more of interest to use.

We argue why we have chosen an additive in-function instead of replacing the markers on a node: If markers are replaced, activation of one origin needs to reach the threshold, to activate the node in one activation step. This contradicts the idea of measuring some graph structure, since all information about the structure is lost, in the next pulse. To evaluate this decision we can run both experiments, which leads to the results that both models come to a similar result but with different configuration of the parameters. This experiment is conducted on the same data set used later on. We will go in more details on the data set in Section 10.1.14. The results with the replacement of markers in in-functions are like the following example: The best-found result is 0.762 Spearman's correlation with the configuration as shown in Table 10.2.

A result with additive in-function can for example yields better results like with the parameters shown in Table 10.3 we achieved a result of 0.882 in the Spearman's correlation. This result has been reached after 121 generation of the optimisation we will look at in Section 10.1.16.

Even if the performance of both Marker Passing algorithms can further be increased through adjustment of the parameters, we chose to continue with an additive in-function because of its better performance.

10.1.11. After-Send-Function

After a concept has been activation and passed its markers, the after-send function allows us to change the node data of the concept. This is done here by removing all current markers from the concepts.

After each pulse, the Marker Passing algorithm checks whether the termination condition has

Algorithm 14 After-Send-Function of concept nodes.

```
Name: AfterSend-Function
Input: NodeData $ND$, Concept $c$ Output: NodeData
1: SetMarkers($ND, c$,∅)
2: return $ND$
```

been reached. We look at how this is done for the semantic similarity measure, next.

10.1.12. Termination Condition

The termination condition of our algorithm is two fold: First, there is a limit to the pulse count. We introduce a global variable which is call **currentPulse** which counts the pulses is increased with each executed pulse. This restricts how many pulses can be created, before the algorithm terminates[4]. A pulse count limit allows us to stop the algorithm from passing markers from one concept to another even thou there could be active concepts. This limit is given by the parameter **terminationPulsCount**. We can imagine a graph of decomposition depth of one, where there are still connections between the two decomposed concepts. In such a graph the markers are passed from one side to the other and back. To eliminate such a back and forth the second part of the termination prevents such behavior.

Second, there is a **double activation limit**. Here we look at the total amount of concepts, which have activation of multiple origins. Since we are putting start markers only on two concepts, we named this condition "double activation limit". This part of the termination condition helps the algorithm to track how much the two subgraphs have to be activated by the opposite concept before the algorithm terminates. Depending on the choice of the relations weights, the overall activations of the graph can increase, degrees or rest the same. Since we chose our weights from $w_i \in \{-1, 1\}$, this means the activation decreases if not all weights are 1.0. Further, the markers which are adapted through the Out-, Edge-, and In-function are removed by the after sent, which makes the overall marker activation on increasing. If this parameter is set to a number near zero, the Marker Passing will terminate with the first contact of markers of the two origins. If the parameter is selected to be close to the activation of the start markers, almost all markers have to be passed to double activated nodes before termination. The termination condition is implemented as shown in Algorithm 15.

Here the currentPulse counts the Marker Passing pulses of the Marker Passing algorithm. For simplicity, the counting and initialization of the counting variable are neglected in Algorithm 15. If the double activation limit is set higher than the amount of activation in the start marker, the first termination condition terminates the algorithm. This lets the algorithm terminate even if multiple origins have activated no concept.

10.1.13. Interpretation

The interpretation is how we use the result of the Marker Passing to determine a distance between the two concepts. For our semantic distance measure, we want to collect all those nodes which have been activated by multiple origins. Since we have only two origins, this means we want to get all activation of the nodes, ever passed to the node, if the node has been activated by both

[4]This should happen before the markers are equally distributed over all nodes [26]

Algorithm 15 The termination condition for the semantic similarity measure.

Name: TerminationCondition
Input: NodeData *ND*
Output: Boolean *result*

```
1: if currentPulse $\geq$ terminationPulsCount then
2:     return true
3: end if
4: Double doubleActivationSum = 0
5: for all Concept $c \in$ Concepts do
6:     Double activation = 0
7:     Integer numberOfOrigins = 0
8:     for all Concept $origin \in$ Concepts do
9:         if getMarkers($ND$,c)(origin) $\neq \varnothing$ then
10:            activation += getSumFromOrigin(getMarkers($ND$,c), origin)
11:            numberOfOrigins++
12:        end if
13:    end for
14:    if numberOfOrigins $\geq 2$ then
15:        doubleActivationSum += activation
16:    end if
17: end for
18: return doubleActivation $\geq$ doubleActivationLimit
```

starting concepts. In consequence, we want to calculate the following overall activation of a node given NodeData *ND*:

$$\hat{a}^*(v) = \sum_{\forall o \in V} getSumFromOrigin(getHistory(ND, v), o) \tag{10.10}$$

where $\hat{a}_t^*(v)$ represents the sum of activation of node v ever passed to it.

This activation can then be used to calculate the semantic similarity between concept o_1 and o_2 given the NodeDate *ND* in the following way:

$$d_{sim}(ND, o_1, o_2) = \frac{\sum_{c \in V} \Phi(ND, (o_1, o_2, c))}{2 * \textbf{startActivation}} \tag{10.11}$$

where,

$$\Phi(ND, (o_1, o_2, c)) =$$

$$\begin{cases} \hat{a}^*(c) & \text{, if } getHistory(ND, c)(o_1) \neq \varnothing \wedge getHistory(ND, c)(o_2) \neq \varnothing \\ 0 & \text{, else} \end{cases} \tag{10.12}$$

$\Phi(ND, (o_1, o_2, x))$ filters out of all nodes which have not been activated by at least two of the start activations. This leaves us with the set of nodes having markers of both origins in their history set. In this way, if we activate two concepts (in our example “noon” and “midday”) at the beginning, this set contains all nodes which have been activated by markers of both concepts. This is normalized by the amount of start activation to obtain the semantic distance. In some theoretical cases, this value could become bigger than 1 or negative. This could, e.g., be the case when the scaling factor of two times the start activation is not fitting. This happens, e.g., when the overall activation over the entire history of the node is negative. For these cases, we then normalize the result so that it is scaled between 0 and 1. This normalization is done by calculating $value' = \frac{(value - min)}{(max - min)}$, where *max* is the maximal and *min* is the minimal similarity which occurred. Here *value′* is the normalized value of the input *value*.

Now that we have a formal description of our Marker Passing algorithm, we need to evaluate its performance. To do so, we need to select an appropriate data set. This is done in the next section.

10.1.14. Data Sets

This section will introduce the different evaluation data sets for semantic similarity measurements and select one of them for our experiment.

The task of finding semantic similarity reaches back to 1965 when Rubenstein and Goodenough first created a small list of noun pairs and conducted a study with humans filling in the semantic similarity for each pair. Since then four more well-known data sets have been developed using the English language. There is a multitude of data sets with special purpose vocabulary or with special tasks for the similarity measure. We will have a look at the most widely used and compare them for our purpose.

RG65 [336] the data set of Rubenstein and Goodenough is a collection of 65 noun pairs which have been ranked by 30 humans for their similarity. The similarity values here reach from 0 (less similar) to 4 (more similar).

WordSim353 [117] the data set of Finkenstein et al. [117] proposes 353 noun pairs which have been rated by 16 subjects with the task to estimate their relatedness. Here a score from zero to ten has been used, where zero means unrelated and ten means highly related.

Mtruk [318] the data set of Radinsky et al. [318] is essentially word pairs which are taken for their occurrence in the New York Times, and they have been rated from 23 Amazon Mechanical Turk workers for their relatedness. Each worker had word pairs of the WordSim353 data set as control pairs, to reject bad workers. The word pairs have been chosen so that the entire spectrum of co-occurrence in the New York Times is part of the data set. Rare words (occurrence with less than 1000 time in the looked at time frame) have been removed. This gives us a data set of 287-word pairs which have been rated from 1 (less relatedness) to 5 (more relatedness).

MEN [39] is a data set, which was constructed to evaluate multimodal models. Here the word pairs of the standard data sets like WordSim353 are selected, where the words appear as well as labels in the ESP-Game[5] and MIRFLICKR-1M[6] data set. This has been done to integrate graphical information into the finding of similar concepts. The selection of data sources seems questionable here since Wackypedia[7] includes information like “Fat Hippos are hippos that are fat. They are often mistaken for galaxies. And the moon. But they are Hippos...” With quote “Damn, those hippos are fat.” allegedly from Oscar Wilde.

[5]`http://www.cs.cmu.edu/~biglou/resources/` last visited on 20.10.2016
[6]`http://press.liacs.nl/mirflickr/` last visited on 21.10.2016
[7]`http://wackypedia.wikia.com/wiki/Main_Page` last visited on 21.10.2016

This might be a good linguistic resource for contemporary humor and modern slang, but for embedding words in a distributed meaning, this seems not fitting.

Stanford Rare Word Similarity Dataset [244] Luong et al. [244] composed a data set of rare words, where words with low frequencies but still having at least one synset in WordNet [270]. Consequently, these are words, which are rare but are no proper names or something similar. The word pairs are constructing by selecting words of WordNet with a step size of at least one and maximally two apart from the first word. In this way for each rare word, two pairs are created adding up to 3145-word pairs. Luong et al. [244] used Amazon's Mechanical Turk to get a rating of ten humans for each word pair. As a result of quality measures, this leaves us with 2034 ranked word pairs.

We have selected the RG65 [336] the data set of Rubenstein and Goodenough to compare our approach with the state-of-the-art since all other data sets are special purpose datasets, which are mostly used by the authors of the data set but not many other approaches. Therefore, the only way of comparing our result to the best available semantic similarity measures was to use the RG65 [336] data set without having to reimplement all other approaches.

The use of the RG65 [336] data set seems insufficient since only nouns are compared. More complex datasets exist, which could be used in the future for comparison. Using part-of-speech independent data sets will worsen thesauri approaches like ELKB since mostly nouns are formalized in thesauri. For future work, creating more complex data sets with different POS and different similarity types would be a benefit for the community. But this would mean that existing approach should be reevaluated for the new data set, which sometimes is not possible because they are not accessible.

10.1.15. Parameter learning

In this section, we will look at how the parameters of our approach are learned. For the evaluation result, we learned the set of parameters with a genetic algorithm. Here the parameters are interpreted as the DNA of our individuals. Then we use N-Fold-Cross-Validation to separate the data set into training and test data. Each individual is evaluated regarding its performance on the data set via its Spearman's correlation. The best individual is kept alive and mutated to generate newer generations. The mutation is done by modifying its DNA within the limits of our configuration shown, e.g., in Table 10.3. Here each parameter has a mutation probability of 0.5. The population size was held at ten, and if no increase in performers has been measured within 1000 generations, the DNA was reset to random values.

The parameters selected in Table 10.3 depend on the design decisions which we have looked at in the last few sections. Because of the vast amount of possible combinations of the parameters the search space is too big to search. Because of the size and its only partially predictable landscape, many combinations of parameters can yield the same or better results, and we can not claim that our parameters produce the optimal result.

Since the parameters are learned, we can only interpret why they work well. Some of the parameter selected depend on others like the double activation limit on the start activation. Some of the parameters may be randomly chosen and might have done almost none effect on the result. E.g. Since the double activation limit in the best parameter set we learn in Table 10.3 is almost

Table 10.3.: Parameters of the Marker Passing approach used for the task of semantic similarity measurement.

Parameter	RG-65		
	min	max	best
startActivation	0	1000	33.81
threshold	0	1	0.21
definitionRelationWeight	-1	1	-0.78
synonymRelationWeight	-1	1	-0.62
antonymRelationWeight	-1	0	-0.90
hypernymRelationWeight	-1	1	0.02
hyponymRelationWeight	-1	1	0.79
terminationPulseCount	1	100	80
doubleActivationLimit	0	2000	1999.99
decompositionDepth	1	3	3

2000, but the start activation is only ca. 34, and additionally half of the edge weights are negative, and all of them are less than one, it is unlikely that the double activation limit was ever triggered.

Some of the parameters are unintuitive. That a synonym weight is negative seems far from what humans do if they assess similarity. But since the antonym weight is more negative than negative synonym and definition relations can be explained by them make similar concepts still more similar than an antonym relations, having the effect that the semantic distance measure still works.

Also curious in this parameter configuration is that hyponyms seem to play a more important role than hypernyms. This could be interpreted, that example of abstract classes help the assessment of similarity more.

10.1.16. Evaluation Results

This section describes the evaluation results and compares it with the state-of-the-art. The experiment has been implemented in Java using the WordNet 3.1 and the MIT Java Wordnet Interface (JWI)[8]. For the Wiktionary implementation the Java-based Wiktionary Library (JWKTL)[9] has been used with a Wiktionary dump[10]. With the above-introduced parameter selection, we were able to reach the results shown in Table 10.4. The other results are taken from [308, p. 116, Table 9] and from [221, p. 148, Table 4] and present the state-of-the-art in this experiment. We sort the approaches in Table 10.4 by their Spearman's correlation because they are given by all approaches in the related work.

The Spearman's ranked correlation coefficient ρ is an overall ranking measure. Together with the Pearson's r Table 10.4 shows the performance of the state-of-the-art semantic similarity measures in comparison to each other. The Pearson's and Spearman's correlation are used to seeing how well the approach correlates with the human evaluation of semantic similarity and the Spearman's correlation how well the algorithm can bring the pairs of similarity in the right order. The Spearman's correlation is more abstract because the actual value of the semantic

[8]`http://projects.csail.mit.edu/jwi/` last visited on 09.09.2017
[9]`https://www.ukp.tu-darmstadt.de/software/jwktl/` last visited on 09.09.2017
[10]`https://dumps.wikimedia.org/` downloaded on 2015.12.19

Table 10.4.: Comparison of Spearman's ρ and Pearson's r correlation coefficients of different approaches with our approach. Anything but our results are taken from [308, p. 116, Table 9] and from [221, p. 148, Table 4].

Approach	RG-65	
	ρ	r
MP (this approach)	0.882	0.83
coswJ&C [221]	0.876	–
Ontology based [342]	0.86	–
Word2vec [269]	0.84	0.83
Hughes and Ramage [177]	0.84	–
ZMG-08 [425]	0.84	–
Lin [232]	0.834	–
Agirre et al. [3]	0.83	–
ZG-07 [423]	0.82	0.49
BDOS [49]	0.81	–
Taieb [374]	0.80	0.80
Rad89 [317]	0.79	0.79
HSO [168]	0.79	0.73
LCH [223]	0.79	0.84
WUP [418]	0.78	0.80
ESA [129]	0.75	0.49
Res95 [324]	0.74	0.81
PMI-SVD [17]	0.74	0.74
ELKB [183]	0.65	–
PP06 [302]	0.62	0.58

similarity is neglected as long a the right rank for the concept pair is found.

Table 10.4 shows that our approach can beat the state-of-the-art in respect to the Spearman's correlation. Concerning the Pearson'S correlation, our approach still reaches rank two in comparison with the state-of-the-art.

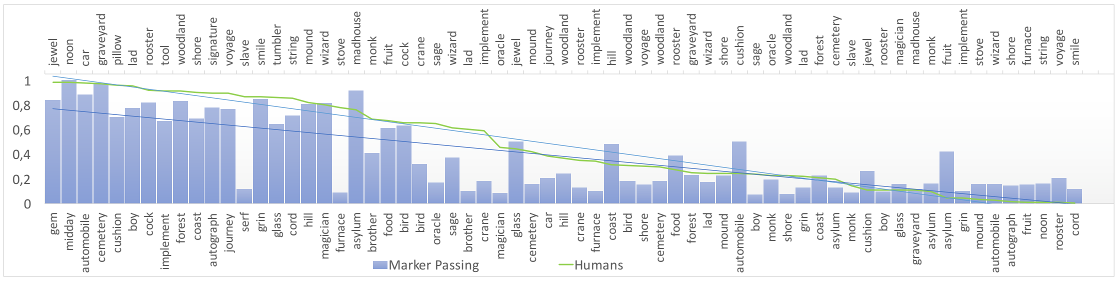
We have tested our semantic similarity measure on other data sets as well. With the WordSim353 [117] data set, we achieved a Spearman correlation of 0.553 whereas the results in Finkenstein et al. [117] reach 0.41 at the best and a linear combination of those approaches they reach 0.55. With the Mtruk [318] data set, our approach reached a Spearman's correlation of 0.551 in contrast to the 0.63 of Radinsky et al. [318].

Since the Pearson's and Spearman's correlation account for the overall result of the experiment, we will discuss the results in more detail in the next section. This is done so that we can analyze the strong and weak points of current approaches including ours.

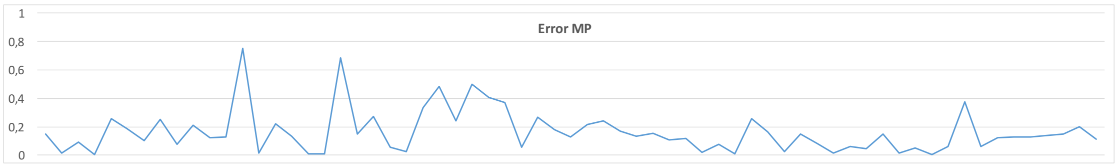
10.1.17. Discussion of the Results

This section discusses the result of our experiment and compares them to the three selected example measures in more detail. We will look at how the different similarity measure performs on the similarity scale (from similar concepts to semantically distant concepts) and identify week sports which could be improved in future extensions the approaches.

Figure 10.4a illustrates the reached results for the RG65 data set, depicting our results. The x-axes represents the similarity of a word pair out of the RG65 dataset. The y-axis listed the



(a) Result of the Marker Passing approach with the RG65 dataset.



(b) Error of the Marker Passing approach with the RG65 dataset.

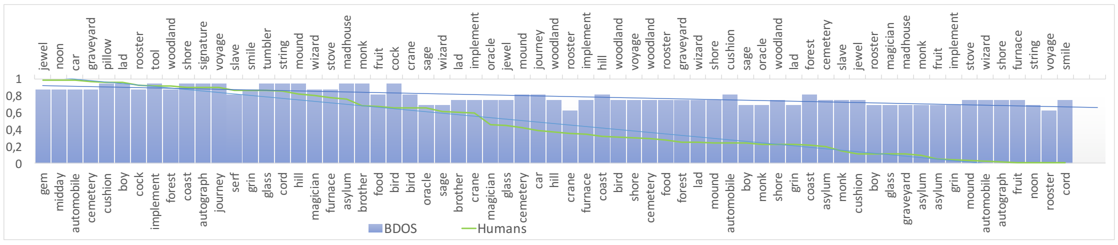
Figure 10.4.: Result and Error of the Marker Passing approach with the RG65 dataset.

65 different word pairs ordered from semantically close concepts (synonyms in the best case) to concepts at greater semantic distances. The underestimation gets worse the smaller the semantic similarity is until at the far end the measure overestimates the similarity. However, we can see that the linear progression (dark blue line) of the MP approach is closest to the human (green line) guess of similarity. Overall the MP approach is in average close to the human average compared by their line progressions (the light blue line is the linear progression of the human estimate). The best average performance of the other approaches is reached by the ELKB approach shown in Figure 10.7a, which overestimates the similarity most of the time. BDOS and Word2Vec (see Figure 10.5a and Figure 10.6a) overestimate the semantic similarity of far concepts consistently. This happens because the general knowledge sources like WordNet or corpora always find a path between two concepts in the example of BDOS.

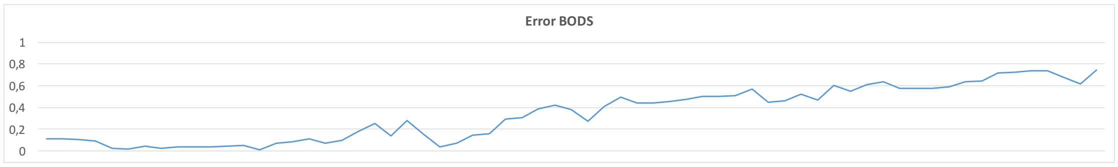
Figure 10.4b shows the error of our approaches. At this point, we can identify the concepts where the approaches deliver good/bad results. For example, it can be noticed that the error of MP is greater in the mid-range of the semantic similarity and that the reference measures get worse with a rising semantic distance.

There are two pairs of words which create an especially high error. Those are “serf - slave” and “furnace - stove”. In both cases, one of the concepts is a hypernym of the other one. If we analyze the information sources, e.g. slave and surf, we can see that the definition of WordNet does not contain a connection to slave and Wikipedia only noticed slavery as a hypernym of serfdom. WordNet has no connection between serf and slave not even as a synonym. This explains why the Marker Passing approach does not find similarity between those two words. This is a good example of what happens if the underlying information sources of the decomposition are missing facts or an incomplete. In this case, the Marker Passing approach fails.

Another curiosity the results in the Marker Passing similarity measure is that starting from a similarity of ca. 0.7 or less the measure produces a greater error. Which then reduces until the similarity of dissimilar concepts are overestimated at the end. It seems like concepts with this semantic similarity are so far apart in the created decomposition, that it becomes harder for the markers to reach common concepts except in some exceptions, which are over estimated like ”asylum” and ”fruit”.



(a) Result of the BDOS approach with the RG65 dataset.



(b) Error of the BDOS approach with the RG65 dataset.

Figure 10.5.: Result and Error of the BDOS approach with the RG65 dataset.

The WordNet-Path-length approach (BDOS) (see Section 10.1.1) in comparison finds nearly the same similarity for close as for distend concepts. The result of the BODS approach can be seen in Figure 10.5a. We can see that the mean of the similarity from the closest to the furthest concepts is almost the same. It seems that the path length in WordNet alone is not enough information to distinguish between the word senses given by the data set RG65. Because of the right tendency of the BDOS result, the correlation of the result in Table 10.4 the Spearman's correlation is still 0.81, which is an overall impressive result regarding the simplicity and computational speed of the approach. In Figure 10.5a we can see that most of the results are overestimated. And it seems that the distance between the concepts in WordNet seam all to have more or less the same path length. Since the BDOS algorithm uses two kinds of semantic relations (Hypernyms/Hyponyms and Meronyms) the distance is the minimal path of along these relations in the WordNet graph.

Shown in Figure 10.5 this leads to good results with concepts which are semantically similar. With concepts with a similarity higher than 0.8 the BDOS approach even beats the results of the Marker Passing approach. Starting from a certain similarity threshold, the error constantly increases until the error reaches almost 0.8. This leads to the conclusion that the minimal path length might measure the semantic similarity, but with a decrease of similarity of the concepts the BDOS's results worsen. The performance of the BODS might be increased by using different semantic relations like antonyms. Further, the restriction of the hypernym relation to a certain maximal height might improve the overestimation of the approach. Since building hyponym paths, concepts are abstracted to the root node, which leads to paths of almost the same length, if no other path is found first.

The Word embedding approach Word2Vec depends deeply on the training corpus it uses to train the neuronal network for later classification. The result in Table 10.4 are taken from Mikolov et al. [269] since we were not able to reproduce their results. Our result with the training corpus of the Leipzig Corpora Collection [11] we reached a Pearson correlation of $r = 0.25$.

Nevertheless, Figure 10.6a describes our results, which can be analyzed as well: The average is lower than with the BDOS approach. This moves the area in which the metric is correct towards the middle of the spectrum between close, and distant semantic similar words. In Fig-

[11]http://corpora2.informatik.uni-leipzig.de/download.html

ure 10.6b this can be seen in the error rate reducing towards the end of the first half of the data set. Further, there seem to be words for which the approach works better than for others independent of the semantic distance between those words. This can be seen on the error dropping to almost zero from time to time, e.g., for magician and wizard but as well for asylum and cemetery, where magician has 42 occurrences and wizard hat 184. On the other hand, asylum had 376 occurrences and cemetery had 512. Since slave has 434 occurrences and monk 365, the occurrence rate is not the source of the error. From the approach of word embedding, it should be clear that words not occurring in the same context should not be seen as similar. Consequently, this depends on the length of the tests and the vocabulary used.

The Word2Vec approach has its benefits as well: The embedding is independent of word type or grammatical form. Hence on data sets with multiple word types, they might perform better [17]. Additionally, the performance can continuously improve if new training data is found. In contrast to the Thesaurus approach ELKB, where the Thesaurus needs to be improved, which is subject to research [192].

The Word2Vec similarity performance depends on the data set on which the ANN has been trained. We selected the English one million word News corpus from the Leipzig Corpora Collection[12] [313]. Since our selection might have been suboptimal, we used the results of the original references [269] to compare them to our results.

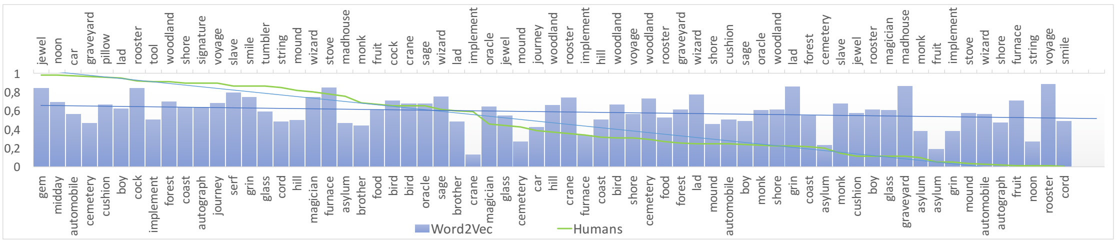
As a final remark to the results of the Word2Vec approach, we need to notice that the co-occurrence is not a semantic similarity measure but a measure of relatedness. This relatedness can be seen in words often co-occurring like homeless and shelter, which are semantically rather the opposite since a homeless person is one without the shelter provided by a home. But they still seem related in the Word2Vec approach. For our evaluation, this observation does not make any difference, since the performance against humans is measured and not an objective or logical semantic distance. Accordingly, the result measures how well an approach can simulate human semantic similarity guesses. Because of the co-occurrence of certain words, even humans might be influenced to guess the semantic similarity wrong. This is also true for words like sun, star, and planet. Where a human all sees them as round objects of astronomy, where an expert would disagree on their similarity. Since this argument could be held for most concepts, we rest our experiment on the simulation of average human estimates.

We can see that the thesaurus based approach (ELKB) (see Section 10.1.1) estimates well when closely related concepts are the problem, whereas the semantic similarity gets smaller the metric becomes less accurate and unable to connect two concepts. In Figure 10.7b we can see that ELKB's error increases with the decrease of semantic similarity of the concepts. There are errors even in the semantically close concepts when the thesaurus does not include a concept. Then the similarity is set to zero. This explains the two error peaks at the beginning of the Figure 10.7b.

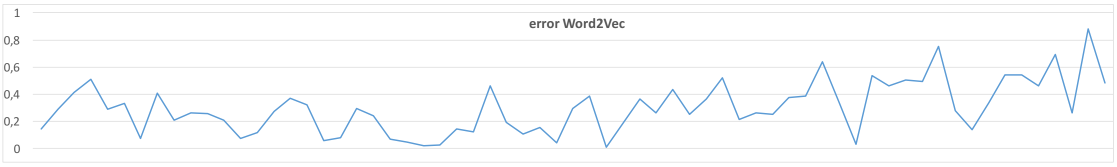
The ELKB approach comes close to the human's similarity estimates in average but only reaches a Spearman's $\rho = 0.65$. The first thing to notice about the result is that there are discrete steps in similarity. The steps size is 0.125 since there are only eight steps in the thesaurus reaching from no connection to being in the same frame. This has the effect that there can not be a fine grained result like provided by human estimates. This can be seen in Figure 10.7a in the first 21 word pairs, where most of the estimates are mapped to 1.0.

If one of the words is not contained in the Thesaurus then the ELKB similarity measure returns

[12] `http://wortschatz.uni-leipzig.de/en/download/`

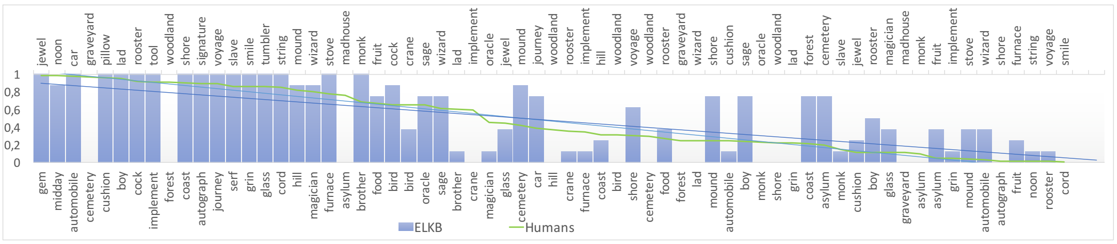


(a) Result of the Word2Vec approach with the RG65 dataset.

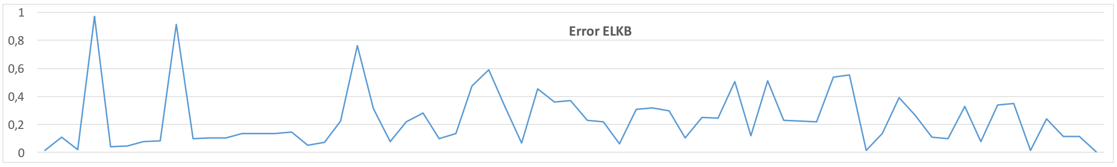


(b) Error of the Word2Vec approach with the RG65 dataset.

Figure 10.6.: Result and Error of the Word2Vec with the RG65 dataset.



(a) Result of the ELKB approach with the RG65 dataset.



(b) Error of the ELKB approach with the RG65 dataset.

Figure 10.7.: Error of the ELKB approach with the RG65 dataset.

a zero. Which can be seen on examples like cemetery and graveyard or forest and woodland in Figure 10.7a. In our dataset, these zero estimates occur more often the farther the words are semantically apart. This is because the algorithm cuts the search for a path from one word to another when it reaches a set count of eight. In particular, paths longer then eight are counted as no path found and with that to a similarity of zero. This can be seen on grin and implements 0.125 and grin an lad 0, lad and brother 0.125 and smile and grin 1.0. Here grin and lad are both parts of the Thesaurus, but grin and lad are too far apart for the approach to find a similarity at all.

The ELKB approach has less error for distance word pairs as it returns zero if no distance is found. As a result, the missing error for semantically distant word pairs here is due to failure of the approach to handle distant words. BDOS, on the other hand, has almost no error for close concepts. Here short paths between two concepts can be found in WordNet. But the further the distance of the words, the less accurate BDOS becomes.

A preliminary analysis with the Stanford Rare Word Similarity dataset [244] of 2034 words has yield the result of Spearman correlation of 0.17 and a Pe arson correlation of 0.19, which is subject to improvement now. Furthermore, the extension to have a WSD algorithm which uses context can be created through the following steps: The contextual words are decomposed, the results are merged into the graph and markers are passed to them. Then the word sense is selected with the most semantic similarity by identifying the nodes that received the most activation from multiple origins. The parameters for such an algorithm are subject to future work.

Conclusion

In conclusion, we can say, that the Marker Passing approach combines with a semantic decomposition can be used to build a semantic similarity measure. Although the success in the performance of our measure, the vast amount of potential parameters and the different design decisions, leave still room for improvement. With the infinite specialization methods for the Marker Passing, there could be other interpretations which are even more successful. More effort could go into designing more sophisticated similarity measures, but we see this as a proof of concept. Through the learning of the parameters, the parameters might be specific to the data set, we used for the evaluation, but never the less, we can show that our approach is applicable for measuring semantic similarity.

As a result, we were able to show that the MP approach can beat the state-of-the-art on the RG65 data set concerning the Spearman's correlation. A detailed analysis of the result has shown that with growing distance of the words compared, the error of the approach gets worse starting from a distance of approximately 0.7.

The errors in our semantic similarity measure could be from including word senses which are not relevant in human semantic similarity estimates. The next section will, therefore, tackle the problem of word sense disambiguation.

10.2. Experiment 2: Word Sense Disambiguation

Word Sense Disambiguation (WSD) is the act of selecting a word sense of one word in a context of use[13]. Therefor we can describe our WSD with the following signature:

$$WSD : Concept \times Concept^{*} \rightarrow Definition$$

The first concept is the target concept, the second list of concepts is the context in which the target word is used. The result of the WSD is one of the word senses (here called definition) of the target word.

Now that we can distinguish the semantic similarity of word pairs, we start a new experiment: Finding the right word sense of a single word in a linguistic context. Since a sense of a word can be described by its definition, this experiment selects one definition of the target word fitting to the meaning of the word used in a context[14], and then look up the sense key of the word sense described by the definition to see if we have selected the right one.

Figure 10.8 describes the Marker Passing approach of a word sense disambiguation algorithm. Here the target Word, its context, and all its definitions are decomposed. The semantic graphs

[13]This work is based on the bachelor thesis of Lukanek [243] and Marienwald [252] who have implanted a first version of the marker passing algorithm for WSD.

[14]Here the context might be the sentence the word is used in

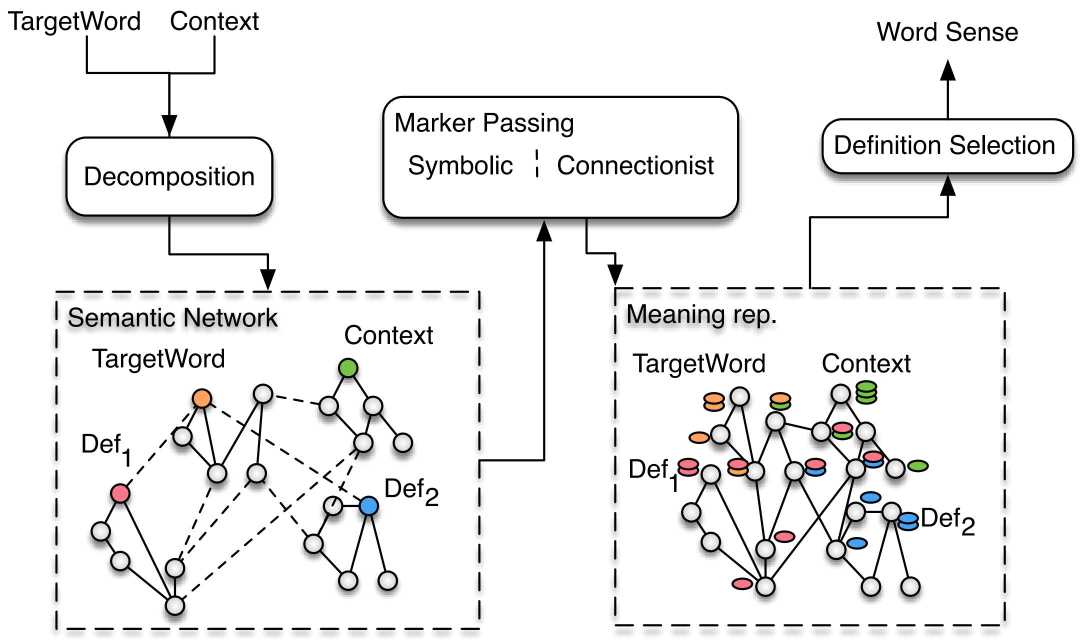


Figure 10.8.: Overview of the algorithm for Word Sense Disambiguation.

are merged to one graph. Each definition, the context, and the target words are given a start markers noting its origin depicted in Figure 10.8 as the color of the markers. After the Marker Passing, the definition with the most markers in common with the context and the target word is selected. The procedure is formalized in Algorithm 16.

In Algorithm 16 line 1 to 6 decompose all the available concepts namely the target concept (line 1) and all concept in the context (line 3 to 6). We then create initial markers for the target concept, all concepts in the context and the different definitions of the target concept in line 7 to 13. Line 14 then starts the Marker Passing, and line 15 selects the definition with the most markers collected in on the concepts of this definition. The Marker Passing parameters the, e.g., the in-, out- and edge-functions are selected like in the semantic similarity measure experiment described in Section 10.1.

The decomposition collects definitions and semantic relations of all dictionaries, and the Marker Passing uses all those relations to pass markers to concepts. In the end, the definition with the most markers is selected. All concepts of the context are decomposed as well and thus contribute to the semantic graph. Since the word senses of those contextual concepts are not known, all definitions are included.

In this section we will first analyze the state-of-the-art in Section 10.2.1. Then we take a look at the Marker Passing in Section 10.2.2 configuration and the data sets used in Section 10.2.3. We then compare our approach to the state-of-the-art and discuss the evaluation results in Section 10.2.4 and Section 10.2.5.

10.2.1. State-of-the-Art

The problem of word sense disambiguation is a simplification of the selection of word sense in context, where a sentence is given and one word, the so called "target word" is to be disambiguated. With the start of machine translation in the 1940's WSD has become subject to research [282]. The first to formulate the relationship of context and the meaning of a word was

Algorithm 16 Word Sense Disambiguation Marker Passing algorithm.

Name: Word Sense Disambiguation **Input:** Concept c_1, List<Concept> *context*, List<Definition> *definitions* **Output:** Definition

```
1: Graph $g_1$ = DECOMPOSE($c_1$,Null)
2: Graph $g = g_1$;
3: for all Concept $c \in context$ do
4:     Graph $g^{'}$ = DECOMPOSE($c$,Null)
5:     $g$ = MERGE($g$, $g^{'}$)
6: end for
7: NodeData init = SETINITIALMARKERS($g$,$c_1$)
8: for all Concept $c \in context$ do
9:     NodeData init = init+ SETINITIALMARKERS($g$,$c$)
10: end for
11: for all Concept $d \in GetDefinition(c)$ do
12:     NodeData init = init+ SETINITIALMARKERS($g$, $d$)
13: end for
14: NodeData result =MARKERPASSING(init)
15: return $\max_{Definition\ d \in definitions} \sum_{Concept\ c \in d}$ GETMARKERS(result, c)
```

Warren Weaver [401] in 1949. Quickly the research was concerned with the size of the context and its influence on the meaning of a word [404]. Later on, after the AI winter, the research on WSD continued [5]. Now there has been the specialization of three types of WSD algorithms [178]:

Supervised approaches use machine learning methods to learn parameters which distinguish word sense with a training set [357, 428]. Here methods like decision trees, Navïe Bayes or Support Vector Machine are used.

Unsupervised approaches use methods from machine learning like clustering, which do not need a labeled training set [3]. This sometimes is called “Words sense discrimination” since the approaches try to cluster word senses or contexts [73].

Knowledge based approaches use lexical resources like Ontologies, Dictionaries and Thesauri as bases for their disambiguation [4, 42, 43, 268, 381]. Here algorithms like the Lesk-Algorithm [229] try to find overlaps of the senses of the target word and its context.

Distributional semantics and with that word embedded approaches can be used to approach the WSD problem [178]. Here a target word is associated with the most co-occurrent words in a corpus, which does not reflect any semantic values, but rather a statistical usage of words in corpora.

Since our Marker Passing approach is unsupervised and knowledge based we compare the performance of our WSD algorithm against the following three approaches:

Hessami et al. [166] use a connectivity measure (the degree of centrality of a node) on a semantic graph extracted from WordNet. Hessami et al. create a semantic graph from the information out of WordNet by interconnecting each word with all its lexical relations to other words. In this graph, the degree of centrality is then measured for each word sense of the target word and the maximum is selected.

Veronis and Ide [389] do WSD via a “very large neural network extracted from machine readable dictionaries”. The word senses of the *Collins English Dictionary* are connected to the words in the definition of this word sense. This is repeated with all words which have definitions in the dictionary. This is equivalent to creating a decomposition by only using definition nodes. All words (nodes of the ANN) are reduced to their lemma. The sense nodes of a word are interlinked with inhibitory links (edges with negative weights). The activation then is passed from the input words and passed along the network. The paper of Veronis and Ide [389] does not specify the concrete parameters of the approach, e.g., the number of cycles and the inhabitation factor or the start activation is not specified.

There is one contribution which uses spreading activation to create a WSD approach. We will look at this approach more closely next: Tsatsaronis et al. [380, 381, 382] describe a WSD approach which utilizes WordNet as a semantic graph. With that the approach of Tsatsaronis et al. falls into the knowledge based category. In their publication Tsatsaronis et al. [380] they extend their approach to be unsupervised and with that fall in a second of the above categories and with that into the related work for our approach.

Tsatsaronis et al. construct a semantic graph from WordNet by adding all directly related concepts of the concepts making up the sentence, to the semantic graph. Each word sense of a word is connected with the concept itself, with edges weighted with positive weights (called activatory edges). The senses of one word are then interlinked with edges with negative weights (called inhibitory edges).

The activation spreading is then done by checking if a node j activation $A_j(p)$ has reached the threshold τ at pulse p. The Equation 10.13 describes how the activation is then propagated:

$$O_j(p) = \begin{cases} 0 & \text{, if } A_j(p) < \tau \\ \frac{(1-\frac{C_j}{C_T})}{p+1} * A_j(p) & \text{, else} \end{cases} \tag{10.13}$$

With C_j being the number of directly connected nodes and C_T being the total amount of nodes. Here p is the pulse count so that with each activation pulse the activation decreases. The edges are then weighted by their term frequency (the frequency to which the edge type is used) times its log-inverse of the total frequency of node occurrence. This approach has been evaluated using the Senseval 2 data set and reached an accuracy of 0.493.

The drawback of the approach described in Tsatsaronis et al. [380, 381, 382] is that a POS tagging needs to be present for the approach to work. Also, only WordNet is used to build the semantic graph; this means only words which are described in WordNet are part of the semantic graph.

The WSD experiment proposed here can be classified as a knowledge based approach since we are using our semantic decomposition as a basis for our disambiguation. Further, the Marker Passing is unsupervised. We propose an offline learning phase for the parameters of the Marker Passing algorithm, but the resulting algorithm can function without a human in the loop.

10.2.2. Marker Passing Configuration

The Marker Passing configuration in this experiment is the same as in the semantic similarity measure experiment. See Section 10.1 for more details. Every parameter of the Marker Passing is here the same as in Section 10.1. From the start Marker, the activation threshold to the termination condition, the approach here only changes by the amount of start markers initially distributed and the result interpretation.

The parameters of the weights and amount of start activation, as well as the termination pulse count and double activation limit, are learned with a genetic algorithm as described in Section 10.1.15.

10.2.3. Data Sets

The golden standard for word sense disambiguation is evaluated each year in the SemEval challenge[15]. This challenge is organized by Association for Computational Linguistics (ACL) Special Interest Group on the Lexicon (SIGLEX). Since our approach is based on English lexical resources like WordNet or Wikipedia, we had to select an English WSD challenge.

Sadly the Semantic Evaluation Exercises (SemEval-2016) held on the International Workshop on Semantic Evaluation is again a SemEval without WSD task. This reduces the choice of data sets to older versions of the data sets provided by SemEval challenge. This is the reason why the state-of-the-art in word sense disambiguation is tested on the Senseval Task 3 (Senseval-3 Task) called "Word-Sense Disambiguation of WordNet Glosses" [267].

This task consists of example sentences where one word, the so called **target word** is ambiguous. The corpus is taken from the British National Corpus, the Penn Treebank corpus, the Los Angeles Times collection and the Open Mid Common Sense corpus [267]. These are combined with the word senses of WordNet creating example sentences like the one shown in Figure **??**.

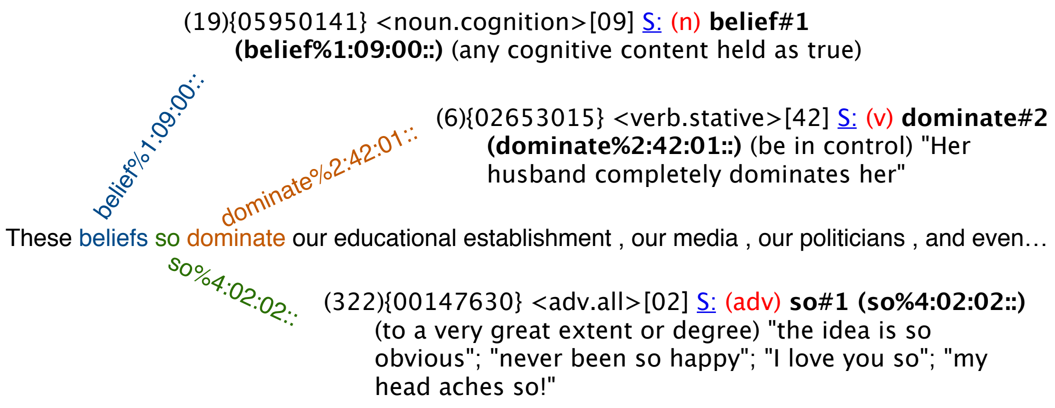

(19){05950141} <noun.cognition>[09] S: (n) belief#1 (belief%1:09:00::) (any cognitive content held as true)
belief%1:09:00::
(6){02653015} <verb.stative>[42] S: (v) dominate#2 (dominate%2:42:01::) (be in control) "Her husband completely dominates her"
dominate%2:42:01::
These beliefs so dominate our educational establishment , our media , our politicians , and even…
so%4:02:02::
(322){00147630} <adv.all>[02] S: (adv) so#1 (so%4:02:02::) (to a very great extent or degree) "the idea is so obvious"; "never been so happy"; "I love you so"; "my head aches so!"

Figure 10.9.: Example sentence with word sense from the Senseval Task 3 data set

In the Senseval data set most of the words of the example sentence are tagged with a WordNet word sense like in the example in Figure **??** the word "belief" with the sense key "1:09:00::" which references the definition "any cognitive content held as true.". The task now is, to identify the right word sense for each tagged word. For us, this means choosing the right definition we have found in the decomposition and looking up its sense key to return it as an answer.

[15]`http://www.senseval.org/senseval3/tasks.html` last visited on 02.03.2017

10.2.4. Evaluation Results

This section describes the evaluation results of our Algorithm 16 on the Senseval Task 3 data set. We compare our results to the best performing of each of the three categories described in Section 10.2.1. Here we compare the approaches depending on their accuracy regarding the selection of word senses for the target words in the different sentences.

Table 10.5.: Results of the WSD approaches on the Senseval Task 3 data set.

WSD approach	Accuracy
Hessami et al. [166]	0.482
MP	**0.385**
Tsatsaronis et al. [382]	0.365
Veronis & Ide [389]	0.308

Table 10.5 shows that the overall performance of our Marker Passing approach (MP) performs second best in the state-of-the-art. The results are reported in the respective papers proposing the approach[16]. Here further optimization of the Marker Passing parameters could result in a higher accuracy, but this is left to future work.

10.2.5. Discussion of the results

The selection of a definition or word sense of a word is a hard task which can be seen in the low overall accuracy reached by the here compared approaches. A perfect match has been reached, e.g., on the sentence “Here the experience of New York City is decisive.” with the target word being experience with the word sense: “the accumulation of knowledge or skill that results from direct participation in events or activities”. This works well because the word experience is already in lemma form. In other cases like in the sentence “Community involvement is an even worse idea.”, with the target word ”idea” with word sense “the content of cognition; the main thing you are thinking about”, our approach does not even select the right word: The Marker Passing resulted in maximally activating the synonym of ideas “thought” with the same word sense “the content of cognition; the main thing you are thinking about.”. This means we have selected some word sense which does not have a sense key in WordNet, consequently is not a possible solution to the question. This means out results could be improved, if additional problem specific knowledge would be added to the algorithm, selecting only definitions which have a WordNet sense key.

The calculation of the maximum activation seems to be too general since sometimes synonyms do have the same definition but are not considered a right response by the dataset. This drawback could be overcome by altering the result interpretation and filtering the synonyms.

Our result has been achieved without any syntactical information, which makes our approach a purely semantical one. In future work, the decomposition could be extended by syntactic information to improve the results.

Since the decomposition collects definitions of all dictionaries, a mapping to WordNet sense keys is needed if a definition is selected which has its origin, not in WordNet. In addition, the results could be optimized if the contextual concepts are disambiguated first, or at least their POS identified so that the decomposition does not contain all relations for all word senses.

[16]For further details the interested reader is referred to the Thesis of Marienwald [252]

In conclusion, we can say that the performance of an unsupervised approach do not reach the results of supervised approaches [20] but improves over the last years. Overall there is still space for improvement regarding the performance of unsupervised methods.

10.3. Experiment 3: Semantic Sentence Similarity Measure

This section describes the experiment on semantic sentence similarity. After having experimented with a word meaning similarity in Section 10.1 and word senses in Section 10.2 this experiment establishes if the here presented approach can be applied to represent the meaning of sentences. This means that the experiment conducted in this section analyzes how the Marker Passing approach can be used to create a sentence similarity measure. This is done to show that the approach is not restricted to single concept similarity. Here we take the idea of the semantic similarity presented in Section 10.1 and extends this approach to measure sentence similarity[17].

The signature of the given problem is defined as follows:

$$\mathit{SemanticSentenceSimilarity} : \mathit{Concept}^{*} \times \mathit{Concept}^{*} \rightarrow \mathit{Double}$$

Here the input is two sentences (given thought two list of concepts), and the result represents a measure of similarity as a double value. The resulting measure is zero if the sentence is not similar and is one if the sentences are equivalent in meaning.

Again, this experiment represents only a proof-of-concept to show that the here presented approach to representing artificial meaning can be used to measure the difference in sentence meanings. This means we do not optimize parameters of the Marker Passing to fit the data set. We will see that there are a multitude of design decisions which do not claim completeness nor optimality. We can regard this experiment as a starting point for future research on specializing the decomposition, the Marker Passing and its interpretation of to represent sentence meaning.

One result of the WSD experiment in Section 10.2 is that we want to identify sentences (in this case definitions) with the same meaning so that we can aggregate them and have less redundant definitions in the semantic graph. The resulting disambiguation could also be used during the decomposition itself since we want to eliminate duplicates in the definitions there as well to have less noise in the decomposition.

Figure 10.10 shows our abstract approach to a semantic sentence similarity measure. First, we decompose every word of both sentences. The resulting graphs are merged into one graph. In this graph, the concepts of the two sentences are marked with different markers. Our basic approach is depicted in Figure 10.10 where the different markers are shown as orange and green marked nodes.

[17]This section is based on the thesis of Schneider [351]

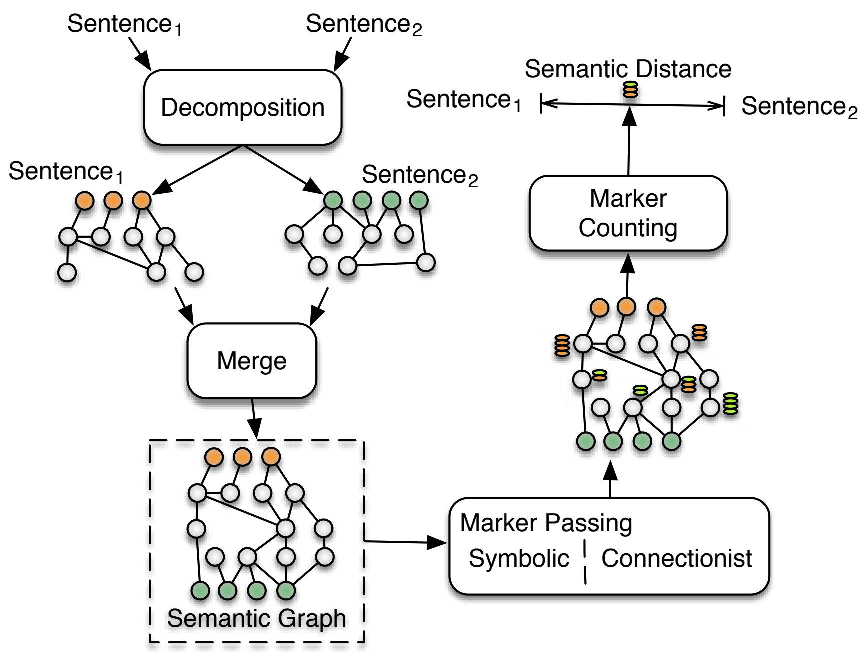


Figure 10.10.: Overview of the algorithm for our semantic sentence similarity measure.

The Marker Passing then passes the markers through the graph. The resulting marked graph is interpreted which is called "Marker Counting" in Figure 10.10. Here we have the basic assumption, that sentences which have similar meaning, have more concepts in common and with that, the resulting marked graph will contain more concepts marker with markers of two colors. The attentive reader might have noticed that the approach we take on sentence similarity is similar to the semantic similarity of words described in Section 10.1.

This approach to a sentence similarity measure is implemented like shown in Algorithm 17. Line 1 to 8 decompose all the words of the two sentences and merger the resulting graphs into one graph. Line 9 to 14 sets the initial markers on the concepts of the two sentences each time with the concept and the sentence it occurred in as an origin. The setting of the initial marker is adapted from Algorithm 7 (on page 168) to Algorithm 18.

In Algorithm 18 we have added line 4, which sets the additional information on the marker, from which sentence it was first placed on. More about the marker information is described in Section 10.3.2.

Line 16 in Algorithm 17 executes the Marker Passing based on the initial marking. The result is used in line 17 where the average activation is collected. In Line 18 the intersection of the words the two sentence have in common is calculated to account for words which occur multiple times and in both sentences. Line 19 combines those two parts of the result into our similarity measure. More details to the Marker Passing are discussed in Section 10.3.3.

We will first look at the state-of-the-art of semantic sentence similarity in Secontion 10.3.1. We then describe which marker information (Section 10.3.2) and Marker Passing configuration in Section 10.3.3. We then look at the data sets which can be used for sentence similarity experiments in Section 10.3.4. Afterwards we present the results of our approach in Section 10.3.5 and discuss them in Section 10.3.6.

Algorithm 17 Semantic Sentence Similarity Marker Passing algorithm.

Name: Semantic Similarity Measure **Input:** List<Concept> $sentence_1$, List<Concept> $sentence_2$ **Output:** Double

```
 1: for all Concept $c \in sentence_1$ do
 2:     Graph $g'$ = DECOMPOSE($c$,Null)
 3:     Graph $g$ = MERGE($g$, $g'$)
 4: end for
 5: for all Concept $c \in sentence_2$ do
 6:     $g'$ = DECOMPOSE($c$,Null)
 7:     $g$ = MERGE($g$, $g'$)
 8: end for
 9: NodeData $init$ = emptyNodeData
10: for all Concept $c \in sentence_1$ do
11:     NodeData $init$ = $init+$ SETINITIALMARKERS($g$, $c$)
12: end for
13: for all Concept $c \in sentence_2$ do
14:     NodeData $init$ = $init+$ SETINITIALMARKERS($g$, $c$)
15: end for
16: NodeData $result$ =MARKERPASSING($init$)
17: double avgActivation = GETAVGACTIVATION($result$)
18: double intersection = $\frac{(size(sentence_1+size(sentence_2)-(size(sentence_1 \cap sentence_2))}{\frac{size(sentence_1)+size(sentence_2)}{2}}$
19: return intersection + ($\frac{avgActivation}{(size(sentence_1 \cap sentence_2)*\textbf{StartActivation}}$)
```

Algorithm 18 Setting initial markers for sentences.

Name: InitialMarking
Input: Concept c_1
Output: NodeData

```
1: function SETINITIALMARKERS($c_1, sentence_i$)
2:     $ND$ = emptyNodeData($c_1, sentence_i$)
3:     $m_1 = \lambda(x). \begin{cases} \textbf{startActivation} \mid \varnothing & \text{, if } x = c_1 \\ \varnothing & \text{, else} \end{cases}$
4:     $ND$ = setOriginSentence($c_1$,$sentence_i$)
5:     $ND$ = setMarking $(ND, c_1, m_1)$
6:     $ND$ = setHistory $(ND, c_1, m_1)$
7:     return $ND$
8: end function
```

10.3.1. State-of-the-Art

This section will analyze the related work of semantic sentence similarity measure. The literature analysis will show which properties of sentences are used in the different measures and find suitable state-of-the-art to compare our approach against.

Sentence similarity has recently become subject to research. Most approaches origin from word similarity measures or represent Information Retrieval Methods which are originally used to compare bigger text documents [294]. Sentence similarity measures can be divided into two types of approaches:

Vector Space Models which are corpus-based methods also known as word embedding. In Vector space models the sentence is seen as a vector where each component of the vector is one word of the sentence. These vectors are then compared to each other mostly based on cooccurrence statistical information in corpora and some geometrical measures like the cosine similarity.

Feature Based approaches split the sentence into its words, and compare features of those words to each other and aggregate this comparison to become the sentence similarity measure.

Besides the standard classification into unsupervised and supervised methods in machine learning, the focus here is on the two types of approaches. As mostly in machine learning, the supervised approaches have a better performance on the test data sets and bare the usual deficits like the size of needed training data and the effort to creating this training data.

Gomaa and Fahmy [147] present a survey on text similarity measures older than the year 2013. These approaches are superseded and therefore are not part of this state-of-the-art analysis.

Corley et al. [60] introduce a semantic similarity measure for larger texts which can be used for sentences as well. They extract all verbs and nouns from the two sentences and select a corresponding (semantically the most similar) verb or noun in the other sentence. To choose the semantic similar verb or noun they use eight different distance measures. Amongst others they use Wu Palmer [418] and Jiang and Conrath [185, 418]. The rest of the words of the sentence are compared with a "word-to-word similarity" [60].

Mihalcea et al. [268] present a corpus based approach, so a semantic similarity measure of texts extending the one of Corley et al. [60] by integrating more part-of-speech classes in the word similarity, with the result of loosening the restriction of only comparing nouns with nouns. The result is weighted by the inverse occurrence frequency (so simulate entropy or amount information content) and normalized.

Approaches like the one of Islam and Inkpen [180, 334, 368] are all using different semantic distance measures of the text like the length of a "congest common subsequence", the semantic similarity of words, or the order in which the words occur. Stabcgev [368], for example, extracts a similarity graph from WordNet and bases its clustering on the occurrence of words in categories in this graph.

Oliva et al. [294], on the other hand, describe a syntax-based semantic similarity measure. Here the abstract syntax tree is calculated then the similarity of the words with the same syntactic functions are calculated, and the result is summed up and normalized. Here the part-of-speech are weighted differently. Oliva et al. introduce a fixed penalty factor of 0.3 for syntactic elements which are present in one sentence but not in the other. Those are then subtracted from the weighted sum of word similarities.

Taib et al. [375] describe a “features-based measure of sentences semantic similarity” (FMS3) combines word similarity of verbs and nouns and the order of words. Taib et al. [375] use a PageRank-Algorithm where the links referencing a concept (originally a web page) determine the amount of similarity of a concept. Hence, if a concept is used in a definition of a word, then it is seen as referencing this concept. The referencing concepts are weighted according to the count of incoming references and the weight of the concept which is referencing this concept. If a word has multiple definitions, the definitions are ranked according to their weights which lead to a sort of disambiguation of word senses.

Pilehvar et al. [308] as well describe an “Alignment-based Disambiguation of the two linguistic items and random walk” (ADW). Here the words of a sentence are represented by their definitions (they call this a semantic signature). This semantic network is the represented as a vector space where the vector contains elements which the definitions have in common. This is different form the concurrence vector spaces because it is based on properties found in the semantic network. The ADW seems to be working well on the datasets according to Pilehvar et al. [308] but the results are not reproducible with the open source variant [18]. For that reason, we neglect from listing those results in our state-of-the-art.

In conclusion, the state-of-the-art of sentence similarity does use a multitude of properties like word order and part-of-speech or syntax of the sentence and has analyzed different word similarity measures. In addition to the contribution of Taib et al. [375] even word sense disambiguation is used to select the right definition of the word in the context of the sentence. Missing in this approaches is the symbolic part of the meaning of the sentences. This means that logical concepts like negation can not be appropriately integrated into the sentence similarity measure. Representing the best sentence similarity measures researched so far, they form the best candidates as a comparison to our approach.

We selected the paper of Cloey and Mihalecea [268] and Pilehvar et al. [308] for the comparison with our approach since these are the most recent surveys on sentence similarity, and they use the same data set.

10.3.2. Marker information

The marker information for the sentence equivalent experiment is similar to the of the semantic distance with the addition that the marker carries the information from which sentence it has started. Consequently, we can define our marker as follows:

This is done to be able to treat those markers separately. If a concept is present in both sentences, markers of both “colors” are placed on it.

10.3.3. Marker Passing Configuration

The Marker Passing configuration in this experiment is the same as in the semantic similarity measure experiment, except the interpretation of the markers after the Marker Passing has been

[18]https://github.com/pilehvar/ADW/

Algebra *SentenceNodeData* = implements NodeData
sorts:
$Markers : \{(ofOrigin, marker) \mid ofOrigin : Concept, marker : Double\}$
opns:
$getSumFromOrigin : Markers \times Concept \times Concept^* \rightarrow Double$
$getSumFromOrigin(markers, origin, concepts) = \sum_{x \in markers} xifgeTOrigin(x) =$
$origin \wedge getConcept(x) = concept$
$emptyNodeData(concept, Concept^*) = \{emptyMarkers, emptyMarkers\}$
$setOriginSentence : Concept \times Concept^* \rightarrow NodeData$
$setOriginSentence(concept, sentence) = NodeData(concept, sentence)$

executed. See Section 10.1 for more details on how e.g., the in- and out-function or the termination condition are defined. The main difference is that the marker carries the information from which sentence they started out from. This information is used in the interpretation of the markers in the way described in Equation 10.14.

$$similarity = intersection + \left(\frac{\textsc{getAvgActivation}(result)}{\mid sentence_1 \cap sentence_2 \mid * \textbf{StartActivation}} \right) \tag{10.14}$$

Where $sentence_1$ and $sentence_2$ are two lists of concepts and *intersection* representing the activation of the set of concepts activation in both sentences.

$$intersection = 2 * \frac{(\mid sentence_1 \mid + \mid sentence_2 \mid) - (\mid sentence_1 \cup sentence_2 \mid)}{\mid sentence_1 \mid + \mid sentence_2 \mid} \tag{10.15}$$

The *getAvgActivation* gets the average activation as the sum of the activation of all markers of all concepts that are activated by both sentences. In Equation 10.14 we calculate the concepts present in both sentences plus the average activation of the concepts activated by markers of both sentences normalized by the total activation present after the initial marking. The resulting similarity is then again normalized to one.

With Equation 10.15we calculate the similarity of two sentences by calculating the ratio of equivalent words in both sentences in *intersection*. This means if the two sentences are equivalent, then *intersection* becomes 1. To this ratio, we add the normalized average activation of all concepts activated by markers of both sentences. This captures that if concepts are semantically closer together, then more markers of both sentences, carrying more activation exist. In extreme cases, this value can become larger than one, which makes a normalization to the interval of zero to one of the result necessary.

10.3.4. Data Sets

The state-of-the-art in semantic similarity measures is evaluated in the SemEval challenge since 1998 irregularly. The contests are documented[19]. Starting from 2012 the proceedings of the papers participating in the SemEval challenge are no longer available; this is why we chose 2012 as the year to take the data set from. Since the results of the last challenges in 2016 and 2017 do not have sufficient description of the participating systems we selected two surveys

[19] `https://en.wikipedia.org/wiki/SemEval` last visited on 02.03.2017

Sentence one	Sentence two	Similarity
A man with a hard hat is dancing.	A man wearing a hard hat is dancing.	5.0
A woman is cutting up a chicken.	A woman is slicing meat.	2.75
A woman is slicing big pepper.	A dog is moving its mouth.	0.0

Table 10.6.: Example sentences form the SemEval 2012 MSRvid data set.

Sentence one	Sentence two	Similarity
Neither was there a qualified majority within this House to revert to Article 272.	It did not not more of the qualified majority in this Parliament to return to the Article 272.	4.333
We often pontificate here about being the representatives of the citizens of Europe.	We are often here to represent the European citizens.	2.750
That provision could open the door wide to arbitrariness.	This paves the way for the of the Rules of Procedure here.	1.500

Table 10.7.: Example sentences form the SemEval 2012 SMTeuroparl data set.

which compare state-of-the-art approaches as a reference like described in Section 10.3.1 these papers are Cloey and Mihalecea [268] and Pilehvar et al. [308].

The data set was first presented at the SemEval 2012[20] as task six. This evaluation data set consists of a set of sentence pairs and their similarity. Table 10.6 shows three example of sentences of the data set and their similarity measure.

Here 5.0 is the maximum similarity, and 0.0 is the low bound. For our experiment, we have normalized this scale to the interval of $[0, 1]$ by dividing by 5 to make the results easier to interpret and comparable with other data sets. We have chosen the data sets: MSRvid, MSRpar, SMTeuroparl like in [308]. This selection has been made to be able to compare the results to the state-of-the-art. These tasks are from SemEval-2012 Task 6[21] which is an “Semantic Textual Similarity” task and from SEM 2013 Shared Task[22] which is also an “Semantic Textual Similarity” task.

Here the data set MSRvid and MSRpar contain stances out of the press and news like shown in Table 10.6. The data set SMTeuroparl, on the other hand, contains generic sentences, which is a harder task. The SMTeuroparl data set consists of sentences like those shown in Table 10.7.

The complexity of words used and the increase in grammatical complexity of the sentences explains why this data set is harder for similarity measures.

The golden standard rates these sentences from 0 to 5. With the following meaning taken from the SemEval-2015 Task 2 website[23]:

5 The two sentences are completely equivalent, as they mean the same thing.

- The bird is bathing in the sink.
- Birdie is washing itself in the water basin.

[20] `https://www.cs.york.ac.uk/semeval-2012/index.html` last visited on 10.09.2017
[21] `http://www.cs.york.ac.uk/semeval-2012/task6/` last visited on 10.09.2017
[22] `http://ixa2.si.ehu.es/sts/` last visited on 10.09.2017
[23] `http://alt.qcri.org/semeval2015/task2/index.php?id=semantic-textual-similarity-for-english` last visited on 03.03.2017

4 The two sentences are mostly equivalent, but some unimportant details differ.

- In May 2010, the troops attempted to invade Kabul.
- The US army invaded Kabul on May 7th last year, 2010.

3 The two sentences are roughly equivalent, but some important information differs/missing.

- John said he is considered a witness but not a suspect.
- "He is not a suspect anymore." John said.

2 The two sentences are not equivalent, but share some details.

- They flew out of the nest in groups.
- They flew into the nest together.

1 The two sentences are not equivalent, but are on the same topic.

- The woman is playing the violin.
- The young lady enjoys listening to the guitar.

0 The two sentences are on different topics.

- John went horse back riding at dawn with a whole group of friends.
- Sunrise at dawn is a magnificent view to take in if you wake up early enough for it.

An alternative data set is provided by Li et al. [231] by taking the word similarity data set of Rubenstein and Goodenough [336] and replacing the words with their definition. In this way, sentence pairs are created to measure sentence similarity based on the similarity of the two words from the Rubenstein and Goodenough [336] data set. This data set has multiple problems: The selection of a definition of a word removes the ambiguity of the task since now only one definition is presented, rather than just the word where the task is to find the right definition. Further, the selection of definition for one concept has not been motivated. With the result that, by selecting a definition which has a rare use, the task becomes un comparable with the results of Rubenstein and Goodenough [336].

10.3.5. Evaluation Results

The experiment is conducted with a decomposition depth of two and the set of decomposition parameters found in Section 10.1. We compare our results against the best state-of-the-art approach which we could find a publication for and which used the here chosen Dataset. The results of the ADW measure [308] have been neglected because even though the source code of their measure is available, their results were not reproducible. Even after trying all signature comparison methods and variating the parameters like the alignment vector size or whether to ignore stop words or not, the publication [308] did not indicate enough parameters so that the experiment can be repeated. The rest of the results have been taken from the survey papers of Cloey and Mihalecea [268] and Pilehvar et al. [308].

The results of the different approaches can be seen in the Table 10.8. The results are interesting since our sentence similarity measure MP is performing in average for the data set MSRvid but outperforms all approaches by far on the data sets SMTeuroparl and MSRpar. Since the SMTeuroparl data set is more complex not only in word choice but also in syntactical structures,

Table 10.8.: Results of the semantic sentence similarity experiment compared to the measure of Cloey and Mihalecea [268] which encapsulates many different semantic distance measures and Pilehvar et al. [308] stating the Spearman ρ and Pearson r correlation.

Approach	SMTeuroparl		MSRpar		MSRvid	
	ρ	r	ρ	r	ρ	r
MP	**0.50**	**0.45**	**0.52**	**0.49**	0.65	0.64
ESA_{WM}	0.30	0.22	0.28	0.25	**0.78**	**0.78**
$C\&M_{RES}$	0.25	0.25	0.41	0.46	0.74	0.73
RES	0.25	0.25	0.41	0.46	0.74	0.73
$C\&M_{LIN}$	0.23	0.23	0.41	0.45	0.70	0.70
LIN	0.23	0.23	0.41	0.45	0.70	0.70
$C\&M_{WUP}$	0.22	0.11	0.39	0.44	0.65	0.64
WUP	0.22	0.21	0.39	0.44	0.65	0.64
$C\&M_{JCN}$	0.20	0.20	0.39	0.44	0.65	0.65
$C\&M_{LCH}$	0.19	0.18	0.39	0.45	0.68	0.67

the result of our semantic measure is surprising. The cumulative error over all 459 sentences is 131.36 which is an average error of 0.286. This means that in average the results are still of by ca. 30% of the actual result.

10.3.6. Discussion of the results

The result of the sentence similarity is better than the state-of-the-art for complex sentences. With this experiment, we have shown that the use of a semantic similarity measure can be extended to the semantic similarity of sentences.

We can see in Figure 10.11 that the MP measure mostly under estimates the similarity. Fur-

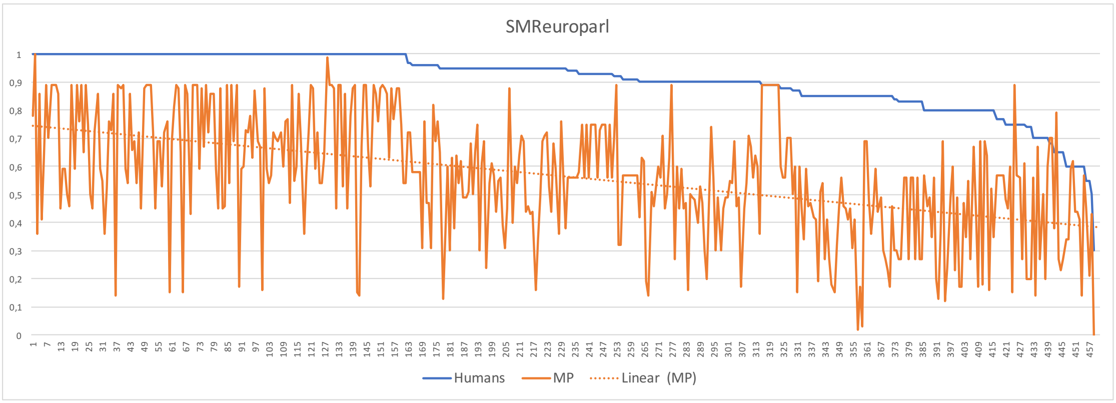


Figure 10.11.: Detailed result of the SMTeuroparl data set.

ther, we can see that the measure is almost never zero. Even though the semantic similarity is underestimated, the semantic decomposition seems to be large enough to find relations between the concepts of the sentences. In addition, we can see that the data set SMTeuroparl mostly consists of semantic similar sentences. Looking at the data set SMTeuroparl, most of the sentences are similar since most of the sample (ca. 440 sample out of 459) have a similarity of 0.7 or higher. Consequently, the results are biased towards measures which work well on similar sentences.

With the variation of the similarity, the Marker Passing approach is still ca. 30% of average error for the SMTeuroparl data set. The shape of the graph depicted in Figure 10.11 explains the bad correlation values in Table 10.8. With that, there is still potential left for improvement.

Figure 10.12 shows the result of the MSRpar data set. Here the sentences are longer and more complex than the example phrases from the MSRvid data set. An example here is: "In 2001 and 2002, wire transfers from 4 of the company's 40 accounts totaled more than $ 3.2 billion, prosecutors said." and "Wire transfers from four of the 40 accounts open at Beacon Hill totaled more than $ 3.2 billion from 2001 to 2002, Morgenthau said."

Here the example phrases are well distributed over the spectrum of semantic similarity with a focus on sentences with a similarity in the interval of $[0.4, 0.8]$. We can see that with the longer sentence length our measure starts to overestimate the similarity. The cumulative error of all 750 examples is 105.97, which is an average error of 0.141. This shows that even though the correlation outperforms the state-of-the-art, there is still potential for improvement. The data set MSRpar focuses on sentences which are between 0.8 and 0.4 (ca. 140 to 680 which makes 540 of 750 which is 72% of the sentences) on the human similarity value. This means that ca. 72% of the sentences are placed on 40% of the similarity range.

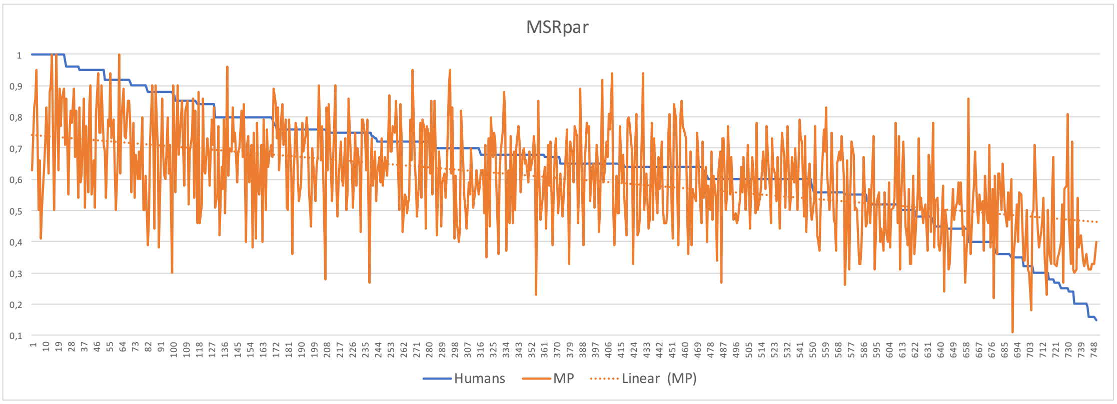


Figure 10.12.: Detailed result of the MSRpar data set.

Figure 10.13 shows our result on the MSRvid data set. Here the entire range of similarity is covered. The difference to the MSRpar data set is that the MSRvid data set consists of short sentences like shown in Table 10.6. Here almost all sentences consist of less than 20 words. In addition, the data set is crafted in a way that antonyms like "woman" and "man" are often interchanged in the two sentences. The cumulative error over all 750 sentences is 205.07 which yields an average error of 0.273. Looking at the MSRvid data set, the emphasis of the similarity is on sentences which are less similar since a lot of the examples (ca. 100 of them) have a similarity of zero.

The bad performance of our measure could be partially explained because those antonyms are skewing the semantic similarity. One can imagine that markers can pass over an antonym relation from, e.g., "woman" to "man" and therefore increase the double activation measure between those sentences. Besides, the MP approach takes into account which concepts are used in both sentences, so examples like: "The dog did bite John." or "John did bite the dog." are considered similar.

The results are not performing as well in comparison to the state-of-the-art on simple sentences, where the structure of the syntax is not as important as the words used. Because on

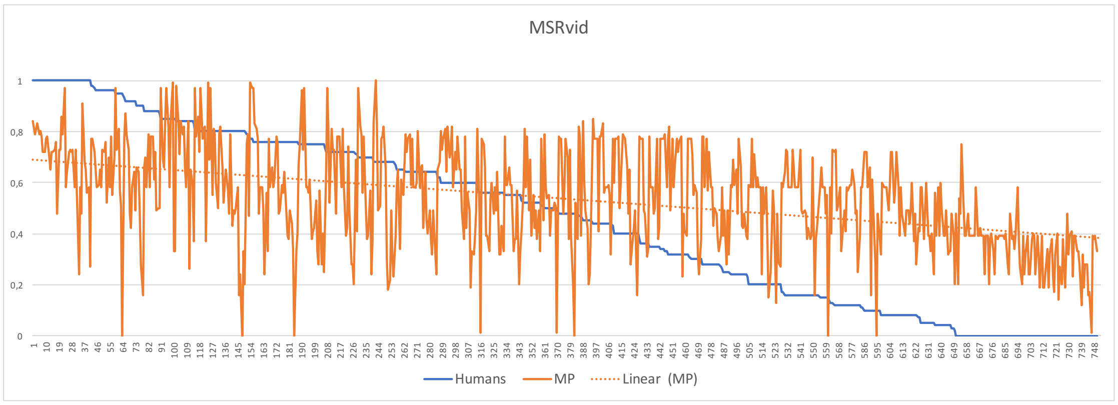


Figure 10.13.: Detailed result of the MSRvid data set.

more complex sentences, the syntactic analysis of state-of-the-art becomes more difficult, their performance decreases more than in the MP approach. Also, the words in those examples are more basic so that they are mostly included in WordNet. This means measures based on WordNet can perform well on those examples. Since WordNet is mainly hand crafted, this shows that the quality of our automatically generated information sources for semantic tasks are still improvable.

Furthermore, this experiment shows that with a simple extension of the same principle in token information and Marker Passing algorithm configuration, the problem of sentence similarity is not yet solved. This leads to the conclusion that further experiments are needed, where the marker information is extended to logical, perhaps syntactical information. In addition to that, the in-, out- and edge-function could make use of this additional symbolic information and further improve the results.

Further, we can conclude that the semantic representation of the words making up the sentence is not enough to represent the meaning described by the sentence. The addition of syntactic information thus is necessary to capture the logical content of the sentence. An example of this is a negation, which could be included in a sentence like: “John did not bite the dog.” With the Marker Passing approach, the negation is not reflected in comparison with, e.g., the sentence: “The dog did bite John.”.

10.4. Experiment 4: Semantic Service Matching

The experiments in Section 10.1, 10.2 and 10.3 showing the applicability of our approach on academic challenges. To show that our approach can also be used in challenges with more direct applications, we selected service matching as one use case. Service matching describes the problem of selecting the right service regarding a service request. This shows that our approach can also be used for more relevant problems in service oriented architectures[24]. Parts of this section are published in [101, 102, 103].

Extending the argumentation of Jennings [184], the development of pervasive, complex and distributed computing systems is ‘one of the most complex construction tasks humans undertake’. Challenges which directly evolve from the pervasive nature of applications are currently countered by development paradigms, such as Service Oriented Architectures (SOA) or Agent-

[24]This section is based on the thesis of Brand [37]

Oriented Software Engineering (AOSE), to name but a few. The basic idea of such development paradigms is to decrease the level of complexity of programming in dynamic environments. Euzenat and Shvaiko [91] refer to this principle as 'level of dynamics'. This level of dynamics increases the number of details that are left unspecified until the runtime of the application. Applications that we want to put our focus on are those which evolve and subsequently create additional heterogeneous problems. These problems are mainly due to the many different developers who are involved in the programming, each one making use of individually preferred technologies. However, it has been argued that systems with a high level of the dynamic at runtime and a heterogeneous character, should account for so-called self-* properties [346].

Getting services to connect is a hard problem lacking automatization. Even with standards like XML[25] and W3C Recommendation XSD[26] or the Semantic Annotations for WSDL (SAWSDL) [205] there is to few formalization to remove all the ambiguity entailed in a service description. This formalization is needed to automize the selection and execution of services, which is needed by dynamic distributed software as we have motivated earlier.

The problem at hand is the selection of a fitting service. The signature of the service matching problem is as follows:

$$\mathit{ServiceMatching} : \mathit{ServiceDescription} \times \mathit{ServiceDescription}^* \rightarrow \mathit{ServiceDescription}^*$$

This means given service request and a list of services the list of services are sorted according to their fit to the request. We call this kind of algorithm, ranking services to their equivalence to a given request, **Service Matcher**.

Reasoning algorithms utilized by Service Matchers use different metrics for different aspects of the service descriptions. One approach is to describe services in four components: **I**nput and **O**utput parameters, **P**reconditions and **E**ffects (IOPE) [373] defined in Section 3.2.

Some effort has been made to formalize semantic service descriptions. One of them is meta description languages we look at in Section 3.2. Such a language is OWL-S [255]. For the language to be less static, descriptions can be enriched with ontologies which structure semantic information of the concepts used in such a language [200]. These service descriptions lack contextual information which makes the information available for interpretation more abstract and context independent [240], with this in mind; the name is **semantic** service descriptions.

Semantic Service Matchmaking (we call it **service matching**) formalizes the problem of finding fitting services in an academic challenge [200]. The special case of the here extended Service Matcher is called "Semantic Service Match Maker" (short SeMa^2). For more details on the SeMa^2, the interested reader is referred to [102].

Figure 10.14 shows an abstraction of how service matching (blue) is described: A Service Matcher (here the example of the SeMa^2 matcher) gets a request for a service (green) and rates a list of advertising services (white) according to how well they fulfill the request. Here the Service Matcher can use a multitude of mechanisms like logical reasoning, metrics about function and non-function parameters like Quality of service (QoS) parameters or semantic models of the service interface to analyze the purpose of the service.

Each component of the SeMa^2 matcher matches a part of an advertised service description to

[25] `https://www.w3.org/TR/xml/`, last visited on 10.09.2017
[26] `https://www.w3.org/XML/Schema`, last visited on 10.09.2017

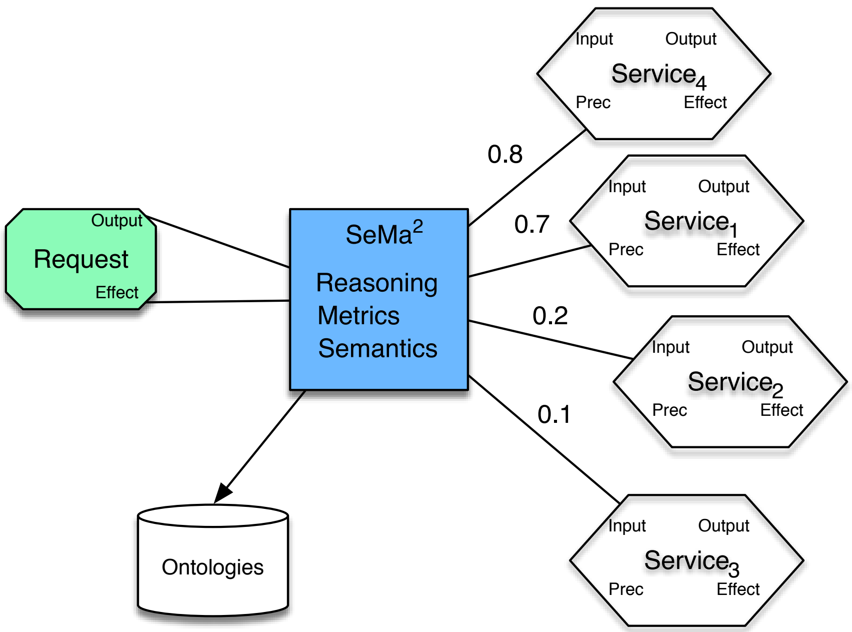


Figure 10.14.: Abstract Service Matching challenge.

a given request [103]. Here, each part of a service description is analyzed by a so-called **expert**. This expert provides an **opinion** about their degree of belief on how well the service request matches the service advertisement. Figure 10.14 shows that the request does not describe all parameters of the service, e.g., in Figure 10.14 the request is missing inputs and preconditions. This means that for example there is an expert looking at the service outputs and judging how well they fit together, but the inputs are neglected. The opinions of the relevant experts are then aggregated by **aggregating experts** who take the opinions of other experts and weight them together to create their opinion. This could be for example that the "input expert" consolidate the opinion of an input parameter name expert and a type expert. The first one checks if the names of the parameter are similar like "StartDate" and the second one checks whether the data type of the parameter is the same os similar.

We start out by looking at the state-of-the-art in Section 10.4.1. We then continue in Section 10.4.2 with the description of the Marker Passing experts. Then we specify the data set to test our Service Matcher in Section 10.4.3. The performance evaluation on this data set of our Service Matcher is done in Section 10.4.4. Finally we discuss the results in Section 10.4.5.

10.4.1. State-of-the-Art

In this section, we will look at the state-of-the-art of service matching and analyze how well those Service Matchers can be adapted to the domain of use. The goal of this section is to identify relevant related work on service matching to compare our approach against.

The challenge of service matching is well researched [200]. Further state-of-the-art can be found in [103] and [198]. The here proposed related work has one thing in common: all Service Matchers analyze the different aspects of a service description. Therefore, all of them need some semantic description of the functionality encapsulated in the service. To the best of our knowledge, there exist only two approaches that utilize machine-learning techniques to cope

Table 10.9.: Fixed scoring assessments or matcher pats as proposed by [23].

Relationship	Weight
Exact	1.0
Plug-in for Preconditions	0.6
Plug-in for Effects	0.4
Subsume for Preconditions	0.4
Subsume for Effects	0.6
Fail	0.0

with the challenge of aggregating service matchmaking techniques. In this section, we will at first give an overview of the related work of semantic service matchmakers followed by learning approaches that make service matchmaking adaptable.

M. Klusch and P. Kapahnke [200] present a hybrid service matchmaking component called *iSeM*, which performs logical as well as syntactical and structural matching. iSem participated in the 2012 S3 Contest and was the only matchmaker able to process the provided PE service specifications in SWRL. Besides logical matching filters for input and output parameters the solution applied a strict logical specification plug-in matching. This means that the component checks whether there exists a transformation of the requested/provided preconditions/effects, to infer from a requested precondition to a provided one and from a provided effect to a requested one. This process is called θ-subsumption and is in the case of iSem provided without any consideration of instances. The inferencing is being done after SWRL rules are converted into PROLOG.

Another work performing PE matching is SPARQLent [344], which not only performs matchmaking but also planning. It participated in the 2012 S3 OWL-S Contest and assumed that PE is described in the query language SPARQL. Hence, the selection process is based on query containment relations not only considering PE, but also input and output concepts.

In contrast to the other approaches the work of Valle et al. [70] named *GLUE* is based on WSML. Since WSML already comes with a conceptual model of service discovery, this work proposes a refinement that has a specific focus on mediating goals of different ontologies. The reasoning process itself is performed with F-Logic.

In the work of Lamparter et al. [219] the authors present a proprietary service definition based on OWL-DL. Besides input and output parameters, the algorithm also considers pricing functions described in SWRL and configurations under which a service is executable. The latter refers to some form of condition checking since the requester can search for services that provide a particular, desired configuration. The reasoning on services is being done with SPARQL queries.

Bener et al. [23] extend SAM (Semantic Advanced Matchmaker) by PE matching strategies based on OWL-S and SWRL. The matching procedure for conditions is separated into four matching modules, namely subsumption based scoring, semantic distance scoring, WordNet-based scoring, finalized and aggregated via a bipartite matching approach. The work introduces weights for the aggregation of different matching results, which are shown in Table 10.9 taken from [23].

The relationships shown in Table 10.9 describe how the request fits the advertised service. An ”exact” march, e.g., in output is weighted with 1.0. A ”plugin-in for preconditions” relationship means that the ”Request is subclass of advertisement = advertisement subsumes request”. Consequently a ”subsume” relationship for preconditions means that the ”Advertisement is subclass of request = request subsumes advertisement". Those relationships are defined for effects of a service respectively. The relation ship fail" means no match could be found [23].

The weights, in this case, are fixed and in fact not learned. Furthermore, those weights only concern discrete level of matches.

In conclusion, the described approaches rely on different languages for the description of conditions ranging from decidable ones, such as OWL-DL and WSML-DL to undecidable ones, such as SWRL, F-Logic, and PROLOG. However, in most of the related work on service matchmakers regarding PE matching the undecidable rule languages have been limited in its expressiveness to guarantee termination. So far, the service matchmakers have included different similarity measures and have introduced some weights to model their importance against each other. The weights started out to classify two types of similarity measures and got more detailed to the point where different parts of the service description like preconditions and effect are weighted differently. These approaches have one fact in common: the choice of the weights are fixed and do not adapt to the context of use.

To optimize the result of the service matching a learning phase can be introduced to adjust the parameters of a service matchmaker to the properties of the domain. The parameters to learn depend on the service matchmaker and accordingly its flexibility depends on the parameters that can be observed.

In Klusch et al. [199] the authors introduced a formal model of defining weights for the aggregation of different similarity measures with the names w_w-similarity and w_s-structural similarity measure. The aggregation method has been learned using a Support Vector Machine (SVM) approach based on training data. The matchmaker component that invokes this approach is designed to match SA-WSDL services.

M. Klusch and P. Kapahnke [197] introduce another learning service matchmaker by extending the approach of a prior work [195] for OWL-S service descriptions. Here matching results of different matching types are aggregated using a weighted mean. The authors introduce different types of matching results that are weighted. Firstly, approximated logical matching, which is divided into approximated logical plug-in and subsumed-by matching. Secondly, non-logic-based approximated matching, which is text and structural semantic similarity-based signature matching. The weights of this aggregation are also learned using an SVM. This supervised learning approach is replicated in our work, but with a different learning algorithm. The relevance set that is used to rank the matching results is reused with a genetic algorithm and a hill-climbing search.

I extended the winner concerning the average precision of graded relevance of the last service matching challenge [27] SeMa^2 and analyzed its components. Using the same dataset our approach can be compared to the state-of-the-art of service matches.

The extension of the SeMa^2 Service Matcher is described in [102]. Next, we will concentrate on the implementation of the components which implement our approach: the so called Marker Passing experts.

10.4.2. MarkerPassing Experts

In this section, we will look at how our decomposition and Marker Passing approach can be integrated into the Service Matcher SeMa^2. As a basis for our comparison of service we take

[27] `http://www.dfki.de/~klusch/s3/s3c-2012-summary-report.pdf`, last visited on 10.09.2017

the description of a Service (see Definition 2 on Page 24). We describe how the name matching, using our semantic distance measure described in Section 10.1, is implemented. Since the other Marker Passing experts of the service description, e.g., input, output, precondition, and effect are implemented similarly, we restrict the description to the name matching expert. Since input, output, precondition, and effect have more structured descriptions we split the analysis up into three parts:

Name: The name of a part of the description is analyzed like the same of the service. This could, e.g., be the name of an input parameter called ”Destination”.

Type: The type of a part of the description is analyzed by decomposing the name of the type. This could, e.g., be that the parameter ”Destination” is of type ”location”.

Value: The value of a part of the description is analyzed like the name of a service. The value a parameter is present, e.g., if an individual is referenced. An example could be that the destination parameter of a flight booking service is bind to the individual ”Berlin”.

For each of these parts, different experts are implemented. Their opinions are then aggregated into a more abstract opinion of a, e.g., input experts. This can be interpreted so that the input expert uses the opinion of the three sub-experts (name-, Type-, and Value-Expert) to form its opinion.

Algorithm 19 Marker Passing Expert - Name Matching.

```
Name: Name Matching
Input: String $Name_{req}$, String $Name_{adv}$
Output: double
1: List<String> $req$ = LITERALANALYSIS($Name_{req}$)
2: List<String> $adv$ = LITERALANALYSIS($Name_{adv}$)
3: return SUMMAX($req$,$adv$)
4: function SUMMAX(List<String> $req$,List<String> $adv$)
5:     return $\frac{\sum_{r \in req} \max_{a \in adv} (dist_{semantic}(r,a))}{|req|}$
6: end function
```

Algorithm 19 shows how the semantic distance measure $dist_{semantic}(\bullet,\bullet)$ (described in Section 10.1) is used to match service names. Here line 1 and 2 tokenize the service names into different words after the Camel-case convention and remove stop words. The outcome is a list of words, which are contained in the service name. This list is compared pairwise creating a similarity matrix. For each word of the request name, the best matching (in other words the maximum) of the advertisement service name words is summed up to create the matching score. This matching score then represents the opinion of this expert.

The mechanism described in Algorithm 19 has been used for the rest of the experts as well. The main difference is the function *LiteralAnalysis*() which selects the concepts that should be compared, from the part of the service description. Here the different parts, e.g., the description of a service is handled like the name. The text is normalized, each word is decomposed, and our semantic distance measure as described in Section 10.1 is used to assess the semantic distance of the concepts. Here all concepts of a request are compared with all concepts of an advertisement, and the maximum of each of those comparisons is summed up. The number of concepts normalizes this, and the resulting double is the value of the expert's opinion. The precondition and effect use Algorithm 19 as well but use the axiom names and parameter names as inputs. The same is done for the input and output parameters of a service.

10.4.3. Data Set

For the evaluation of the Service Matcher we used the OWL-S service retrieval test collection "OWL-S Test Collection" (S3) v4[28] which consists of 1083 services and 48 queries. This test set has been selected because the state-of-the-art Service Matchers have been tested using this dataset and by providing our results our approach can be compared to other matchers. Since in the test collection, the services are described in OWL-S 1.1, and we used the OWLAPI 4.1, we had to translate the services to OWL-S 1.2. The services are ranked with two kinds of ranking regarding the 1083 services: First, there is a binary rating where a service is either relevant to a query or not. The second part is a graded relevance where a service can be nonrelevant (0), potentially relevant (1), relevant (2) and highly relevant (3).

The S3 is the standard for semantic service matching with OWL-S. The preconditions and effects of the services are described in the Semantic Web Rule Language (SWRL). To compare our approach to the state-of-the-art, we used this test collection for our evaluation. This use case utilizes our approach to counter the semi-optimal performance of Service Matchers regarding the usage of different ontologies in the service description. The performance of the different Service Matcher can found in the results of the Service matching Contest S3[29]. We, therefore, compare our results with the performance of the SeMa^2 without the Marker Passing experts. We use the Semantic Service Selection (S3) contest [200, 376] to evaluate our extension of the SeMa^2.

10.4.4. Evaluation Results

The comparison of the SeMa^2 to the state-of-the-art in service matching is made with the S3 contest [200]. With an normalized discounted cumulative gain (NDCG) of 0.927 the SeMa^2 was able to outperform the state-of-the-art. This section, therefore, will compare the Marker Passing experts to the experts of SeMa^2 without the Marker Passing. We compare the performance of the Marker Passing experts to our experts because we do not have data about other implementations of the state-of-the-art.

We evaluated the performance of the different Marker Passing experts separately on the S3 data set to determine for which parts of the service description our approach is applicable. To be able to compare the resulting performance of the experts with the other experts of the SeMa^2 architecture, we will measure the NDCG to capture how well the similarity of services is estimated. To evaluate the introduced calculation overhead, we also measure the total execution time for the data set, the time it takes to evaluate one service advertisement and one service request. The performance of the Marker Passing experts are shown in Table 10.10.

In more details, the Marker Passing expert has been evaluated separately to the other experts. Here we analyze the average time it takes to compare one service advertisement (Avg time$_{adv}$)

[28] `http://projects.semwebcentral.org/frs/?group_id=89&release_id=380`, last visited on 10.09.2017

[29] `http://www.dfki.de/~klusch/s3/s3c-2012-summary-report.pdf`, last visited on 27.06.2017

and the average time it takes to respond to one request (Avg time$_{req}$ (sec)) as well as the total time it took to evaluate all 42 queries (total time (min)).

Table 10.10.: Service Matching results for the Marker Passing expert. The average time for a request time$_{req}$ is calculated over the 42 requests of the S3 data set. The average time for an advertisement time$_{adv}$ is calculated for the 1083 service advertisements of the S3 data set.

Expert	NDCG	total time (min)	avg time$_{adv}$ (ms)	avg time$_{req}$ (sec)
Name	0.819	8.12	34	10.7
Description	0,844	39.71	320	55.88
Input	0.521	4.177	36	5.03
Output	0.717	4.23	33	5.30
Precondition	0.628	4.43	34	5.44
Effect	0.877	4.82	33	6.03

In Table 10.10 the correlation between execution time and input size can be seen as the result of the description expert: A description consists of one or more sentences, which consists of more concepts than a service name or an input description. This is reflected in the average response time per request, which is ca. 56 seconds for the service description and only 10 seconds for a service name. In comparison to the other experts (shown in Table 10.11) they perform well, but with more time consumption.

When compared to the performance of the Marker Passing in comparison to the reasoning mechanisms of the SeMa^2 in its functional matching parts shown in Table 10.11 the Marker Passing can outperform the name, description and effect parts. Since the ”Text Similarity” expert combined name and description similarity, the evaluation with the Marker Passing experts is more detailed.

Table 10.11.: Service Matching results for the experts without Marker Passing. The average time for a request time$_{req}$ is calculated over the 42 requests of the S3 data set. The average time for an advertisement time$_{adv}$ is calculated for the 1083 service advertisements of the S3 data set.

Expert	NDCG	total time (min)	avg time$_{adv}$ (ms)	avg time$_{req}$ (sec)
Text Similarity	0.814	3.78	35	4.46
Input	0.793	3.89	34	4.66
Output	0.756	3.73	33	4.45
Precondition	0.635	4.4	39	5.26
Effect	0.565	4.06	38	4.82

From Table 10.10 and 10.11 we can see that the overall performance depends on the semantic information available to the approach. Where the Marker Passing is outperformed, we can see that the reasoning on, e.g., the input and output type beats the semantic information solely given by the input itself. If we analyze the structure of inputs, it is defined by its type and the variable name. The variable name can be something like “arg0” or “var1”, and the type can be something like “books.owl:Title.” This reduces the semantic information available for the Marker Passing to be analyzed.

A surprising result is the effect analysis. Here the intuition was that the logical reasoning would outperform the semantic analysis of SWRL rules. This seems not to be the case since the Marker Passing effect expert performs with an NDCG of 0.877 and the logical reasoning on the effects yield an NDCG of 0.565. The increase of performance might be due to the effect that the predicates of the SWRL description hold more semantic information, e.g. “isBookedFor(?Customer,?Flight)” contains five concepts which can be subject to the semantic analysis.

The evaluation result of all Marker Passing experts together on the 42 requests for the S3 contest data set with equally distributed weights yields an NDCG of 0.91 as shown in Figure 10.15. This shows, that without weighing the experts to their performance, the experts who perform badly on some aspect of the service description reduce the overall performance.

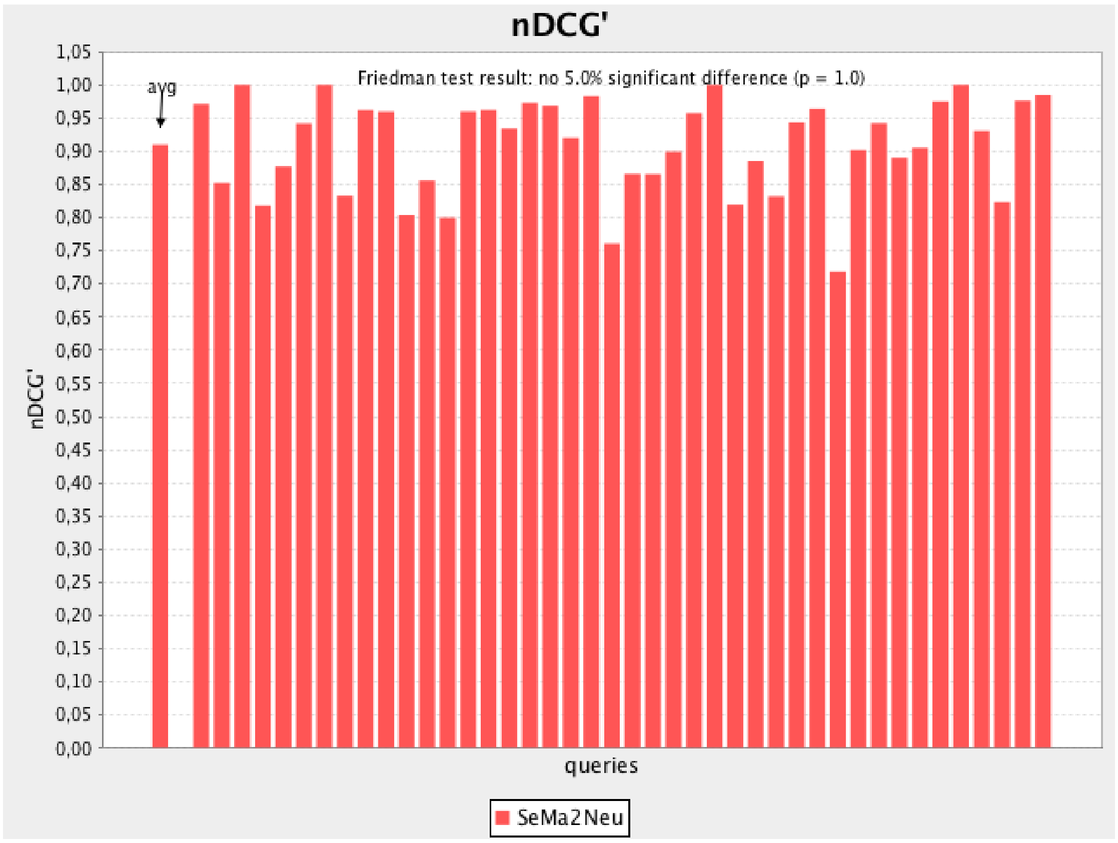


Figure 10.15.: NDCG of the overall result with all Marker Passing experts and equal weights.

Here the matcher “SeMa2Neu” is the Service Matcher with the Marker Passing experts. We have to emphasize that this experiment is done with equal weights. Meaning with our learning phase, we could still optimize the results to the data set by choosing different weights.

10.4.5. Discussion of the results

The results show that the semantic information created from the service description with the decomposition can augment the matching results as we have shown in Table 10.10 and Table 10.11 in the case of the effect, name and description expert. This is because all experts except input and precondition performing over average (NDCG 0.713) of the Marker Passing experts. However, this is not the only consequence of this experiment. Also, we can deduce from those results, that the semantic information from the service description used in the decomposition is not necessarily the information we suspected to be helpful in a match. This can be seen in the difference of performance for the input and output. However, we can see in Table 10.10 that the output experts outperforms the input experts.

The here introduced Marker Passing experts measure the semantic similarity of the words used to describe the different parts of the service description. These experts do not guarantee that a service can be replaced by another one since the logical truth value of precondition and effect are not evaluated. Neither does the Marker Passing expert analyzing the description, use syntax on the description. Therefore it is unable to determine exact details of the descriptions like, e.g., negation.

The evaluation of the matching performance shown in Table 10.10 can be seen as an indication of the fact that humanly created service descriptions use natural language and that the here encoded information can be useful for finding similar services. The overall performance of an NDCG of 0.91 shows that there is still performance optimization necessary. This optimization can be done by selecting the right weights for the Marker Passing experts, or by integrating the Marker Passing experts into logical experts.

In this experiment, we have pragmatic interpretations of the concepts used in a service description. Because of the markers set to additional concepts (e.g., on advertisement side), different concepts in a semantic graph get activated. Meaning like depicted Figure 3.9, we can interpret concepts in a context dependent manner. Consequently, having a conception of the concepts making up the service. In the service matching experiment, the effect is, that for different advertisement the concepts of the request are connected differently, and with that markers might pass over other word senses, or other relations creating a context dependent interpretation of the concepts.

In conclusion, the results speak for the use of some ontology matching since we could use our approach on semantic decomposition and Marker Passing to increase the performance of the semantic service matchmaking. Future work could be the combination of Marker Passing experts with other experts, e.g., where they perform better. This combination could be the case in the input- and output-Expert since the Marker Passing experts here perform less well than the non-marker passing experts. From matching one request to decide which service is suitable to be used in a plan will be the topic of the next section, where we extend the semantic analysis of service descriptions to create a semantic goal oriented heuristic.

10.5. Experiment 5: Heuristics in AI Service Planning

This section describes the experiment on the creation of a heuristic for service planning. With this experiment, we are going to use some of the results established earlier like our semantic similarity measure and the semantic sentence similarity measure. The goal of this section is to create a semantic heuristic which takes into account the start and goal state as well as the semantic service description. Consequently, this section defines what we see as our extension to those heuristics and which properties of heuristics are important for our approach.

We see a heuristic as a relaxed problem solution, which can estimate the usefulness of an action to a given goal. Pearl [304] describes a heuristic as “rule of thumb to guide one’s search.” Russell and Norvig [337] describe a heuristic as a function “$h(n)$ guessing the cheapest cost of a path from node n to a goal node.” This means that a heuristic is a function: $H(n) : State \rightarrow Double$

The function H(n) maps a node to the length of the path to the goal. This notation shows how static heuristics are: Their heuristic does not need the goal as an input, because, the goal is fixed in the heuristic and cannot change. The heuristic is therefore primarily designed for one goal. Ghallab et al. [134, p. 199] describe such a heuristic as "node selection heuristic" which selects the node *n* with the minimal heuristic value $H(n)$.

To create a heuristic we are going to look at the state-of-the-art in Section 10.5.1 We then argue why those heuristics need to be created at runtime and how this could be done in Section 10.5.2 and analyzes then search problems and their properties in Section 10.5.3. With that, we can describe an abstract approach on how to use this information to create a heuristic for service planning in Section 10.5.4. After that, we analyze the information available to the heuristic like the service description and the start- and goal-state, in Section 10.5.5. Then we look at the contextual information and analyze how pragmatic our heuristic could become in Section 10.5.6. Afterwards, we can describe our algorithm calculating our heuristic in Section 10.5.7. Then we select a data set which allows us to evaluate our heuristic in Section 10.5.8. The performance of the so created heuristic is thereafter evaluated in our experiment described in Section 10.5.9.

10.5.1. State-of-the-Art

In this section we want to find heuristics for AI planning to which we can compare our heuristic. Sometimes the heuristic and the search used are entangled. One example could be greedy search. We will analyze entire classes of heuristics if possible (e.g., greedy heuristics) to describe their properties. The goal of this section is to find heuristics which use as much semantic information as possible, by staying usable for general purpose planning.

Additionally, we want to gain insight into how general purpose heuristics are created. That supports the ability to describe our heuristic appropriately and makes our design decisions understandable. This means taking heuristics build for optimization problems and adapt them to work on our non-optimization problem. In other words, we reformulate our nonoptimization problem in a way, that we can adapt these heuristics for optimization problems.

The state-of-the-art of heuristics is twofold [341]: One part is well researched and theoretically founded. This part is concerned with the properties of heuristics and the abstract search methods like A* [304]. The other part is concerned with domain specific problem solving like the research of **Hyper Heuristics** and **Meta Heuristics**. These approaches try to find heuristics by combining old ones with machine learning approaches or optimizing hyper parameters. The second part can again be classified into the following categories [341]:

Deterministic vs. stochastic: Deterministic means that the heuristic returns the same output for the same input. Stochastic means that this is not the case, like when using random walks to search the state space.

Parallel vs. one solution at the time: Parallel means that the heuristic returns multiple solutions at the same time like in populations of genetic algorithms. Other heuristics just return one solution.

Fast vs. powerful: Fast heuristics provide less information about the problem but save resources. Powerful heuristics provide more insight but use more resources.

Classical vs. modern: Classical heuristics are problem specific and use, e.g., abstraction, landmarking, cost clusters or problem relaxation. Modern heuristics are Hyper- and Meta-

Heuristics which use the combination of classical heuristics and machine learning to select their parameters.

We will have a look at all of those categories of heuristics, concerning the use of semantic information and the domain independence. Afterwards, we will classify all the here discussed approach according to these categories. This is done because we want a powerful heuristics which is not stochastic and uses as much semantic information as possible. Why we do not want stochastic approach is explained in Section 10.5.2. Since service planning is not an optimization problem, we want our heuristic to find one possible solution as fast as possible, and we are not interested in parallel solutions.

Since we are looking for heuristics in non-optimization problems, this section focuses on the state-of-the-art of heuristics which can be applied in general purpose planning and do not necessarily need an optimization problem. Planning as a non-optimization problem is one particular type of planning. An overview of different planning problems can be found in [133, 222, 266].

Heuristics are needed if a problem is too hard to solve, and therefore an approximate solution can suffice [341], where heuristics can be seen as an approximation solution for such problems. Heuristic search is a fundamental methodology in problem-solving for hard AI [206]. Heuristic search means analyzing the abstraction of a search for a solution, by solving a similar but less complicated problem [34].

One way to create a heuristic is by relaxing the original planning problem [34]. One example relaxation is to ignore the negative effects of an action. In the language PDDL, this means ignoring the delete list of the effect of an action. Delete relaxation is a quite simple way of creating domain independent heuristics. The drawback is that we need to know which negative effects of an action are. In the case of planning with services, we do not know which effects are negative.

Another way to create heuristics is by abstracting the problem at hand automatically [202]. The idea is that we abstract the problem from a so called ground-level problem to a simpler to solve the abstract-level problem with fewer details. Abstraction in the sense of Knoblock [202] is to remove some facts from the description languages of the problem and with that making the problem easier. This abstraction assumes that ”[t]he existence of an abstract-level solution implies the existence of a ground-level solution.” (this is called the Downward Solution Property) [202, p. 54]. An example of such an abstraction is, e.g., if we want to plan a journey we can abstract from which exact airplane we plan to book, and rather plan the journey with its stops and how long we want to stay at each stop. This plan can be concretized to actual flights later on. Of course, there is the probability that there are no fitting flights for our abstract plan and in consequence, no plan can be found. Accordingly, the abstraction works only in problems where the Downward Solution Property holds.

Planning problems present many dead ends in the search space, ignoring facts like the delete list of effects results in bad planning performance [163]. Translation to a SAS^+ problem[30] restricts the search space and allows a mechanism to prove when tasks are unsolvable. That can be used for dead end detection. Because the effects of a service might depend on the input and could be entangled, a service cannot simply be split up unto N services which each having just one effect. Thus this is not an applicable approach.

[30]In SAS^+ each service has, e.g., only one effect.

Greedy Algorithm

A greedy algorithm has a selection strategy (heuristic) which selects the best next possible move (for use service to add to the plan). We see a greedy heuristic as a heuristic, which selects the best fitting service first. Since greedy algorithms are a general class of algorithms, we specify what we mean by ”best fitting service” next.

Greedy algorithms describe a best first search graph traversal algorithm. Here each node is marked with a heuristic value regarding its path length to the goal state. The successor function gets the next best state, for which the successors are part of the frontier. From this new frontier, we select the next best state. The successor function for a nonoptimization problem has been selected to be the overlap of facts in the state with the facts in the goal state. This means that the algorithm is greedy, because it selects those services first, which fulfill more facts of the goal.

This kind of heuristic has the drawback that no looking-ahead is taking place. That means choosing what is best in a current state might be suboptimal for following states. In consequence, greedy algorithms do not always find an optimal solution. In other words: the choice of every local optimum in one state does not guarantee a global optimum.

A* Algorithm

Evolved from the greedy approaches, the A* algorithm overcomes some of the drawbacks of the simpler “take the best fit first” heuristics [338]. The A* algorithm can still be seen as a greedy algorithm but with the difference that it uses an additional informative function which allows it to decide better how a greedy option is affecting the global optimum. This informative function sums up the cost of the path so far from the start state to the current state. The cost function of the A* algorithm is, therefore, the cost function displayed in Equation 10.16.

$$f(x) = g(x) + h(x) \tag{10.16}$$

In Equation 10.16 $g(x)$ represents the cost of the path through the search space so far to node x and $h(x)$ the heuristic value of node x. A* has some interesting properties one of them being the optimality guarantee of the solution of the heuristic $h(x)$ is admissible: A* finds the optimal solution if guided by a monotonic heuristics [304, Theorem 10].

There are further properties of heuristics which can be analyzed:

Completeness is the property of a heuristic if it finds a solution when a solution exists. By a solution, we define a path in the search space from state to goal state.

Dominance is the property of a heuristic H_1 compared to a heuristic H_2 if H_2 extends all nodes in the search space that H_1 expands. Thus H_2 does at least evaluate the same node as H_1.

Optimality is the property of a heuristic H_1 if it dominates all heuristics of a certain class H_C. Then we call H_1 optimal over the class H_C of heuristics.

There are additional properties of heuristics which allow proving optimality for certain search strategies using a heuristic with those properties. We will now list those properties and explain them:

Admissibility: A heuristic h is admissible if

$$h(n) \leq C(n), \forall n \tag{10.17}$$

where $C(n)$ is the minimal cost of the path n to the goal [304].

Consistent: A heuristic h is consistent if

$$h(n) \leq c(n, n^{'}) + h(n^{'}), \forall (n, n^{'}), \tag{10.18}$$

with $c(n, n^{'})$ is the cheapest cost of the state transition from nodes n to nodes $n^{'}$.

Monotone: A heuristic h is monotone if

$$h(n) \leq c(n, n^{'}) + h(n^{'}), \forall (n, n^{'}) \mid n^{'} \in successor(n), \tag{10.19}$$

where successor(n) denotes all nodes which are successors to n towards the goal.

With those properties, we can prove that, e.g., all consistent heuristics are admissible [304, Theorem 9 p. 83] or that the search algorithm A^* finds an optimal solution if it uses an admissible heuristic [304, Theorem 2 p. 78].

Admissibility is solely concerned with the quality of the out coming solution. However, in most problems, the best solution is not the only acceptable one. By looking for the best solution search strategies like A* waste resources on evaluating many solutions which are semi-optimal on the way to find the best solution. By leaving the restriction of optimality the admissibility and monotony of a heuristic can be relaxed as well. This requires more general search mechanisms than an A* [69] or the best first search [304, Section 3.3].

Lipovetzky and Geffner [236] extend the heuristics by a width-based exploration if the heuristic reaches a plateau. A heuristic plateau is given when in the search a set of service executions does not get closer to the goal state. Lipovetzky and Geffner shorten such plateaus by adding a width based search to the heuristic, by adding a novelty measure to the state. States with new facts meaning which differ more from the current state are visited first. The LAMA System [326], e.g., uses multiple heuristics to overcome this problem. Both approaches are not satisfying because they do not use information about the problem to speed up the search but use statistical improvements to speed up the search.

Redavid et al. [323] describe how service composition can be done by implementing plan control structures with OWL-S. Redavid et al. describe how rules in SWRL can be combined and propose a control structure on how they can be ordered. They create an “SWRL rules plan” by creating all possible rule combinations to fulfill the goal (the goal is formulated in SWRL atoms). The focus of this work lies in the description of service composition using OWL-S as language. The result is some kind of semantic heuristic because the SWRL rules are combined without variable binding. To explain, the resulting graph of connections between effect and precondition can be used to speed up the search for a plan. But here the approach neglects the variable binding and with that cannot produce an executable plan.

Srivastava and Kambhampati [367] separate the planning task into planning and scheduling. This separation leaves the planner with the task of finding all relevant services and gives the scheduler the task of bringing them in a specific order. This separation of tasks results in an abstraction of the problem in both parts: the planner does not have to care about the order of the services, relaxing the planning problem. The scheduler gets all relevant actions and their interdependence and needs to find a fitting schedule. Depending on the execution resources and the planning this approach might create plans which cannot be scheduled, and for that reason, fails to find a solution at all. If the planner checks for the schedulability of his plan, then the approach cannibalizes itself.

Since the example application of this thesis is the creation of heuristics for service planning, we have to look at alternative solution possibilities for the domain and problem independent creation of heuristics.

The other approach of guiding the search through the state space is called landmarking [190]. In landmarking known plans are analyzed, and services call these which have most in common and marke them as landmarks. The landmarks, e.g., present states, have to be passed to find a solution. Sometimes the landmarks can be brought into an order [172]. Landmarks are facts which must be true on the way of reaching a goal. The two sub problems of landmarks are: How to find landmarks, e.g., [310] and how to use the information given by a landmark to create a heuristic, e.g., [426]. These landmarks then can be used to decompose the search problem and use a local search to search from landmark to landmark [190] iteratively. The problem with landmarking heuristics is that the problem we want to solve has to be solved already to create landmarks. Because we want to create heuristics, which can be provided for the first time a problem is solved; landmarks are merely an optimization to our approach later on.

The state-of-the-art in general heuristics for planning problems is limited. The main Conference on AI planning and heuristic search is the International Conference on Automated Planning and Scheduling (ICAPS) / Conference on Artificial Intelligence Planning Systems (AIPS) [1]. Here starting from 1990 the last 27 years the community of AI planning has to discuss the different approaches to problem solving[31]. Here multiple specializations of the general planning domain have been identified. This leads to a classification of planning problems described in Section 3.3.

A common approach on heuristics creates an evaluation function through combination of the heuristic and the path cost [125] like described in Equation 10.16 on Page 211. With $h(n) \geq 0$ is the heuristic function (see Section 10.5.1 for more details). If $h(n) = 0$ the search is called not informed. The function $g(n)$ represents the cost of the path from the start state to the current state. In our example in Figure 10.19 for our current state (blue) $g(n)$ is 0.7. Korf [206] proposes a real time A* which, e.g., takes the distance from the current state to node n as $g(n)$. In this way the real time A* is relative to the current state, and once the planning algorithm leaves the initial state, every evaluation function is relative to the current state. With this, the real time A* becomes more like a best-first search which is semi optimal because the heuristic is almost the only guidance through the search space.

Nilsson et al. [285] argue that the cost of the path might be ignored since we are interested in a solution, not a cost effective solution. The purpose of the heuristic is then to speed up the search for a solution, not to guarantee an optimal solution. There are heuristics like the minimum step count to the goal which is called a **uniform action cost** [304]. These heuristics are equal to the step count if an action cost is equal to 1 [194]. With the uniform cost function of 1, the heuristic is admissible since it predicts that the next action is always only one step from the goal. An admissible heuristic is sometimes also called “optimistic”.

[31] See www.icaps-conference.org for the proceedings. Last visited on 10.09.2017

Greedy heuristics are those which count the overlap of the effect of a service to the goal and hence the usefulness of a service is how much of the goal it achieves [125]. Such a greedy heuristic is a quite simple heuristic which performs well, regarding its calculation effort. The search traverses the states by looking first at states which are similar to the goal.

As a baseline we use a **random heuristic** to get a lower bound on the problem. The random heuristic assigns each action a random usefulness in the interval $[0, 1]$ for any state. Such a baseline is needed because it is not possible to compare the here created results to results of planning competitions. The different nature of a STRIPS-like planning problem without semantic and service planning addresses the problem of a search for a plan in different ways. Classical planners are highly optimized to solve problems like the 15-puzzle [322] or the four-peg towers of Hanoi problem [207].

Haslum and Geffner [158] describe a greedy heuristic which they derive from STRIPS planning problems, where they ignore the delete list of a service effect and add only the add list to the current state to create a new state. This heuristic creates a relaxed planning problem compared to the original problem. The relaxed problem can show which services are minimally needed and then shows which items of the delete list of the used services need to further resolution. With the relaxed problem, the heuristic counts as a greedy heuristic since the search executes the services with the most overlap of their add list and the goal, first. This heuristic is admissible and is applicable for all STRIPS planning problems. Planners using such a search are sometimes called Fast-Forward-planners (or short: FF-planners). FF-planners are the basis of the most successful planners according to the International Planning Competition [164].

Another relaxation heuristic is the forward chaining heuristic proposed by Sierra-Santibanez [359]. Here services are rated “Good(a,s)”, “Bad(a,s)” or “Better(a,b,s)”. Good means here an optimal service a in state s. Bad services cannot be part of an optimal plan. The annotation better creates a partial order among the services. These ratings are generated manually by so called “action selection strategies” and are very problem-specific.

The approach of Haslum et al. [158] has been extended by an optimization in which abstraction of the effects called **Patterns** [157]. Those patterns are then subtracted from each effect and the start and goal state, creating abstract states which are mapped to the state space through these patterns. The patterns represent sub problem of the original planning problems, which are already solved. A pattern is a set of variable assignments which reoccurs in different states, e.g., the start and goal state, creating homomorphism abstractions. The drawback of those **Pattern Database Heuristics (PDB)** is that they do not scale up to real world problems [191].

The approach of Katz et al. [191] again optimizes the result of [157] by adding a **Causal Graph structure** and the **Domain transition graph** to the PDB heuristics. By including the causal graph, Katz et al. can create “causal graph structural patterns”, which approximates the original problem but are more abstract. This abstraction is done on SAS^+ formalization of the planning problems [13] which has an additional restriction for the domain descriptions. The simplification SAS^+ has the effect that abstractions like those done by Katz et al. [191] are possible. Here, e.g. “Post-uniqueness” means that an effect is only given by at most one action. Additionally, the “Binariness” restriction demands that all state variables have exactly two possible values. With an open world assumption and a distributed service development, we cannot fulfill those limitations, so these results cannot apply to our problem.

Learning the domain structure by observing plans is still subject to research [150]. Here Gregory and Lindsay [150] propose a model of action cost, which learns through the observation of plan traces. Even though the resulting cost function can be used as a heuristic, its creation, the observation of executed plans, puts this heuristic into the runtime. This interleaving of a plan- and runtime is out of scope for this work because we want to study the understanding of services and measure the degree of understanding by their use in a plan, not the other way around. Despite this being a valid approach, the idea is a trial and error mechanisms of learning the usefulness of services, because the services have to be executed to see if the plan succeeds. For certain services, e.g., indeterministic services, this might be appropriate, but we restrict our domain to planning problems where deterministic action are analyzed.

The same argument can be applied to approaches learning other planning properties than plan optimality [284]. The research of Marinescu and Coles [253], for example, is focused on heuristic creation in uncertainty with numerical effects, which we neglect here.

Using landmarks for creating a heuristic is done for instance, in the LAMA planner form Richter and Westphal [325] which performed well in the IPC 2008 [190]. Landmarks are used as a heuristic by Richter through counting fulfilled landmarks in contrast to unfulfilled ones [325]. In [325] Richter et al. combine this greedy heuristic search with landmarks with preferred operators, which take into account the usefulness of services by keeping them in a "preferred-operator queue". Those preferred operators are in consequence always tried first. Here a constant value of 1000 is added to the heuristic value of the service if it is a preferred one. Deciding which service is preferred, is part of the heuristic. Here again, the problem is formalized as a SAS$^+$ problem, which lets Richter et al. decide which service is a landmark (because its effect is unique). This is not given in our planning problem. In consequence, this kind of heuristic needs adaptation to be able to function with, e.g., the open world assumption.

Two research areas which analyze the automatic creation of heuristics are called **Hyper-Heuristics** and **Meta-Heuristics**. We will look at both of these research areas with the focus on understanding the underlying mechanism, which leads to the creating of a heuristic. The goal of this analysis is to extract ideas on how heuristics are adapted to the specific problem and how the needed information for the abstraction of a problem is found.

Hyper-Heuristics

Hyper-Heuristics can be seen to have the goal to automate the generation of heuristics. Most approaches do this by using some machine learning. The motivation here is to be able to generate more general search strategies to solve hard computational problems [40]. The basic idea behind Hyper-Heuristics is to use machine learning methods on a set of heuristics to generate a problem-specific heuristic. Burke et al. [40] separate the Hyper-Heuristics into two parts: heuristic selection and heuristic generation, which both are again separable into constructive heuristics and perturbative heuristics. Constructive heuristics start with an empty set of solutions and add new solutions until the problem is solved. Here the optimization is on selecting the best solutions to add. Perturbative heuristics start with a solution and optimize this solution by using local search and some optimization to improve the quality of the solution.

The drawback of Hyper-Heuristics is that the selection and generation of heuristics out of a heuristic part needs an learning phase. This learning phase forces us to create test examples of our problem or to optimize the heuristic during execution of the planning and the execution of the plan to create a feedback for the learning and selection components of the Hyper-Heuristic Approach.

Meta-Heuristics

Meta-Heuristics sample a search space and optimize a heuristic to fit a problem-solving approach [27]. Here methods like stochastic optimization can be used to search through possible parameter sets of a heuristic to generate a sufficiently accurate heuristic for a given problem. For the optimization to work the Meta-Heuristic algorithms need a reward signal to be able to search the space of possible heuristic parameters. An overview of Meta-Heuristics sorted by the problem or machine learning approach can be found in [298, 391].

The drawback of the Meta-Heuristics approach is that it needs a learning phase, where the outcome of the underlying problem is needed to be able to optimize the parameters of the heuristics. For our problem in AI planning, this means we need a learning phase, where we monitor plan execution or learn on old plans to create a Meta-Heuristic. Solving a problem finally becomes only better if the problem is already solved. The second argument against this kind of optimization is that it is done without the semantic knowledge and still can be improved by the selection of the right samples of the search space.

Conclusion

We now can conclude that building a heuristic is often problem-specific. Creating problem-specific heuristics has the benefit of having highly specific and performant heuristics. The drawback is their lack of generalizability. The approaches like Hyper- and Meta-Heuristics can create more abstract heuristics from special ones by using machine learning. Meta-Heuristics have the benefit of not having to design a heuristic but have the drawback that they introduce a learning phase.

As a conclusion, the state-of-the-art in the heuristic generation bases its approach mainly on the relaxation, abstraction, and landmarks of an original problem. Some of them use the domain description to structure the search space; others analyze services to identify landmarks, which help to break down the search problem. However, no heuristic found so far has used the semantics of the planning problem. The lacking use of semantics can be explained with the academic community of AI planning, which is mostly concerned with academic problems formulated in PDDL, which do not have a semantic description. This fact reflects in the data sets available for example planning problems discussed next[32].

The result of this state-of-the-art analysis is that we will compare our approach to a random heuristic to have a baseline for the search space, the uniform action cost heuristic and the greedy heuristic.

Table 10.12 lists the here looked at approaches from the view of the classification proposed by Salhi [341]. Some of the listed elements are entire classes of approaches, which are discussed earlier in this section. If an approach might fulfill properties or some of the approaches of a class do, and some do not, then we set the check mark in parentheses.

None of these approaches use semantics of the domain and problem description to create a more informed heuristic. Most of the approaches do not even use the semantics of the state, precondition or the goal to select services which could be of interest. The relaxation technique to creating heuristics additionally needs to decide which facts to ignore from the domain description of the planning problem. If we neglect the wrong ones, then the problem does not fulfill

[32]The interested reader is reflected to the international planning competition: `http://www.icaps-conference.org/index.php/Main/Competitions`, last visited on 24.07.2017

Table 10.12.: Comparison of heuristic approach.

Heuristic	Deterministic	Stochastic	Parallel	Fast	Classical
Greedy	✓	(✓)	×	✓	✓
A*	✓	×	(✓)	✓	✓
Uniform cost [285]	✓	×	×	✓	✓
Lipovetzky and Geffner [236]	×	×	×	✓	✓
Redavid et al. [323]	✓	×	✓	✓	✓
Srivastrava snd Kambhampati [367]	✓	×	×	✓	×
Random	×	×	×	✓	✓
Haslum et al. [158]	✓	×	×	✓	✓
Hyper-Heuristics	✓	(✓)	×	×	×
Meta-Heuristics	✓	(✓)	×	×	×
Landmarking	(✓)	✓	×	×	×
Relaxation	✓	×	×	×	✓

the Downward Solution Property, and the heuristic will not provide information for the original problem.

Since we do not have the data to learn heuristics, Hyper- and Meta-Heuristics, landmarking and Srivastava and Kambhampati [367] can be excluded for the comparison with our approach. This excludes all nonclassical approaches form Table 10.12. The same argument count for the Approach of Haslum et al. [158] because we do not have a PDB to lookup patterns. The approach of Redavid et al. [323] is excluded since the evaluation of SWRL rules can only be done with regards to the individuals given by the grounding. Lipovetzky and Geffner [236] can be considered as an extension of an already implemented heuristic by adding a novelty measure to state. Therefore they are not compatible as an own heuristic. With these exclusions, the abstract approach of Greedy search, a Uniform Cost function and as a baseline Random heuristic values are used to evaluate the performance of our approach.

Since Greedy heuristics are heuristics which use a best-first search, we will define a greedy heuristic using the facts stated in a goal state and compare them to the effects of a service. The Uniform Cost heuristic is an attempt to create an optimization problem out of our planning problem, by assigning every service call a cost of one.

As an extension of A*, our approach it will implement different behaviors, like a best-fist-search depending on the functions $g(x)$ and $h(x)$ used, thus we compare an A* approach against the other selected approaches. This seems like a restricted choice of related work. The next section will argue why this choice makes sense in the light of service composition.

10.5.2. Dynamic Heuristics during Runtime

We start by answering the question: How do we want to achieve creating heuristics which can keep up with the state-of-the-art described in Section 10.5.1.

To narrow this search space we humans use heuristics [386] to select services which seem more likely to bring us further to our goal. This behavior is mapped to planning algorithms, by defining problem-specific heuristics like the Manhattan distance [78] (for more details on heuristics see Section 10.5.1). Those heuristics are selected for specific problems estimating the distance of every state to the goal. Classically finding the right heuristic is done during the problem specification.

In a dynamic environment, where services and the goal are not known during design time, this approach fails, because fitting heuristics cannot be created during design because the services and the goal are unknown during design time. One possible approach is to create heuristics during runtime, taking into account the available services and the goal state. Both parts can change and above all need a mechanism for interpretation. Consequently, the problem of creating heuristics can be seen to be at least threefold:

Goal dependence: The goal is unknown during design time. Creating a heuristic which correctly estimates the usefulness of a service makes it necessary to know the goal.

Service interpretation: The available services are not known during design time. Using only the services which are known at design time, would reduce the adaptability of our agent to changing environments where services appear or disappear. If all needed services are known at design time, the developer can integrate them into a heuristic. At run time there is no developer. Therefore an explicit description of what the service does is required. Being able to include unknown services the interpretation of service descriptions is needed.

State/Context awareness: For general purpose planning to be an appropriate approach the problem and its rules are not known during design time. Hence the agent must be able to integrate unknown concepts into its knowledge representation to be able to reason with them. This allows the heuristic to adapt to changing problems and contexts.

Because it is difficult to pass all domain knowledge to the run-time by the developer, or services out of other domains might appear during runtime, the interpretation of domain knowledge has to be extendable during runtime.

As a result, there are three main tasks before we can create a heuristic measure of the usefulness of a service: the interpretation of goal, service, and context. We assume that these three descriptions are again made up of semantic concepts which allow our interpretation. The agent creating the heuristic for that reason needs to explain unknown concepts of those descriptions to itself. For this, the service description can contain the information needed, or the agent needs to create this information. What we want a service to have, is: *A self-explanatory description providing all information necessary for a reasoner in a given context.* This includes self-describing services like discussed in Oake et al. [290].

These self-explanatory descriptions can be created in two ways: The description provides all the needed information to the receiver, or the receiver adds this information. The first one we call the activity to self-explain: *Self-explanation identifies the process when an agent gives information about itself and its functionalities to other agents or human beings.*

During design time we can create the service description and change which information it contains. Indeed, we can include contextual information into the description giving the agent the ability of self-explanation.

How such self-explanatory descriptions can be created and how development tools can help create them, is described in [101]. Regarding the self-explanation, we have done some engineering work on providing tools to ease the semantic description of services. For a summary of these tools see Section A.1.

With such a tool chain it becomes easier to develop self-explanatory descriptions. Nevertheless, some problems remain: At design time we cannot foresee all the possible uses of a service nor the contexts the service is used in. The interpretation of service descriptions can be different than the intended service the developer wanted to create. Looking at the travel domain again, wanting to go from A to B could include the service of an ambulance, e.g., if B is the location of a hospital. This might not be intended by the service provider, but describing an ambulance service a travel planning agent could interpret the service in this way.

This makes the ability to understand new concepts in new contexts still necessary. We humans as reasoners need contextual information for interpreting most of the information around us, too. For example, the number '4870.88' might be interpreted in many ways. However, if we know the context: this number stands in a table with the row monthly income, we can reason with it more specifically. Sooriamurthi et al. [364] follow this fragmentation and present in an early work their view point on explanations in the AI research domain. One part of a self-explanatory description is thereby that contextual information is incorporated in the interpretation and creation of explanations to enable systems to adapt to dynamic situations and therefore introduce the use of pragmatics as a context-dependent interpretation of meanings. This is important since the explaining system might have to cope with partial observable situations while creating an explanation. Leake [224] substantiates this finding while arguing that with changing system goals the interpretation of explanation should be changed, too. The author also emphasizes that this requirement holds in different research fields like Psychology, Philosophy, and AI. At the same time, Leake [225] uses the factors plausibility, relevance and usefulness for explanations concerning COI regarding a given goal. Coming to the conclusion that "(m)any explanations can be generated for any event, and only some of them are plausible" [225]. The requirement we identify here is that a self-explanatory description must include not only regular information but also semantic information (about the meaning of the regular information) and context information for the context-dependent meaning. This correlates with the overall goal of self-explanation proposed by Müller-Schloer et al. [278] to enable systems to explain their current state, which seems to be difficult without providing contextual information.

To project the result of this discussion on to our agents: We want the agent to be able to self-explain new concepts. *To self-explain identifies the ability of an agent acquire new concepts and their relations and to integrate them into its beliefs.* The ability to self-explain is then used to create heuristics about new services.

We at least have two views on self-explaining: The first one is the learning part where the system learns new concepts. This part is used to analyze service description in hindsight when the description has been written, and the context of the author is lost. In this situation, the agent has to try to recreate the meaning the author of the description had in mind while writing the description. The second one is the one where semantic and contextual information is added to a service description to better explain the services of an agent to another. Both approaches on getting an agent to better understand the meaning of concepts used in service descriptions could be subject to research.

It should be noticed, that the self-explanation researched in this work does not explain the adaption of the system to some user as proposed by Sawyer [343] but rather to artificial reasoners which utilize the self-explaining descriptions for matching or planning tasks.

10.5.3. Dynamic Heuristics for Service Planning

This section will describe how we can create a heuristic which can estimate the usefulness of services given a planning problem. We look at the kind of search we want to use and then look at the theoretical properties this search has. We do so by looking at our planning algorithm. We finish up this section by describing how we build our goal oriented heuristic.

In service planning the actions which are choreographed in a plan are implemented by services. These services do not necessarily provide some cost function. This is because we do not take into account Quality of Service (QoS) description or Service Level Agreement because we are interested in a solution for a given problem, not the best solution. Therefore we cannot optimize for some quality parameter. This means that our planning problem is not an optimization problem. Since some of the services, like booking a plane ticket, are expensive and cannot always be reverted, we need a one-shot planning algorithm[33]. This means we cannot try out our plans to find better combinations of services.

Next, we look at some of the properties of a heuristic introduced in Section 10.5.1 and discuss their usefulness in non-optimization problems: A heuristic estimates the distance from a state to the goal state. Since we want to build a heuristic which can be used in general purpose planning, we do not have an optimization value, to which we can compare our heuristic. Since we do not have a cost function for our services, there is no actual cost for a state transition in our planning problem. Neither can we use an abstract optimization function like the step count since we do not know if the execution of one expensive service is better than executing two cheaper ones. This means asking for an optimal solution does not make sense in our case. Regarding admissibility, a heuristic is called admissible if it never overestimates the distance to the goal [304]. It is then sometimes called optimistic. This property is wanted because we can proof that with an admissible heuristic the A^* Algorithm returns an optimal solution [304]. Since we do not have an optimality citerion for our solution, the question of admissibility does not make sense in our case, and neither does monotone.

Since we are not interested in an optimal solution, because we do not have an optimization problem, we do not need our heuristic to be admissible. What we want to find is a solution without looking at too many services. This is the way we evaluate our heuristic: by how many nodes are extended during the search for the goal state [50].

The kind of heuristic we are looking for is mostly used in pure heuristic search or sometimes called best-first-search [304]. Creating such a heuristic includes analyzing the planning algorithm using the heuristic. This is because the search algorithm which does the exploration of service combinations use a heuristic differently. One example of state space planning is the idea of minimizing risk or maximizing the usefulness of the plan. This is why we explain out the planning algorithm first:

The Algorithm 20 describes a standard state-space planning approach [133, pp. 70] applied to service planning. Here, the planning is entirely in the service world without translating the service to the Planning Domain Description Language (PDDL) or similar to solve the planning problem.

This is done because the semantics used in the SOA domain and the available service do exceed the actions described in PDDL in expressiveness and semantic information. Furthermore, PDDL does not provide any semantics for the domain. Even though there exist extensions of

[33]One-shot planning, means we want to find a possible solution to our problem, before executing it.

PDDL to integrate something like a type hierarchy into PDDL, they lack expressiveness and reasoning support.

Algorithm 20 Service Planner algorithm.

Name: ServicePlan

Input: S_{start}, S_{goal}, Services **Output:** Service Composition

```
 1: $Closed \leftarrow \varnothing$
 2: $Open \leftarrow \{S_{start}\}$
 3: while $s \leftarrow StateSearch.next(Open)$ do
 4:     if $s \notin Closed$ then
 5:         if $s \subseteq S_{goal}$ then
 6:             return $reconstructPath(S_{start},\ S_{goal}, Services)$
 7:         end if
 8:         $grounded \leftarrow ServiceSearch.UsefulServices(s)$
 9:         $succ \leftarrow \{execute(s,\ g)\ \mid\ g \in grounded\}$
10:         $Open \leftarrow (Open \cup succ) \setminus Closed$
11:         $Open \leftarrow Open \setminus s$
12:         $Closed \leftarrow Closed \cup \{s\}$
13:     end if
14: end while
15: return failure
```

In Algorithm 20 the *Closed* list of nodes is the set of already visited nodes. The *Open* set of nodes are possible to visit through a grounded service execution.

The loop in line 1 to 14 searches through all possible state. The search ends if the goal state is reached or a failure is returned because the goal cannot be reached with the given services. The search used is defined in the function *StateSearch.next(Open)*. Depending on the implementation of the state search, the next state to be extended is selected. Here a search mechanism like breadth- or depth-first search can be used. Since in Line 8, the most useful services are selected to create the successors of the open state, the function *StateSearch.next*(*Open*) can select the next state based on this usefulness. The usefulness of a state could, e.g., be described by the overlap it has with the goal state, thus making it a best-first search. The Algorithm 20 leaves this implementation details open. We selected the best-first search as motivated in [93] for our experiment. Line 3 filters all state which could be reached from the current state which has already been visited. If the state has been visited before, it is removed from the open state. Line 4 to 7 check the goal fulfillment. If the goal is fulfilled the needed services are selected from the path in the function *reconstrucPath(path + [s])*. Here we walk, e.g., from the goal state to the start state including all necessary services into the plan.

In line 8 the available services are checked for their executability in the current state. For all executable services a grounding for input, and the precondition of the service is created. This gives us a set of grounded services, consisting of all possible grounded services. In each state *s* that will be extended next, the selection of the services and their grounding is formalized in the function *ServiceSearch.UsefulServices(s)*. Here a set of grounded services[34] is selected, which define the transition to the following open states. Depending on the selection of the services, the search space can be pruned here. If not all possible groundings are returned, e.g., because the

[34]grounded means here that all preconditions are fulfilled, and all input parameters can be satisfied. This means the service is applicable and can be executed.

heuristic rates them as not useful, then they will not be considered in a future search, because *s* will be added to *Closed*.

Line 9 executes the services to generate the next states. The state transition function is given by *execute(s, g)*, where the output and the effect of a service are integrated into the given state *s*. This is a theoretical execution since the execution at runtime includes backtracking and a context sensing mechanism to sense the effect of a service. By theoretical execution, we mean the adding of the effect and output of the service description to the current state.

Line 10 to 12 handle the search space, by removing *s* from the set of open states and adding it to the closed set.

The complexity of the algorithm depends on the implementation of the state search and state pruning mechanism, being the heuristic that selects useful services, including the complexity of the Service Matcher used. In general, the worst case complexity of such an algorithm is exponential [134, p.72].

Our heuristic is then implemented in function *ServiceSearch.UsefulServices(s)*. Here we only rank the executable services due to their heuristic value. Especially no services are pruned from the search space. The search used in the planning problem then selects the service to execute regarding our heuristic in the function *StateSearch.next(Open)* and returns the next state to be extended. Here the service, leading to the next state, is selected due to their heuristic value, thus trying our services with high heuristic values first.

Next, we look at how such a heuristic can be built. This means we need to interpret the service description in the context of the given goal. To this end, depending on the goal, the service descriptions are interpreted differently.

10.5.4. Abstract Approach

Now we can overview the overall structure of the approach used to create heuristics over our artificial representation of meaning (ontologies) in a context-dependent manner. The Methodology used can be structured as shown in Figure 10.16. We will first extract all information possible, e.g., from service and state. We then look into creating the structured knowledge (Decomposition) and how this is used to create semantic similarity towards the goal or unknown concepts (ontology matching).

The ontology matching is done via Marker Passing which integrates connectionist information from the structures knowledge graph created in the decomposition and symbolic information. We then use this semantic distance of words or sentences to estimate how useful a service is regarding a given goal.

Shown in Figure 10.16 we build dynamic heuristics by assessing the similarity between states and services to a given goal. This is done by decomposing the facts making up the state or the action description and using Marker Passing to find a semantic similarity metric between those facts and the facts making up the goal state.

In the next section (Section 10.5.5), we look at the semantics of action, goal and state/context to dived the interpretation into sub problems. Then we will first analyze which contextual information we have available for our heuristic in Section 10.5.6. Then we look at the semantic information available in Section 10.5.7 and describe our heuristic here. We then draw a bottom line in Section 10.7.

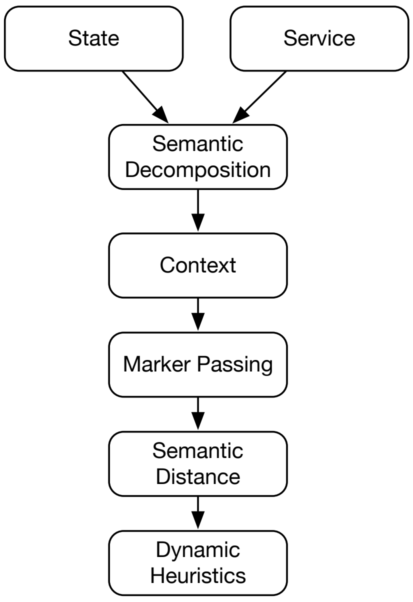


Figure 10.16.: Overview of our methodology to create a heuristic.

10.5.5. Semantics of States, Service and Context

Now we will have a look at the information available in a planning problem. As we have seen in Section 3.3 a planning problem is defined by the services available and the start and goal state. These descriptions contain all information needed to create a solution.

The states are represented as ontologies describing all facts known in the state (see Definition 1). We have looked at the definition of an ontology in Section 3.6 on Page 31. The start and goal state are states with a special meaning since the goal state is something we want to achieve to solve our problem and the start state is the facts the agent knows. In this section, we will look at those separately.

As we have discussed in Section 4.6 our action descriptions are encapsulated in a semantic service description. Here we look at these service descriptions and discuss how they can be interpreted to help solve the planning problem. Last we look at the contextual information of the planning problem.

As depicted in Figure 10.17 the state space is built from a start state by invoking the state transition function to all possible state. The state transition function is defined by the set of services available. This means in each state where the preconditions of a service are fulfilled, and all inputs are available, meaning the service can be grounded and the resulting state is a consistent state, the service is executed and creates a new state with the application of its effect. Thereby a graph is built where nodes are states and edges are service calls. Services in this view of planning, therefore, create or change facts from a state with known facts to goal facts. Since we look at the planning task as a mental process, information services (services without an effect) are excluded from the plan because they do not generate a new state. As a mental process, the planning is hypothetical. For that reason, services execution is simulated by declaring its effect as known facts.

This lets us create a hypothetical graph like depicted in Figure 10.17 where we then select a path from the start state to the goal state. We now look at the four different parts of this view

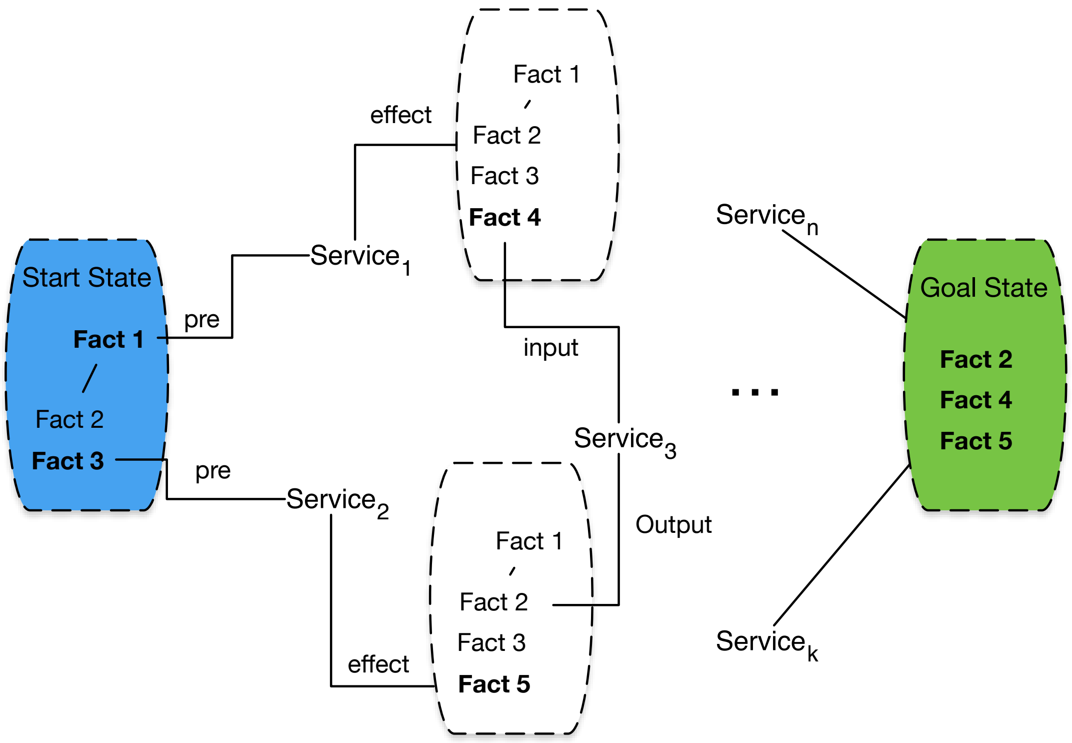


Figure 10.17.: Abstract example of state space in a planning problem.

of the planning problem and analyze their meaning to be able to create fitting heuristics for the service selection part. Then we embed these semantics in the context of the planning process.

Semantic of start state

We formalize a state as a set of facts see Definition 1 on Page 23. The start state declares the facts which are known to be when the planning process starts.

The semantics of the start state is described like every state: as an ontology. Hence it includes abstract facts called T-Box as well as A-Box[35] statements. Both kinds of statements can be changed by a service during the planning. This means that inconsistencies in both (A-Box and T-Box) can occur. Inconsistent state reflecting some world state which is either inconsistent in itself or it is inconsistent with the start state. These inconsistencies have to be handled by the planning algorithm. The facts of the start state, on the other hand, are seen to be true and consistent.

In an agent perspective, the start state can be seen as the facts known to the agent. The plan execution might change something about these facts, or the environment might change which reflects changes in the context of the planning problem which then induces changes in a state. Nevertheless, when starting to plan, some of the facts of the start state have to be seen as fixed. This part of the state is called domain. The domain describes the fact of the planning problem which needs to hold over the whole plan. We can see them as the rules of the game. E.g., while planning to travel, we do not want flights to be booked which arrive before they depart.

[35]T-Box are terminological descriptions of abstract, conceptual statements. In contrast to that, A-Box statements are statements about individuals (linguistic references) which are described by the conceptual T-Box statements. For more detail see Section 3.6

This domain is linked to the start state within import. This import ensures that the facts of this ontology are known but cannot be changed. An example is shown in Listing 10.1.

Listing 10.1: Reference example of ontologies in an OWL syntax.

```
<Ontology xmlns="http://www.w3.org/2002/07/owl#"
    xml:base="http://www.w3.org/2002/07/owl#"
   <Prefix name="j.0" IRI="/ontology/Health-Scallops_Ontology.owl#"/>
   <Import>"/ontology/MedicalTransportCompany_Ontology.owl"</Import>
   <Import>"/ontology/Health-Scallops_Ontology.owl"</Import>
   <Import>"/ontology/FlightCompany_Ontology.owl"</Import>
   <Import>"/ontology/MedicalFlightCompany_Ontology.owl"</Import>
   <Declaration>
      <NamedIndividual abbreviatedIRI="j.0:ArrivalAirport"/>
   </Declaration>
</owl:Ontology>
```

Listing 10.1 shows how, e.g., the SCALLOPS[36] domain is integrated into a state ontology, making all the facts stated in this ontology accessible in the importing ontology.

As an example, an agent could plan how to get to work on Monday, but a state in the state space states that gravity has ceased to exist. In this fictional start state, everything might be described as always, but setting the effect of gravity to $F_G = 0\frac{m}{s^s}$. Now starting to plan for this start state, the heuristic for service selections might need to adapt. With enough conceptual knowledge about the world, one might consider the bus to go to work on a normal day. However, planning for the “0G-Monday” this needs to change. Knowing about the propulsion system of a bus, we can reason that without gravity the friction between bus tires and the ground will stop as soon as the bus hits a bump since it will not fall back to earth. Consequently, all services provided by this kind of vehicles will be neglected in the creation of the plan.

If this is not the case, and the start state contains contradicting facts or inconsistent statements, the service selection becomes probabilistic, since one has to determine which fact is more likely to be true.

To be able to reason upon a consistent start state it needs to state all facts relevant to the planning problem. If facts are not included the contextual information needs to be involved to enabling the reasoning process. In our “0G-Monday” scenario the facts about vehicles and basic physics are included in the domain description, and therefore they are part of the contextual information which is further discussed in Section 10.5.5.

Classical planning approaches describe their state as a mixture of the domain description, the type declarations, causal rules or contextual facts [86]. Since the start state is described in an ontology other ontologies are a reference. This is an easy way to separate the domain description from the start state. This means that we can include an ontology describing the basic physics of the world we plan for, simply by referencing the ontology describing those facts.

In conclusion, we can say that a start state describes the knowledge of an agent or a fictional state for which it wants to plan. The start state needs to include all necessary facts for the planning algorithm to solve the problem efficiently. The less abstract the facts are described [37] in the start state the more efficient the planning can be done.

[36]`http://www.dfki.de/scallops/`, last visited on 29.06.2017

[37]The more abstract class in ontology is the more individuals could be found to instantiate it.

Semantic of a Service

In Section 3.2 we have looked at a definition of service (see definition 2 on page 24). Now we want to describe the semantics of a service: We start out by separating the service from its description. The reason for that is that one service can have multiple descriptions. An example for such a service provider could be a freezer. It provides the service to cool things. The manufacturer of this freezer might provide a service description describing what cooling means and which things can be cooled. Now we imagine this freezer in an environment, where we have to control the grid stability of an electronic grid. Here the freezer can be used to store energy by cooling it down more than normal when energy needs to be consumed and later on, stopping the cooling process, if energy is needed by others. In this why the freezer works like an energy buffer. A new service it can provide, without having to change the freezer or the description by its manufacturer. This new service needs a description, which we can provide if the service is separated from the description.

We have looked at how a service is described in Section 3.2. Now we want to analyze this description on what its parts contain as information for us to use to create a heuristic. We are neglecting informal parts of the service description like quality parameters or the natural language description because they are collected in the contextual information described in Section 10.5.5.

Input: The input describes the parameters we need to invoke the service. The input is given by a set of facts. It also describes the parameters on which the service can base its change in the world upon. This means the input parameters should contain all facts a service will use. For our heuristic, the inputs are those facts which need a grounding to be able to use a certain service. Input parameters can be seen as variables. If the variable is free, meaning not grounded to an individual, then the service does not know how to use this variable. This means all variables of the input need to be replaced by individuals (TBox) for the service to be executed. If we have a flight-booking-service as an example, its input could be start and destination location and a time interval. The description of the input means then that we have to decide on a start and destination location, to be able to use this service. This means a concrete manifestation of these facts needs to be specified. These facts can be part of a state, or we can interpret them as a new sub-goal to be fulfilled by another service.

Output: The output of a service describes the facts it contributes to a state. Here we can describe new or changed facts of a state by the executed service. In our example of the flight-booking-service, the output could be a traveling route. This traveling route might not have been known before the service invocation, and now it is a fact of the state. Hence a new fact was created as the outcome of the service. For our heuristic, we might want to analyze outputs in situations we want new facts in our state. The output of a service needs to be distinct from its effect. The effect might change the state of the world as well. The output is the direct return value of a service, not its side effects.

Precondition: The precondition describes rules about the state which have to be true to invoke a service. Here the same semantics as for the input applies: If a fact, which is used in a precondition has no individual who fulfills this fact in the current state, then the service is not executable. In addition to that, the precondition can describe rules upon those facts, which restrict the grounding of a fact. These rules need to be fulfilled for the service to be semantically sound. In our example of the flight-booking-service one precondition could be that the departure date is before the arrival date in the time interval given as input. Such a precondition allows us to analyze facts like arrival and departure dates and to extract knowledge about those facts. Thus we can extract that arrival might only come after a departure. For our heuristic, we want to analyze preconditions as an extension of our goal regarding the relations between facts if output or effect of the service is useful.

Effect: The effect describes side-effects of a service. These are changes which might not have been described as output because they do not include the creation or change of facts but describe relationships between those facts. In our example of the flight-booking-service one effect concerning the traveling route might be that tickets are still available for each route segment and that some of those tickets are now reserved for us. For our heuristic, we want to analyze an effect if new relations between facts are wanted.

Each part of the IOPE description has its own semantic about the described service. To separate this specific purpose of each description part multiple conditions need to hold for the implementation. For the input to have the semantics described above, the service needs to declare all facts, which it bases its calculation upon as input. In consequence, the precondition should describe the relations between those facts extensively. The same should hold for output and effect. Also, all effects should be formalized in the effect of the service.

Also a service in OWL-S contains a natural language description and a name. Those can be used as additional contextual information by the creation of a heuristic. This contributes to the need for a semantic distance measure for words and sentences.

Here we have analyzed the four formal parts of a service description, for their semantics regarding their role in creating a heuristic.

Semantic of goals state

The goal state describes a set of states of the world which we want to achieve with our planning approach. The goal is formalized as a set of facts. A state fulfills a goal state if the facts of a state, subsume the facts described in a goal state. An example could be, that if the goal state ”IsAt(?x, Berlin)” stating that something should be in Berlin, the state ”IsAt(Johannes, Berlin)” fulfills the goal.

This means we restrict our goal to goals which describe facts to achieve. These goals are sometimes called **achievement goals**. It can be argued that maintenance goals [64] are different because we want to maintain the state described in the goal.

The facts described in the goal can be subject to our semantic analysis. This analysis leads to conclusions regarding the semantic relations of the facts in the goal with, e.g., facts from the effect of a service. If for example some fact is described by a concept which could be described with the word “open” and can deduce that the antonym, e.g. “closed” might be some unwanted fact.

Like in every state (e.g., external changes in the context) the facts of the goal state might change as well. We assume the goal to be fixed throughout the planning process. If the goal changes at runtime a new planning process can be triggered. To change the current state of the world to the desired one, change has to occur. This change can be proactively or autonomously influenced by services the agent can execute.

For the heuristic the goal describes wanted facts. Particularly, it defines what the problem describes as “good”. In our flight-booking-service example, the goal might define that we want to be in Berlin in two weeks. The fact “being in Berlin” therefore might be seen as wanted in contrast to being somewhere else. The information from the goal affects the selection of parameters for services or the selection of services itself. However, this representation entails a few challenges:

Axioms If an effect is described on not grounded facts, the goal can be fulfilled by multiple groundings of this fact.

Individuals in goals An individual differs from a general ABox fact since it describes one concrete entity TBox. In our flight-booking-Service example, a generic goal might be to bring someone to Berlin. However, if the goal is to bring the author of this work to Berlin, then we need to specify this individual to book the flight for. As a result, the goal does not only consist of abstract facts (Abox) but also on individuals. This calls for an extension of the reasoning for individuals, meaning that the grounding of certain rules (e.g., an effect of some service) needs to create individuals. Individuals cannot be created in an effect because the SWRL language does support such an extension only in an extension (SWRLB) and the Pellet reasoner does not support them.

This leaves us with the goal as the main guidance of the heuristic. Besides the domain description which defines the “rules of the game”, the goal forms the heuristics and the interpretation of the services. This includes especially which facts are wanted (meaning which services are useful) and which facts are not wanted (meaning which services should be avoided).

Semantic of context

In Section 3.8 we have defined in Definition 20 on Page 35 what we see as context. This context definition is now applied to our problem of using service description and a planning problem to create a heuristic. Therefore we will now collect the information which represents this contextual information and analyzes its semantics.

First of all, a context is formalized like a state as a set of facts (see Section 3.8 for more details). These facts include domain descriptions taken from the planning problem, description parts like the natural language description of a service or Quality of Service parameters. This contextual information is not problem specific. This means in the same context; multiple planning problems can occur as well as one planning problem can be seen in different contexts. This includes the conception of a service from Section 3.8, where the agent using the service has his conception of the service and creates its context. In our flight-booking-service example, the context in which the flight might be booked might be spare time and with that count as vacation or it might be a business trip and count as work. In consequence, depending on this contextual information, the credit card to pay for the flights might differ. This means our interpretation of “the right credit card” to be used to pay for the flight, might differ.

We generalize this observation: The contextual information influences how we interpret the service description. With other words, depending on the context, the words making up the description can be interpreted in different ways, if those words have multiple word senses. The semantics of the context will be used by our approach to select the right word senses from a description, the states or whatever resources we are analyzing. Indeed, the context might change the meaning of the facts analyzed (to become pragmatic) and with that influences how we create a heuristic.

10.5.6. Using Pragmatic Meaning

This section describes how the beliefs and the goal of an agent are used to set the context for creating a heuristic when evaluating service descriptions for service planning.

To be able to build a pragmatic heuristic from a goal state to evaluate service description for their usefulness we need to distinguish concepts used in the description of the goal and the service. Since most likely the description of the goal and the description of the service have been made separately, it may be that the service description formulates similar things than the goal, but uses other words. This raises the need of comparing two words and establishing a semantic similarity. Further, some of the used words to describe the service might have different meanings in different contexts. This leads us to the problem of having to disambiguate word senses in a service description.

Therefore, we started to build a heuristic by creating a semantic similarity measure (see Section 10.1), to be able to decide how close two concepts are. We extended the analysis of the meaning of single words to sentences (see Section 10.3. Furthermore, we analyzed the different parts of a service description (see Section 10.4). Using those four tools, we then analyze the concepts of a goal state and the concepts describing a service, to elaborate if the service might help to reach the goal.

There are many more linguistic tools like coreference analysis which could help understand the service description. However, since those are syntactic, we leave their analysis to future work.

Let's start out by looking at which information is available for us to build our heuristic upon. In classical planning problems the domain description, a start and goal state and a list of available services are given. We have discussed the planning problem closer in Section 3.3. Since we are looking at service planning, we postulate that the services, the states, and the domain are described using OWL and OWL-S and SWRL, like we have described in Section 10.5.5.

The different parts of information contain different aspects of the problem. The domain of the problem might, for example, describe the rules of the world the problem is located in. Like the goal state, which defines what is good and what is bad in the context of the planning problem; this information can be used to decide how to interpret the service descriptions. Let's look at an example where we have a planning problem in which the goal state is to get money. In this example the word Bank might be interpreted as a place where money is stored, services giving information about a $bank_{geological}$[38] will most likely not be of interest.

The states and the domain description consequently make up the contextual information which we can use to create an interpretation of the service descriptions.

Building contextual meaning from the service description now means given the contextual information; we want to be able to discover those services most likely to be executable and useful for the goal. This means the heuristic is calculated on grounded services. Grounded services are used since there is more contextual information available. Imagine a flight-booking-service, where the grounded effect states that "Johannes is in Berlin" which fulfills our goal, but

[38]Here the subscript differentiates the word sense. Here as in edge, shore, rim or slope.

if someone else is in Berlin, this does not. The service description is used with the following course of action:

1. Decompose each new service description output and effect
2. Merge this decomposition with the decomposition of the goal state.
3. Decompose each new service description input and precondition
4. Merge this decomposition with the decomposition of the current state.
5. Using our Marker Passing to measure the amount of fulfilled goals by the effect and measuring a number of preconditions which are fulfilled.
6. The heuristic is built by a weighted sum of the "usefulness for the goal" (amount of subgoals fulfilled) and "executability" (amount of preconditions fulfilled) to create a heuristic for the service.

This approach might vary, depending on the kind of heuristic we are looking for. This means that if we are using a backward search, we probably want to start the Marker Passing from the goal state. If we are looking for a heuristic for a FF-planner, we want to start markers from the current state we are in. We selected a forward search during our plan. The approach suggested here can be modified to be used with other planning algorithms. In the next section, we will look at the details on how this heuristic is built for a forward search.

10.5.7. Building Context-Dependent Heuristics

We have motivated our work by the need of a heuristic for AI planning. Since the search space of domain independent planners for large problems becomes computationally intractable [187] we need heuristics to guide our search through the state space. The heuristics are needed because we cannot search the whole search space; therefore the heuristic narrows the search space to a required part we need to search to find the goal state. We have defined heuristics in Section 10.5.1. The interested reader is referred to [304] for a formal description of heuristics and their properties. During our analysis of this topic (see Section 10.5.2 and Section 10.5.3) we want to create an heuristic which is *goal dependence*, *service interpretation* and *State/Context awareness*.

We have found that understanding the described functionality of a service is AI-hard task [419]. The interpretation of what a description creator might have understood the service to be is not entirely reflected in the description. Furthermore, the service can have multiple interpretations in different contexts. The context we defined is the additional information relevant to our problem. As an example strategy for problem solving, we have selected planning. This means our context consists of the start and goal state which includes a domain description. With this we are able to extend the signature of a heuristic from the classical one $H : State \rightarrow Double$ to:

$$H : State \times State \times Service \rightarrow \mathbb{R} \tag{10.20}$$

In this work a heuristic is defined as a function H which estimates the usefulness of a service $a \in A$ for a goal state $s_{goal} \in S$, with the contextual information of a current state s. Our heuristic function H is calculated upon the description of the service a and the description of the current and goal state of the planning problem. How we build this heuristic in detail, using the available semantic information, will be subject to the next sections.

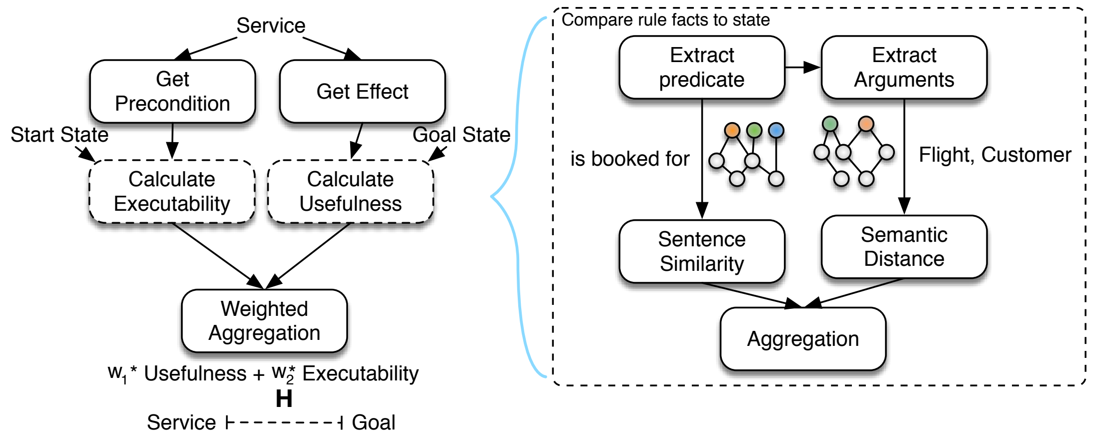


Figure 10.18.: Abstract approach to a greedy A* heuristic.

To create a heuristic which is adapted to the problem at hand and can be created at runtime, we analyze which information is available, and how it can help to create a heuristic. From the planning problem, we have the start-, and goal-state, as well as all services given as information sources. As we have discussed in Section 10.5.5 the start state contains the domain description as part of the context from which the planner needs to start from. Furthermore, the semantics of the service describes how these facts are the wanted state the search in the planning process should reach. The description of the service is a basis for the decision if the functionality encapsulated in the service is helpful for the goal. Additionally, we have discussed the semantic of a service description are further.

This now leads to the discussion on which part of the service description should be part of the heuristic? In principle, every part of the description containing helpful information should be used. We have discussed the service description and its parts, and they're semantic, now we look at how we use those parts in our heuristic:

Input: The input is compared with the current state in the state space search to determine which service can be executed. This is done by looking at the facts of the state and deciding if the input parameters of the service can be fulfilled.

Output: The output is compared with the goal, checking if some facts of the goal are fulfilled or if some fact contradicts the goal.

Precondition: The precondition is used as the input to check whether it is more likely that the service can be executed.

Effect: The effect is used as the precondition but in comparison with the goal to determine to which degree a service fulfills facts from the goal.

In addition to these parts of the IOPE service description, the facts of the state like start and goal states are analyzed as well. Each of the facts described has three components which can be analyzed:

Predicate comparison is done with the sentence similarity measure d_{sen} proposed in Section 10.3. This is because a predicate mostly describes verbs and their form, direction and if they are passive or active. Our example "is booked for" is a typical use case for a predicate in ontologies.

Argument comparison is done with the semantic similarity measure d_{sem} proposed in Section 10.1. Here the arguments are compared, and the maximum is summed up. That makes the argument comparison independent of argument order.

Type comparison is done through the decomposition since all predicates are decomposed and with that generalizations and specializations are found as well, which will then be merged in the resulting graph. This might lead to markers passing over those edges and activating facts with similar types.

This analysis is special for two parts: The precondition and effect because they are formulated in SWRL. This means the predicates are additionally concatenated with SWRL predicates.

Our approach for the heuristics as described in Algorithm 21 now takes those parts and decomposes them. The parts which should be compared are then put in merged graphs, e.g., the effect and the goal graphs. Then we use an appropriate Marker Passing to compare the semantic distance from, e.g., the effect predicates to the goal predicates. The heuristic is implemented in the following way:

In Algorithm 21 line 1 to 6 calculate the overall heuristic value of the given service. Here the weights w_1 and w_2 describe how important the two components of the heuristic are. Those weights can be changed over time with the progression of the heuristic to the goal state. We held the weights static at 0.5. Line 8 to 12 extract the predicates from the given state and stores the predicate with its arguments in $predicates_{goal}$. Those are the subgoals we want to achieve. Now we compare each predicate of an effect of a service to those goals and sum up how much sub goals are achieved. The function *getArguments* gets the name of the argument, where the *getPredicat* function gets the type of the argument.

Line 13 to 27 then extract the effects or precondition, its predicate and arguments to compare them to the facts in the goal or start state and select the best matches. The outer loop (line 13 to 26) selects each fact of the goal and lets as compare the effects to those facts. The loop from line 15 to 24 compared the facts of the goal to the effect of the service. This is done by comparing the predicates (line 17) and the parameters (line 19). Line 21 weights the parameter and predicates similarity. In lines 22 to 24, we then remember the best match result, which then is summed up in line 25, to form the overall matches to the goal facts so that the service is evaluated to how much of the goal it fulfills.

Line 27 returns and normalizes this result by a number of predicates in the goal or start state. This means that if all predicates are fulfilled with an exact match, the heuristic becomes one.

Line 29 to 30 implement a helper function which returns the sum of the normalized maximum matches between effects and goal.

This argumentation leads us to introduce two weights w_{pred} and w_{param} which can be adapted depending on how far the search has progressed towards the goal. In the beginning, the executability should, in consequence, be highly weighted and become less important the closer to the goal the search progresses. This is inverse for the usefulness.

Algorithm 21 Goal oriented heuristic.

Name: GoalHeuristic
Input: State S_{goal}, State S_{start} Service *Service*
Output: double

```
1:  double result = 0;
2:  List<Rule> effects = GETEFFECT(Service)
3:  usefulness = DISTANCETOSTATE($S_{goal}$, effects)
4:  List<Rule> precondition = GETPRECONDITION(Service)
5:  executability = DISTANCETOSTATE( $S_{start}$, precondition)
6:  return $(w_1 * usefulness + w_2 * executability)$
7:  function DISTANCETOSTATE(State S, List<Rule> rules )
8:      Map<String, List<String>> $predicates_S$ = ∅
9:      for $z_i \in S$ do
10:         List<String> $arguments_{z_i}$ = GETARGUMENTS($z_i$)
11:         $predicates_S$.put(GETPREDICATE($z_i$),$arguments_{z_i}$)
12:     end for
13:     for $p_S \in predicates_S$ do
14:         best = 0
15:         for $e_i \in rules$ do
16:             $p_{e_i}$ = GETPREDICATE($e_i$)
17:             $match_{pred} = d_{sentence}(p_{e_i}, p_S)$
18:             List<String> $argument_{e_i}$ = GETARGUMENTS($e_i$)
19:             double $match_{param}$ = SUMMAX($argument_{e_i}$, $predicates_S.get(p_S)$)
20:             $double_{match} = w_{pred} * match_{pred} + w_{param} * match_{param}$
21:             if $double_{match} \geq$ best then
22:                 best = $double_{match}$
23:             end if
24:         end for
25:         $result += best$
26:     end for
27:     return $\frac{result}{|predicates_S|}$
28: end function
29: function SUMMAX(List<String> eff,List<String> goal)
30:     return $\frac{\sum_{j=1}^{|goal|} \max_{i=1\ldots|eff|} (dist_{semantic}(eff.get(i), goal.get(j)))}{|goal|}$
31: end function
```

Both parts use the same kind of mechanism to check rather a fact is fulfilled (in the precondition or effect of a service) which is given by the goal or start state. To check this fulfillment, we extract the predicates from the service precondition (effects) and the start (goal) axioms and their arguments and compare them. The comparison is made in two ways: first, for the predicates name, we separate the word included in the predicate, e.g. “IsBookedFor(Flight x, Customer c)” become the predicate “is booked for”. Since this resembles a sentence, the sentence distance measure d_{sen} is used to compare predicates (see Section 10.3 on Page 189). Second we compare the arguments with our semantic similarity measure d_{sem} (see Section 10.1 on Page 157). The result of those both similarities is then aggregated.

The interpretation of the marker resulting from the Marker Passing is done like described in Equation 10.21 which describes how we calculate the heuristic for a service S:

$$H(Start, Goal, S) = w_1 * Usefulness(S, Goal) + w_2 * Executability(S, Start) \tag{10.21}$$

$$Usefulness(S, Goal) = \\ = \frac{\sum\limits_{g \in Goal} \max\limits_{e \in S.effct} \left(w_{prd} * d_{sen}(n(e), n(g)) + w_{arg} * \left(\frac{\sum\limits_{arg_g \in Args(g)} \max\limits_{arg_e \in Args(e)} d_{sem}(arg_g, arg_e)}{|Args(g)|} \right) \right)}{\mid Goal \mid} \tag{10.22}$$

The name of the axiom denoted with $n(\bullet)$. Here the comparison of an axiom is threefold as described above: The predicate comparison, the Argument comparison, and the Type comparison. In OWL typing is described by, e.g., class assertion predicates. That is why in the Equation 10.22 we have combined all predicate comparison including type comparison into one part. This means that we include the comparison of all three parts of an axiom. A similar equation to Equation 10.22 can be formulated for the Executability where the normalization is done over a number of preconditions. This means that if the sum of w_1 and w_1 equals to one, the overall heuristic has values between zero and one.

We selected this two measures for our heuristic, because if we leave out one of the aspects two effects happen:

Goal overcommitment If we only look at the usefulness the services fulfilling subgoals will be tried first. Even though the probability of them being on the end of the plan is higher. This means we are not talking about a planning problem which is trivial because all services in the plan are independent. This means that one or more services need to be executed to enable a useful service. By only looking at the usefulness the search will always try services we can not yet execute, first.

Low hanging fruits If we only look at the executability, services which are executable are always tried first. This is good at the beginning of the planning process because reaching the goal is less probable at this point. However, the more services are executed, the more service preconditions have the chance of enablement, and all of them are tried first. The search then becomes like a breadth-first-search, where most services are tried before we get closer to the goal.

Now we analyze the different properties of our heuristic. We have discussed at the beginning of Section 10.5.3 that we do not have an optimization problem. That does not mean that this heuristic can not be used for optimization problems as well. Depending on the search mechanism used during the planning, we can have different properties of the outcome. This is why we restrict our analysis to sub classes of the best-first-search.

The A* search algorithm is thought for optimization planning, where each service s executed has a specific cost. We have discussed earlier that our planning problem is not an optimization problem. This means we do not have a service cost. In consequence, we need to adapt the A* algorithm. The modification is done by selecting the different function $g(s)$ for the optimization function $f(s) = g(s) + (1 - H(Start, Goal, s))$ used. By selecting an appropriate $g(s)$, our planning problem can become like an optimizing problem, since then the cost of the path to the current state can be estimated. In our problem, where we do not know the cost to the current state, we analyze the following options:

$g(s) = 0$: makes the search only rely on heuristic and as a result is a best-first-search.

$g(s) = \mid P \mid$: with P being the shortest known path to state s. This is equivalent to a cost of one for each service call, in this case, the A* optimizes the path length which does not have to be optimal in cases where service might fulfill many subgoals but are more expensive.

$g(s) = \sum_{a \in P}(\mid Precondition(a) \mid)$: with P being the shortest known path to state s. Where we look at the sum of needed preconditions of services leading to state s.

The first option neglects the service cost and thus behaves like a pure heuristic search. The pure heuristic search is greedy because the services with the best heuristic value are used first. The second option has a constant service cost of one. In this case, the amount of service is minimized. The third option is an educated guess for the selection of g. Here any function could have been selected. The third option punishes the execution of services with a precondition; thus services with more preconditions are punished as well. During the path to the goal, this $g(s)$ increases, since more service have been executed. With that the number of services executed is minimized as well, but also services with fewer preconditions to fulfill are preferred. Because of its better performance, the third option for g has been selected for the rest of the experiments.

In conclusion, the heuristic produces one value which states how executable and useful a service call is. Figure 10.19 shows an example of how a state space might look like during planning. Here the green state is the start state, and the red state is the goal state. Black states are states to which we already know a heuristic value and gray states are not yet known. Black states are possible next state (sometimes called **open states**) in the search. Unknown states are states which can not be reached with a single service call. The blue state represents the current state a search algorithm has expanded. In Figure 10.19 shows states which have been visited but do not reach a gaol as crossed out states. Those state are sometimes called **closed states**.

Now we need an example problem to test our heuristic upon. The data set used for our evaluation will be discussed next.

10.5.8. Data Sets

There is one data set which uses semantic service descriptions for service composition using OWL-S as service description language. This data set is called **Secure Agent-Based Pervasive**

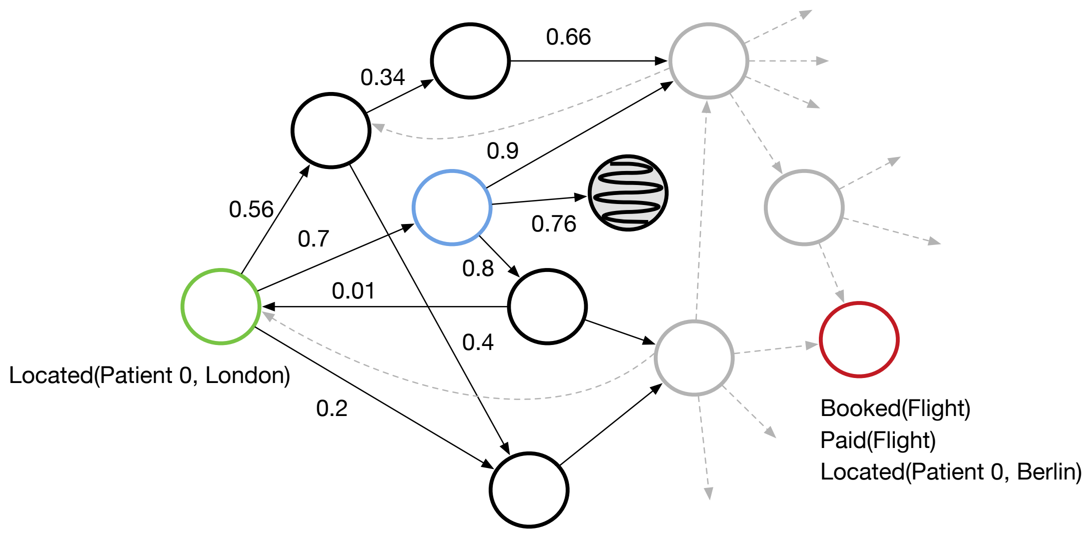


Figure 10.19.: Example state space with heuristic values.

Computing (Scallop) domain[39]. It has a collection of 21 services in the domain of health, air travel and medical transport. It includes a set of 24 ontologies building up the domain model. Here the scenario in focus is the medical transport of victims to a medical facility mostly by air plain and some ground transport.

Here the problem to be solved is to transport a patient from one place to a hospital. This includes finding the nearest airport from which one can fly to an airport which is close to a hospital. Also, a means of transport from and to the airport has to be organized and booked. To book a flight a flight account has to be created. A fitting flight has to be found, and the flight needs to be booked. Having done this, the goal of our example problem is reached.

We created a start and end state of this domain in which a victim has to be transported to a medical destination. The goal state consists of 64 axiomatic facts which need to be fulfilled to reach the goal state.

The initial state has declared multiple facts about the domain. We start out with Figure 10.20 which describes the transports available in our domain. Here we see that Vehicle Transports and Flights are the two possible means of transport.

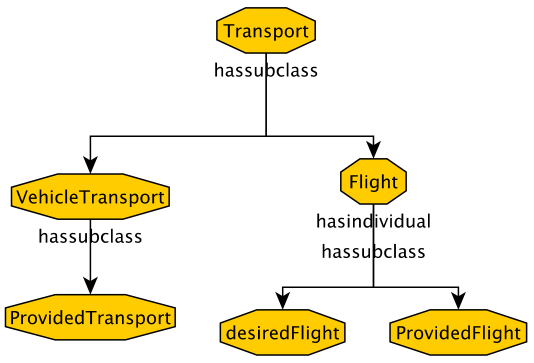


Figure 10.20.: The transport part of the Scallop health domain start state.

[39] `http://www.dfki.de/scallops/` Funded in the publicly funded project: BMBF 01-IW-D02-SCALLOPS. Last visited on 10.09.2017

In our example scenario, we need to create an account to be able to book a flight. This account requires a credit card to purchase a flight. An account having a credit card is a validated Account. Figure 10.21 shows how an account is modeled. An account has the subclass of a valid account and the individual of the desired account.

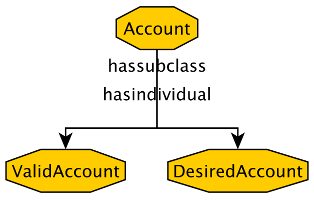


Figure 10.21.: The account part of the Scallop health domain start state.

Figure 10.22 describes three parts which are interlinked: The credit card, a location and a person. An example fact here is that a credit card is owned by an EMAWorker who is a person. Further, this ontology describes the destination hospital and the departure hospital which is instantiated in the goal state with an individual stating to which hospital we want to transfer the patient to.

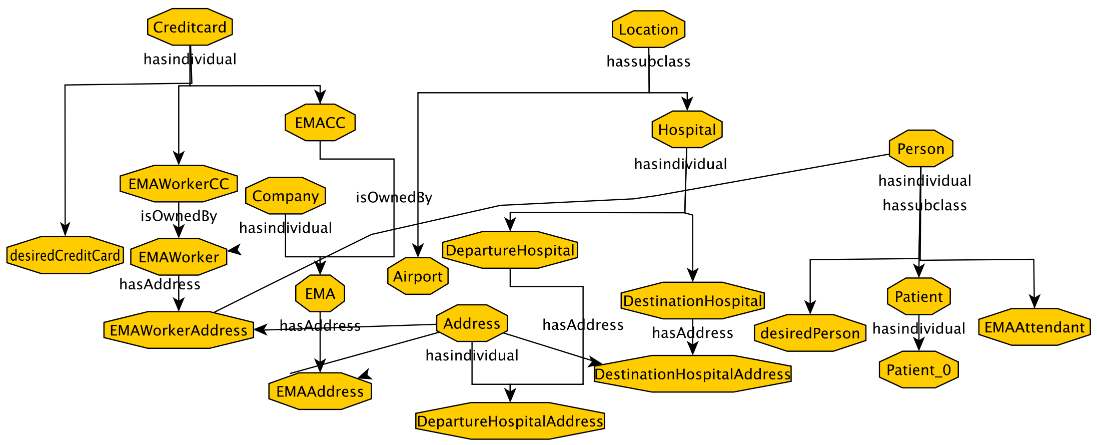


Figure 10.22.: The location part of the Scallop health domain start state.

Figure 10.23 describes the parameters a transport can have. These parameters describe how the transports can be configured and make up a big part of the individuals which need to be created to parametrize our plan. Here we have for example Time described in dates, where there can be an infinite amount of individuals.

The overall domain is additionally described in an ontology. This ontology describes the individuals and their relations. This ontology is too big to be displayed as a figure here. However, the interested reader can download this ontology from the project web site[39].

Figure 10.24 depicts the facts about the patient in the goal state. The patient which should be transported is modeled in the individual “patient 0”. The flight, the departure and destination airport individuals are called “Mustermann” which is a German equivalent to “Johan Doe” for an unknown name. Thus our ”patient 0” is “Mustermann”. The creation of individuals has to be done since we need to create those individuals beforehand because SWRLB in combination with the Pellet reasoner can not create individuals at runtime.

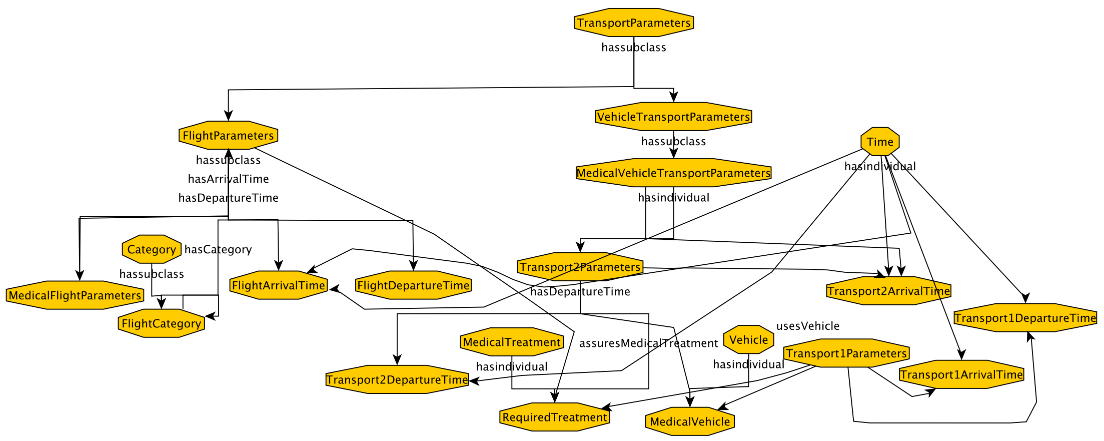


Figure 10.23.: The parameters part of the Scallop health domain start state.

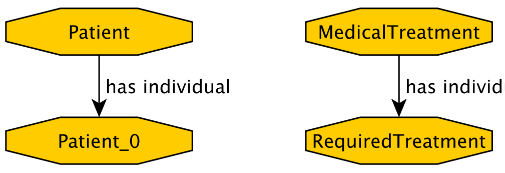


Figure 10.24.: The patient and transport part of the Scallop health domain goal state.

Next, we model the individuals who are needed to reach our destination. In Figure 10.25 we model the departure and arrival airport.

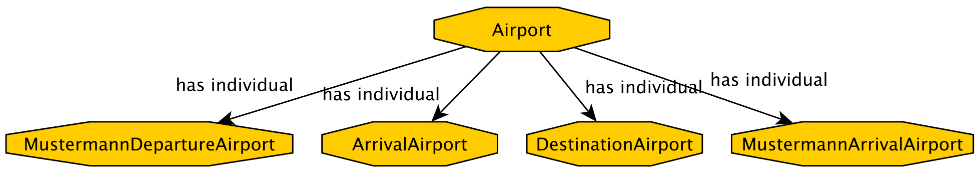


Figure 10.25.: The airport part of the Scallop health domain goal state.

Since we need a valid account to book a flight, we model our account we need in our plan in Figure 10.26. Here we want that our patient “Max Mustermann” has a personal account to transport our patient zero and book the relevant flights and transports from and to the airport.

There are many ways to model such a fact. In Figure 10.26 we have modeled the fact that we want a valid account as sub class relations. In consequence, our account must be of type “ValidAccount”.

Next, we model the more complex parameters the flight has to fulfill in Figure 10.27. Here we state that we wish the flight to be booked for “Max Mustermann”, who owns the credit card the flight can be booked with and that the flight arrives at our goal destination at our goal time.

Figure 10.27 is a way of specifying that we want to be at a certain location at a certain time. All the modeling around those facts is necessary because we have to make sure all individual are available so that we can evaluate a potential execution of a service and with that reason upon the effect of this service.

When starting to plan, the start and goal state already overlap with 57 axioms. The goal state consists of 64 axioms. Consequently, the plan to be made has to fulfill seven axioms to reach

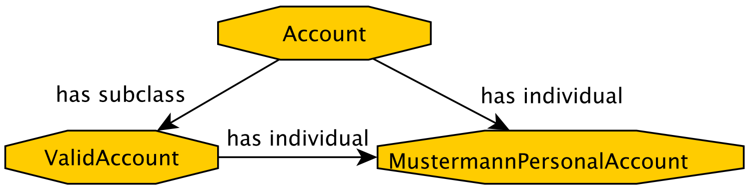


Figure 10.26.: The account part of the Scallop health domain goal state.

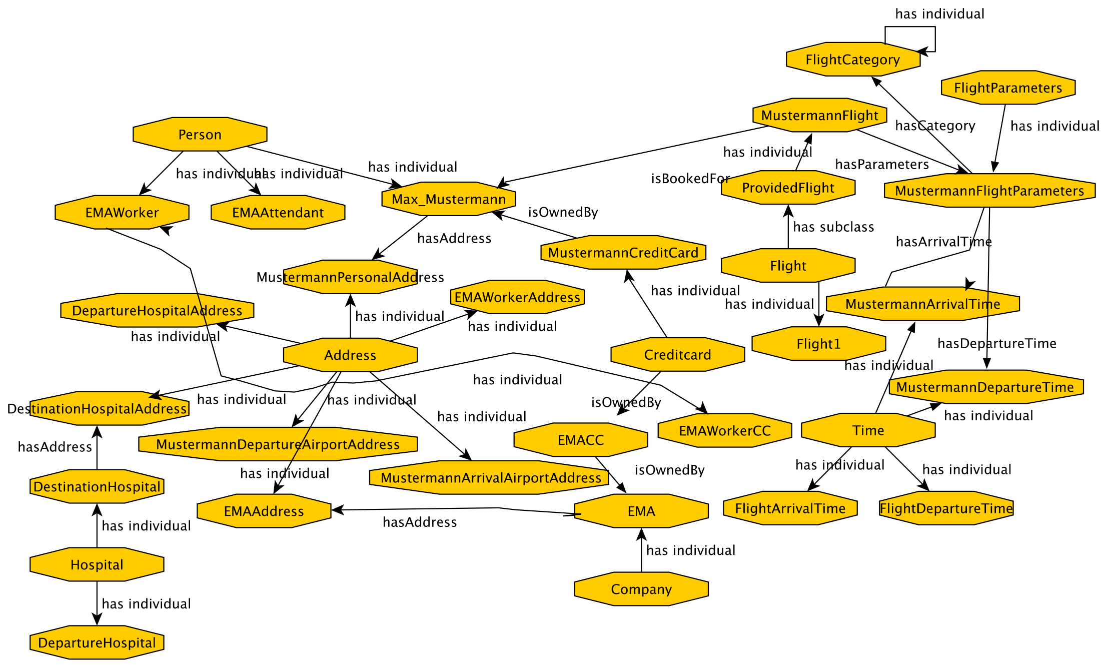


Figure 10.27.: The parameter and person part of the Scallop health domain goal state.

the goal state. These axioms are described in Figure 10.28. The optimal plan for this problem includes four steps: two requests for flight information for departure and arrival time, creating a flight account and booking the flights.

The planning problem, with its 21 services which have continuous inputs like dates, has an infinite search space. But since we do need to describe the inputs as individuals, the search space is artificially reduced to about 54 valid and reachable states.

The selection of the data set can be criticized because it is a single example of such a planning problem the SCALLOPS domain described in Section 10.5.8. The planning problem has been selected because it is prototypical for real world example of services descriptions and the description of a domain. The service description has been made in an independent project, and they are not fit to our experiment.

The next section will elaborate on the results of our planner with the different heuristics.

10.5.9. Evaluation Results

The evaluation is run using this data set using the different heuristics. To measure the performance, we count the extended nodes during the search for the goal state. The results of our experiment are listed in Table 10.13. The column avg$_{steps}$ indicates how many states the search had to extend to find the goal state. The column avg$_{time}$ describes how much time one search run

hasParameters(MustermannFlight,MustermannFlightParameters) ∧
isBookedFor(MustermannFlight,Max_Mustermann) ∧
hasArrivalTime(MustermannFlightParameters,MustermannArrivalTime) ∧
hasCategory(MustermannFlightParameters,FlightCategory) ∧
hasDepartureTime(MustermannFlightParameters,MustermannDepartureTime) ∧
ValidAccount(MustermannPersonalAccount) ∧
ProvidedFlight(MustermannFlight) ∧

Figure 10.28.: Example goal axioms which are different in our goal state to our start state.

to has taken on average in seconds. The average has been created over ten runs[40]. All heuristics are tested on the same start and goal state. Hence there is no difference in the planning problem state space the search has to traverse.

From the state-of-the-art analysis for general purpose heuristics (described in Section 10.5.1) the greedy heuristic [304], which checks the overlap of the effects of a service, with the facts wanted in the goal state, is performing well on this data set regarding the time consumption. An additional comparison is the Uniform Cost Distribution, where each execution of a service is calculated to cost one abstract cost measure. With that admissible heuristic, A* finds an optimal solution. The random heuristic has been added to the comparison so that we can evaluate how far from the base line we are with our approach.

Table 10.13 shows the results of the average planner performance of ten runs. The variation of the performance is rooted in the fact that there are different ways to achieve the goal, depending on which service is used first. If two services have the same heuristic value[41], one of them is selected randomly. Table 10.13 shows the mean μ and standard derivation σ of the experiment results regarding a number of steps and execution time.

Table 10.13.: Planning results for an average over ten runs of planner (time is in seconds).

Heuristic	μ_{time}	σ_{time}	μ_{step}	σ_{step}
MP	**120**	31	**20.3**	5.2
Greedy	292	51.9	43.3	6
Uniform Cost	325.4	27.5	51.4	2.6
Random	358.9	9.7	53.9	0.32

Concerning the overall result, our MP approach is at least twice as fast and looks at half of the nodes the other heuristics do. This reduction of the search space has to be emphasized because this shows that less than half of the nodes are extended during the search. Now we will discuss the rest of the results and analyze what they mean.

10.5.10. Discussion of the results

We start by comparing the different heuristics:

[40]We have tried the random experiment with 100 runs, which did not yield any different result, which leads to the conclusion to save the time of running all experiments over 100 runs.

[41]Two services having the same heuristic value seems unlikely, but the data set shows services which are almost copies of each other, thus making this likelihood bigger.

The greedy heuristic is the golden standard for general purpose planning. It is used in a best first search like in the successful Fast-Forward Planners [171]. As the name suggests, the services with the most overlap with the goal (the "best" once) are tried first. This leads to the result that in each state of the state space, the same "best" service is tried over and over again. The evaluation of the goal overlap does take almost as much time as the semantic heuristic. This can be seen by comparing how many steps on average are looked at, here: 42, and how much time is spent. Here we have an average step calculation time of 6.74 seconds which is close to the 5.91 seconds the Marker Passing heuristic spends on each state. The standard derivation from the mean can be explained by the random selection of a service with equal usefulness.

The Uniform Cost function which seems useful for all services creates a breadth first search. Here we can see that the standard derivation of the steps reduces to 2.6. This is because we are looking for all service in one state before progressing to the next, it does not matter in which order we look through the services. The uniform cost heuristic spends an average of 6.33 seconds on a state.

The Random cost heuristic is the baseline.. If we are worse than this, our heuristic creates more confusion; then it guides the search. Also, this is used to ground the overall speed of our heuristics. This can be done because the creation of a random number does almost consume as few resources as the Uniform Cost heuristic and does not give any information to the search. Here the standard derivation of steps is less than one because the random heuristic almost alway searches the entire search space. Of course, the random heuristic has to look at the most states in the search space (see Table 10.13) which is at an average of 53.9. The random heuristic spends an average of 6.66 seconds on each state.

The difference in average time spent on a state is not because of the complexity of the heuristic calculation. An average state visited of 53.9 for the random heuristic seems unintuitive. This intuition is why we have repeated the experiment with 100 more runs, with the result of an average sept count of 52.71 and a standard derivation of 2.28. Which seems more plausible and the result with ten runs can be explained by the arbitrary selection of 10 bad heuristic samples. The Random heuristic has a height time consumption for states because it looks at more states deeper into the search space. Since our algorithm represents the state space by keeping the deltas between the states, the collapse of a state to reason, e.g., its consistency takes time. If we collapse states further from the start state, we need more time. This explains why a heuristic extending only a few states before arriving at the goal spends less time in average on a state.

Of course the relation between looking at more states and taking more time correlate. Since the evaluation of the precondition and the instantiation of the effect take time, and with that, a uniform cost heuristic (which has no cost to calculate the heuristic) is still ineffective because each service executed takes some time. The semantic use in the Marker Passing heuristic reduces the standard derivation from the mean, which means that we gather more useful information then it is done in just comparing the overlap with the goal. The standard derivation from the mean can be justified with the random selection of services with the same heuristic value. In addition, the services are requested from a web server, and the order in which the services are returned is random, which means that the heuristic is still not precise enough. This might change if grounded actions would be analyzed.

This concludes our experiments. Next, we describe the experimental setup, to enable reproducibility of the experiments.

Now having laid out our our evaluation, we describe the experimental setup with which the experiments have been conducted.

10.6. Experimental Setup

For reproducibility and scale, we will describe the experimental setup. This is done by listing all factors known to us which could influence the result of the outcome of our experiment. All here conducted experiments are done in the same experimental setup. Table 10.14 describes the used hardware and Table 10.15 describes the external software resources used with their version.

Table 10.14.: Hardware description of the computer used for the experiments.

Part	Type
Model Identifier	MacBookPro11,3
CPU	Intel Core i7
CPU Speed	2,5 GHz
RAM	16 GB
SSD	512 GB
Graphic	NVIDIA GeForce GT 750M

Table 10.15.: Software description main external components used.

Software	Version or date of download
Docker	1.12.1
WordNet	3.1
Deeplearning4j	0.4-rc3.11-SNAPSHOT
CoreNLP	3.5.0
Wiktionary Dump	28.03.2016
BabelNet	2.5.1
Wikidata JSON Dump	06.10.2016

The downloads of the dictionaries Wiktionary Dump[42] and Wikidata JSON Dump[43] where the newest relates at the given dates.

We will now take a step back and look at our results with some mental distance. For that, the next section will critically analyze the drawbacks of our approach.

10.7. Evaluation Results

In conclusion, we can say that we have tested our approach on different example problems. Even though we have not used the whole potential of the Marker Passing approach, we could

[42] `https://dumps.wikimedia.org/`, last visited on 28.03.2016
[43] `https://dumps.wikimedia.org/wikidatawiki/entities`, last visited on 06.10.2016

reach the state-of-the-art in all our experiments. Creating a semantic graph out of a multitude of data sources and using the graph with our Marker Passing approach to reason upon this graph has shown to be useful in an application like WSD, Semantic Similarity, Sentence Similarity, Service Matching and Heuristic Planning. Combining the here gained experience we were able to create a goal oriented context-dependent heuristic, which outperforms other general planning heuristics.

All those experiments show that our automatically generated semantic graph in combination with an appropriate Marker Passing represents some kind of meaning representation. We did not include our experiments with common sense reasoning into this thesis. To measure our approach on the task of Common Sense Reasoning (Winograd Schema Challenge[44]) we implemented a Winograd Schema resolution algorithm extending our decomposition with semantic edges. With that we were able to outperform the state-of-the-art by five percent.

Overall the results of our approach are positive and its potential is not yet reached. Figure 10.29 shows the percentage of performance gain in the different experiments compared to the best of the state-of-the-art. So there are many more questions open concerning our approach and many challenges to tackle with our approach like semantic entailment or generalization.

The restriction on problems in the Natural Language Processing domain is a result of the focus of this work. Extending this narrow application domain with additional symbolic information on markers, new in-, out,- node-, and edge-functions as well as with new data sources is work in progress.

One drawback of this experiments is its data dependency. For the here shown example approximately 90 GB of data sources have been used. The choice of decomposition depth have been limited to three because the 16GB of memory are not sufficient for bigger graphs. These resource limitations become noticeable in the execution time, e.g., in the Service Matching use case. A more comprehensive conclusion of our work follows in the next section.

[44]`http://commonsensereasoning.org/winograd.html` last visited 30.06.2017

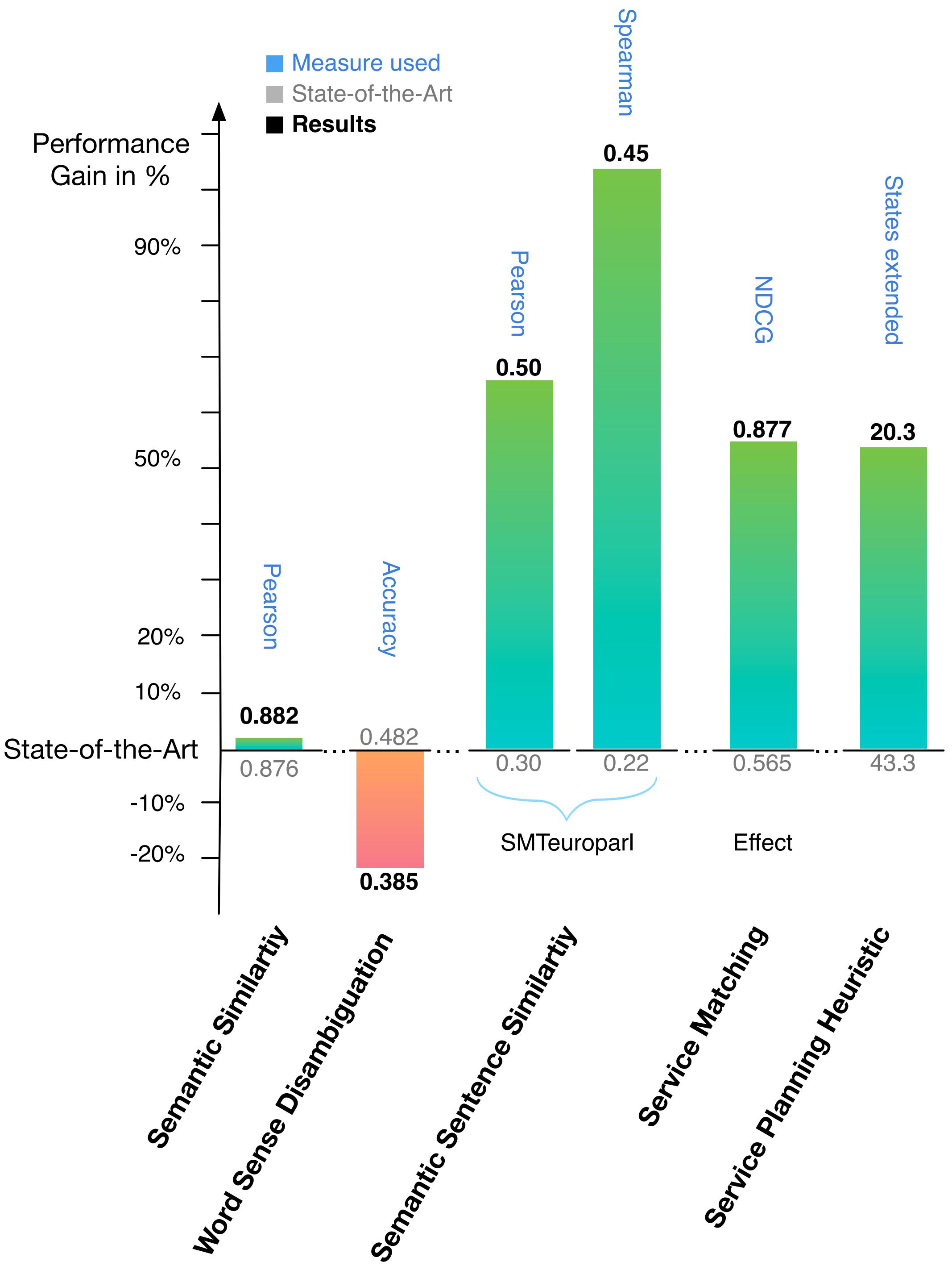


Figure 10.29.: Overview of example results in the different experiments in percentage of performance difference to the state-of-the-art baseline.

Part V.

Conclusion

11. Summary

This section concludes the here presented work. We will look at my hypotheses and the contributions I have made. However, first I give a short overview of my approach:

I have created an approach for automatically generating a semantic graph focused on a context and a reasoning using Marker Passing to answer questions about the knowledge encoded (for an abstract descriptions see Section 1.5). This approach has two parts:

Knowledge representation: This is the first part of the approach which tackled the challenge of automatically creating a knowledge representation. I did so by looking at the different semantic theories, selected the NSM approach and created a semantic decomposition to build a semantic graph automatically.

Reasoning: This is the second part of my approach which tackled the challenge of using semantic graphs to create reasoning capabilities for an agent. I approached this challenge by extending a cognitive mechanism of activation spreading with more symbolic information to Marker Passing. The resulting common sense reasoning has been shown by identifying subproblems in AI research, like the Sentence Similarity, Word Sense Disambiguation and Semantic Similarity for which I have proposed solutions which keep up or beat the state-of-the-art.

To use this approach one has to select data sources, set up the decomposition and create an application specific Marker Passing by selecting, e.g., the information on the Marker and in- and out-functions.

Next, we will look at my hypotheses and discuss the contributions of this work and their relation to those hypotheses. We start out with my experimental thesis:

I claim in my hypothesis that: "**Symbolic and Connectionist representation of meaning can help agents to reason with new concepts.**" I support this claim with a set of experiments which include the decomposition of new concepts and their integration into the semantic graph representing the agent's belief.

I am now able to answer my research questions proposed in Section 1.4, which have guided my research.

How can meaning formally be described in a context-dependent manner? The formalization of the result of the decomposition (see Section 6.4) allows us to encode factual knowledge in an semantic graph (see Section 6). The context dependence of meaning comes from the Marker Passing (see Section 7) and finally from the distribution of marker over the graph. This structure can represent meaning, which has been tested in multiple experiments (see Section 9).

I was able to show the usefulness of the so created artificial representation of meaning by showing that context-dependent heuristics can be created by observing service descriptions. The use of the artificial meaning representation to build a heuristic has been done by comparing the semantic distance of goal concepts and the concept describing a service (see Section 10.5.3). Additionally, I was able to show that the performance of agent planner and Service Matcher can be improved by using dynamic heuristics. The performance of Service Matcher improves by adding a kind of ontology matching to the comparison of service descriptions (see Section 10.4). The performance of my planner is improved since the heuristic lets us select only relevant services. This selection is done by combining all approaches build in this work (see Section 10.5).

Of course, there are many more questions which can be asked now like: How do we handle inconsistency in such a graph? How can logical reasoning be implemented using such a semantic graph? Which classes of problems are destined to be addressed with such an approach? We will discuss some of more concrete extensions next.

I did analyze the usefulness of the newly gained meaning representation in five experiments and with that answered the second research questions: **Can the reasoning be improved for the following tasks**:

In my experiment 1 I asked: "Does the representation of meaning include information to create a state-of-the-art semantic similarity measure?" This question was assured by my experiment in Section 10.1. Here I am able to use my approach to create a semantic similarity measure which outperforms the state-of-the-art. I show in Table 10.4 that the semantic similarity guessed by humans can be reproduced and with that, the meaning of concepts can be grasped.

In my experiment 2 I asked: "Does the representation of meaning include information to create a state-of-the-art Word Sense Disambiguation approach?" This was answered by my experiment in Section 10.2. Here I was able to use my decomposition to capture the different word meanings in the different definitions of the information sources. Table 10.5 shows that my approach can keep up with the state-of-the-art, even though there are better performing results. Here we can conclude that the decomposition is still too broad and that a selection of definitions to decompose should be made during decomposition to specialize the resulting semantic graph.

In my experiment 3 I asked "Does the representation of meaning include information to create a state-of-the-art semantic sentence similarity measure?" This question was answered by my experiment in Section 10.3. Here I was able to use my approach (including the results of my first experiment) to create a sentence similarity measure which matches the state-of-the-art. Table 10.11 shows the results on different data sets. Here we can conclude that the more difficult the data set, the more the state-of-the-art is outperformed by my approach.

In my experiment 4 I asked: "Can we improve the performance of Service Matchers by using the connectionist and symbolic representation of meaning as Ontology Matching?" This question was answered by my experiment in Section 10.4. Here I was able to use my approach to compare name, description, input, output precondition and effect of a service description. Except for input and precondition I was able to increase the performance of the Service Matcher. Here we gained the insight that some of the service description parts contain more semantic information to be analyzed with Marker Passing, other perform better with logical reasoning. Here the natural language part of the description of a service outperformed all other experts.

In my experiment 5 I asked: "Can heuristics for general purpose planners profit from using a connectionist and symbolic representation of meaning?" This question was answered by my experiment in Section 10.5. Here I were able to show that the newly created heuristic utilizing semantic information narrows the search space in a state space planning problem. To check this hypothesis, I build a service planner described in Algorithm 20 because I had to preserve the semantic information of service descriptions. I then build a dynamic heuristic that takes into account the goal of the given planning problem and uses the results of my experiments to search for semantically useful services. I showed that this heuristic performs well in Table 10.13.

With those experiments, I have shown that my approach is useful in different AI problems. These are convincing arguments for the usefulness of my artificial representation of meaning. I started out with the idea of an approach, which has more potential than I was able to use in the here described experiments. By adding additional interpretations for semantic primes, this approach could prosper further. For now, the specific interpretations of the primes are reduced to a general rule on how to handle all primes. But there are experiments in progress, which use the here proposed approach in common sense reasoning problems (e.g., the solution of Winograd Schemas[1]) to outperform the state-of-the-art.

After having accomplished all that, I have found drawbacks of the design decision I made during my scientific endeavor. I will discuss these drawbacks regarding the here analyzed experiments next.

[1]http://commonsensereasoning.org/winograd.html, last visited on 30.06.2017

12. Discussion

This section discusses design decision, drawbacks and the experiments and their relation to the overall goal of this thesis. Here we will emphasize on the drawbacks of the experiments. This includes the implicit design decisions which could be altered in a future application or further research of the here proposed approach.

Starting out from my approach, I had to select some data sources which are used during the decomposition. Those data sources are imperfect, they contain errors or include beliefs which are false. Furthermore, the here selected data sources do not forcibly align with the hierarchy of the NSM theory and the idea of Wierzbicka of a mental lexicon [410]. This has the consequence that a complete decomposition is unlikely and sometimes impossible. Because of the parameter of decomposition depth, it becomes questionable how much the NSM primes are beneficial to my approach. Perhaps this can be answered with an example: In the decomposition of the semantic similarity data set RG65 of 65 similarity pairs, a total of 227 semantic primes have been used at a decomposition depth of one. This number increases with decomposition depth of two to 17317.

Another drawback is the dependence of the Marker Passing on the decomposition. Thus if the performance of the approach is not as expected, it might at first be unclear if a faulty decomposition or a misconfigured Marker Passing is the cause. Solving such development problems needs experience on which data sources contain which kind of information so that the decomposition can be changed to fit the needs of the problem. Additionally, the developer needs a sufficient understanding of the Marker Passing and the effect change in parameters have on the result. During the design of the algorithm, the needed information in the decomposition can be estimated, and with that information, the needed data sources can be specified. Based on the available information in the semantic graph the Marker Passing can be specified.

Another uncertainty in the approach is the completeness of the data sources. The decomposition fails if none of the data sources contain a description of the given concept.

Next, we will look critically at the experiments I conducted and evaluated their outcome.

Experiment 1: Semantic Similarity Measure I tested my semantic similarity measure with a data set which states the similarity of word pairs as perceived by humans. A critical view on this approach could argue that the perception of the meaning of humans (especially the average of multiple human opinions) does not have to be the semantic distance of those concepts. The response of the humans depends on their education and beliefs. The average of multiple opinions mixes those different levels of knowledge and beliefs, which does no longer represent the semantic distance of those words in the sense of a semantic measure an expert or scientists would evaluate. For instance, an astronomer would not agree that the sun and a planet are semantically close, but for an average citizen, they are both round and in the sky, which might make them similar. Those two opinions can be discussed to be more or less correct, but the average will not yield a similarity representing what both subjects intended. Measuring an artificial representation of meaning on those semantic similarities, can in consequence, only partially measure the performance of the representation. It only shows that the approach can make the same mistakes as humans do. We would need a set of word pairs, where the semantic similarity is defined by experts regarding facts about the objects referenced by those words so that the artificial representation of meaning can be evaluated on how well it can represent the expert knowledge of humans. Another critique could be that the found weights for the semantic relations are not intuitive. Further investigations on which information in the semantic graph can be used by the Marker Passing is needed. This implies that there is room for improvement. The found result is most likely only a local optimum.

Experiment 2: Word Sense Disambiguation The data sets used in WSD comparisons are based on WordNet sense keys to identify the word sense. This, of course, is not the case in other information sources, where different definitions can occur. Those definitions are then not mapped to the WordNet sense keys; as a result, the do not count as a right answer. The missing syntactical information makes it even harder for my approach to use all information available in the surrounding words of the sentence. The results could be improved by integrating syntactic information and thus giving the approach more information in which context the word is used in.

Experiment 3: Semantic Sentence Similarity Measure My approach is a purely semantic approach. This might be good for analyzing the semantics of single concepts, but with sentences, the order of words, the inflection, and the syntax make up the meaning of the sentence. Most of this is disregarded in the here proposed results. With the reduction of concepts to their stem, the inflection is lost. Additionally, the decomposition is combined to one graph, losing all information about word order or syntax. Hence this approach can not make a difference between the sentences: "Johannes bit the dog" and "The dog bit Johannes." Even though the performance of the semantic sentence similarity measure shows the ability to outperform the state-of-the-art in some cases, the logical (in means of truth-value) meaning of a sentence is not captured by it. This shortcoming could be countered by extending the semantic decomposition with syntactic information and create a Marker Passing algorithm which uses this syntactic information.

Experiment 4: Semantic Service Matching The results of the service matching experiment have shown that the semantic analysis performs well on semantically rich parts of the service description, but need far more resources than the compared approaches. Table 10.10 shows that the Marker Passing on natural language descriptions takes up to ten times more time than the other experts. This could be avoided by combining those to expert types on the different description parts. Also, the overall performance could be improved by learning the expert's weights. The Marker Passing does some kind of ontology matching between the concepts used in the service description. This means that before using such an approach, the service description needed to use the same ontologies. With the Marker Passing, this is no longer needed since the decomposition creates a merging ontology. This ability exceeds the state-of-the-art of Service Matcher and should be analyzed further. For example to answer the question on how the inputs are transformed if they are semantically similar, but have different data types.

Experiment 5: Heuristics in AI service planning The planning problem discussed here is not a straight forward optimization problem because we do not have costs for the services. The kind of problem solved here means there is no optimization goal for the solution plan. The problem can be made to an optimization goal if, e.g., the services provide Quality of Service parameters which directly include a cost or can be used to calculate a service execution cost. Additionally, the evaluation data set does not provide a cost function for the services. The selection of the data set is another weak point of this thesis since I were unable to find any other OWL-S based planning problem. The work on this topic has produced an entire tool chain to describe such data sets for the future (see Appendix A.1 for a short introduction).

13. Final Remarks and Future Work

"Because this world doesn't belong to you, or the people who came before. It belongs to someone who is yet to come."
- DOLORES

Stepping back from the research done in this thesis, we can again ask if the overhead of creating, e.g., semantic service description, an OWLS description of start and goal state and the ontologies needed for that, is worth the while, regarding the problem to be solved. We can question if this overhead can be countered by the benefit of reusing services. I have encountered this problem in applications of this approach in publicly funded research projects. Those projects developed tools to ease the description of services and their composition as described in the methodology in Section A.1.

In comparison to PDDL problem descriptions, often used in academic examples, the problem description includes only services needed to solve the problem. Consequently, a selection of the appropriate service is not part of the problem. Additionally, the problems typically described in PDDL are lacking semantics, like the 15-puzzle [322] which is more about making the same four moves in a particular order, which was difficult to conclude from the semantics of the move.

Making a plan and executing it are two different things. It seems unrealistic to prepare a plan for the most likely eventualities, meaning that during the execution the environment could change and a plan could fail. This work presents a mechanism to choose from the large set of possible services available to an agent.

In the case of incomplete information in the data sources, I have extended the decomposition with a component which then provides the user with a user interface where a definition can be described manually. There are multiple points, which can be handled semi-automatically: first the selection of the concept type. Here the word type needs to be specified. This parameter influences the definition selection as well as the search for synonyms, antonyms hypernyms, and hyponyms. Second semi-automatization could be the selection of the definition. Here the sense of a word is selected. The user can select which definition to use. In the example of "well" this could be the meaning of in a satisfactory way or a hole in the ground with water in it. The third semi-automatization point could be the decomposition of a concept which has no definition or where the definition is circular. A concept could have no definition if the used dictionary does not contain a description of the word. In this case, the decomposition stops at this prime and might only continue with the decomposition after a manual definition or decomposition is introduced. Fourthly the breadth of the recursion needs to be defined. This parameter specifies how many synonyms and antonyms are searched for in each decomposition step. Fifthly the amount of additional information included. This parameter specifies how many examples of hypernyms and hyponyms are included in the decomposition. Further, the information sources need to be specified. The specification includes the used dictionary and a mapping of the information sources to the information retrieval parts of the decomposition.

With every scientific endeavor at the end, there are always more questions to be answered. I will split the future work into two parts: First the improvement of the approach and second the improvements for the heuristics in service planning.

Artificial Meaning: This work is restricted to the semantic analysis. This needs to be extended with a syntactical analysis. The Marker Passing has not yet been tested for its full potential, e.g., encoding logical symbolic information of negation on a marker and using it in in-, out- and edge-functions. The selection of the right parameters or a guide how to select them concerning a type of problem could be investigated in the future. Reasoning with the Marker Passing could be tested on more complex problems like Winograd Schemas [230]. Last but not least, additional data sources can be added to the decomposition to improve future results. For example, the integration of the semantic of the NSM primes could be an addition to the Marker Passing, since then meaning could be reasoned bottom up.

Heuristics: This work has stopped with the planning, which leaves the execution and monitoring as well as plan repair to future work. The question to answer here is: How can those part profits from the semantic information given by the problem? The expressiveness of the description language can further be developed since modern reasoners do not implement all SWRL built-ins making the description of a planning problem more complex than it needs to be. In the end, better data sets are needed to evaluate the drawbacks and benefits of service planning and proposed heuristics.

Part VI.

Bibliography, Glossary, Index, and Appendix

A. Appendix

A.1. Tools

This section describes the tools and the development methodology published in [101]. We describe this publication here since one o the common question encountered during scientific presentations of our work is: Does anyone use this semantic approach? It seems like it is too much work for the benefit." To show how this creation can be simplified, we have described our tools to creating semantic services and composing them. The tools are shown in Figure A.1.

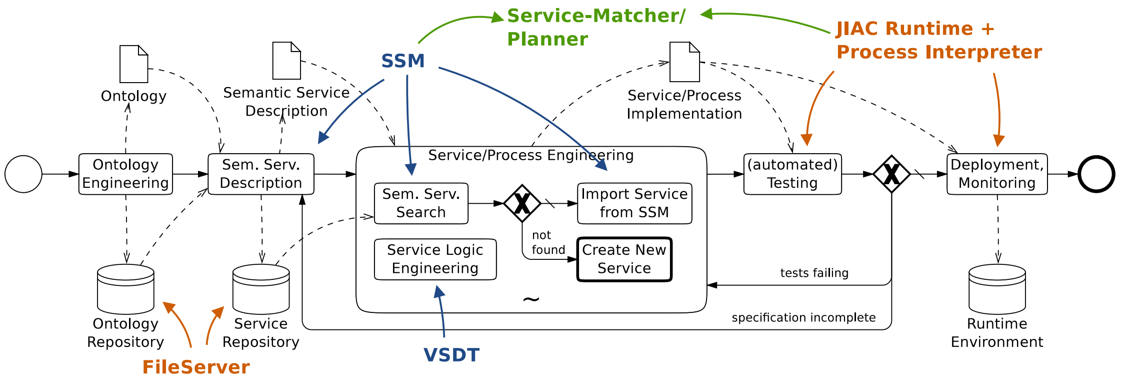


Figure A.1.: Service Development tools and their interplay.

In Figure A.1 blue denotes tools that are for manual Development, Green denotes automatic aids and orange are the surrounding runtimes. We foresee the development process to state with the **ontology engineering**, which builds upon existing ontologies. Here the Domain descriptions, with all its entities and their relations are modeled. To ease the ontology engineering for developers, we create an encore (EMF) to OWL transformation. With that, a developer can use its usual class modeling tool and translate it into OWL later on. These Ontologies are then used to create **semantic service descriptions**, which are the explanation of what we want to do or have done. This service description is used to find existing service with the **Service-Matcher**, to foster reusability or to serve as goal state of the **planner**. If the service matcher can not locate a service, which fulfills the request, the service can be engineered using the planner, a **Semantic Service Management**, and the **Visual Service Design Tool (VSDT)** which is a Business Process Modeling Notation (BPMN) editor which integrates the other components. The outcome of this engineering task is a service implementation and a semantic service description which references this service in its grounding. Here a **Java-based Intelligent Agent Componentware (JIAC)** agent is automatically generated which implements the service. The Service description is automatically deployed into the runtime Environment, and thus further development can use this new service.

A.2. Class Diagrams

This section holds the overview class diagrams of the implementation of the decomposition. Figure A.2 shows the main classes and their relations.

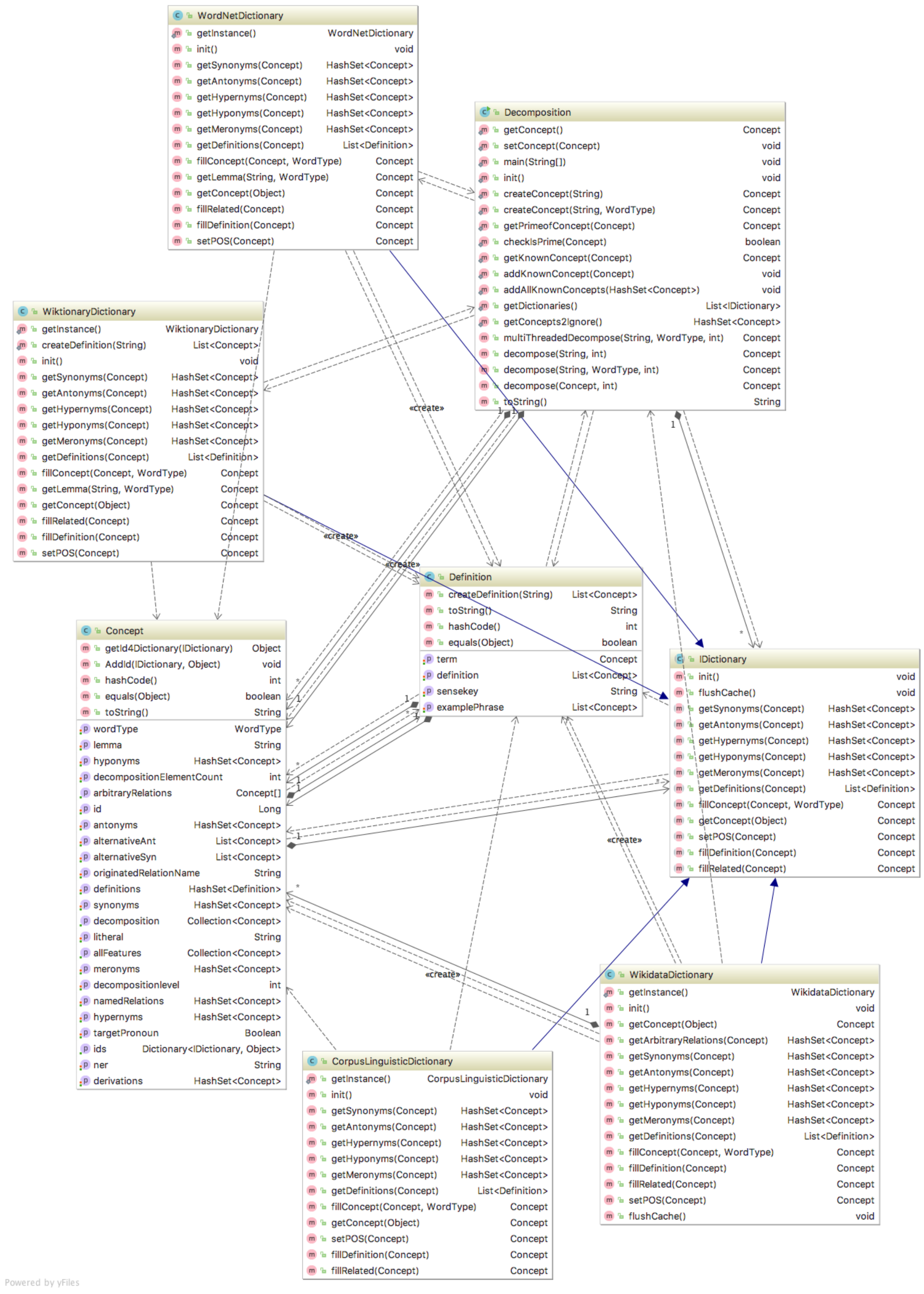

WordNetDictionary
getInstance() WordNetDictionary
init() void
getSynonyms(Concept) HashSet<Concept>
getAntonyms(Concept) HashSet<Concept>
getHypernyms(Concept) HashSet<Concept>
getHyponyms(Concept) HashSet<Concept>
getMeronyms(Concept) HashSet<Concept>
getDefinitions(Concept) List<Definition>
fillConcept(Concept, WordType) Concept
getLemma(String, WordType) Concept
getConcept(Object) Concept
fillRelated(Concept) Concept
fillDefinition(Concept) Concept
setPOS(Concept) Concept
Decomposition
getConcept() Concept
setConcept(Concept) void
main(String[]) void
init() void
createConcept(String) Concept
createConcept(String, WordType) Concept
getPrimeofConcept(Concept) Concept
checkIsPrime(Concept) boolean
getKnownConcept(Concept) Concept
addKnownConcept(Concept) void
addAllKnownConcepts(HashSet<Concept>) void
getDictionaries() List<IDictionary>
getConcepts2Ignore() HashSet<Concept>
multiThreadedDecompose(String, WordType, int) Concept
decompose(String, int) Concept
decompose(String, WordType, int) Concept
decompose(Concept, int) Concept
toString() String
WiktionaryDictionary
getInstance() WiktionaryDictionary
createDefinition(String) List<Concept>
init() void
getSynonyms(Concept) HashSet<Concept>
getAntonyms(Concept) HashSet<Concept>
getHypernyms(Concept) HashSet<Concept>
getHyponyms(Concept) HashSet<Concept>
getMeronyms(Concept) HashSet<Concept>
getDefinitions(Concept) List<Definition>
fillConcept(Concept, WordType) Concept
getLemma(String, WordType) Concept
getConcept(Object) Concept
fillRelated(Concept) Concept
fillDefinition(Concept) Concept
setPOS(Concept) Concept
«create»
«create»
«create»
Definition
createDefinition(String) List<Concept>
toString() String
hashCode() int
equals(Object) boolean
term Concept
definition List<Concept>
sensekey String
examplePhrase List<Concept>
Concept
getId4Dictionary(IDictionary) Object
AddId(IDictionary, Object) void
hashCode() int
equals(Object) boolean
toString() String
wordType WordType
lemma String
hyponyms HashSet<Concept>
decompositionElementCount int
arbitraryRelations Concept[]
id Long
antonyms HashSet<Concept>
alternativeAnt List<Concept>
alternativeSyn List<Concept>
originatedRelationName String
definitions HashSet<Definition>
synonyms HashSet<Concept>
decomposition Collection<Concept>
litheral String
allFeatures Collection<Concept>
meronyms HashSet<Concept>
decompositionlevel int
namedRelations HashSet<Concept>
hypernyms HashSet<Concept>
targetPronoun Boolean
ids Dictionary<IDictionary, Object>
ner String
derivations HashSet<Concept>
IDictionary
init() void
flushCache() void
getSynonyms(Concept) HashSet<Concept>
getAntonyms(Concept) HashSet<Concept>
getHypernyms(Concept) HashSet<Concept>
getHyponyms(Concept) HashSet<Concept>
getMeronyms(Concept) HashSet<Concept>
getDefinitions(Concept) List<Definition>
fillConcept(Concept, WordType) Concept
getConcept(Object) Concept
setPOS(Concept) Concept
fillDefinition(Concept) Concept
fillRelated(Concept) Concept
«create»
«create»
WikidataDictionary
getInstance() WikidataDictionary
init() void
getConcept(Object) Concept
getArbitraryRelations(Concept) HashSet<Concept>
getSynonyms(Concept) HashSet<Concept>
getAntonyms(Concept) HashSet<Concept>
getHypernyms(Concept) HashSet<Concept>
getHyponyms(Concept) HashSet<Concept>
getMeronyms(Concept) HashSet<Concept>
getDefinitions(Concept) List<Definition>
fillConcept(Concept, WordType) Concept
fillDefinition(Concept) Concept
fillRelated(Concept) Concept
setPOS(Concept) Concept
flushCache() void
CorpusLinguisticDictionary
getInstance() CorpusLinguisticDictionary
init() void
getSynonyms(Concept) HashSet<Concept>
getAntonyms(Concept) HashSet<Concept>
getHypernyms(Concept) HashSet<Concept>
getHyponyms(Concept) HashSet<Concept>
getMeronyms(Concept) HashSet<Concept>
getDefinitions(Concept) List<Definition>
fillConcept(Concept, WordType) Concept
getConcept(Object) Concept
setPOS(Concept) Concept
fillDefinition(Concept) Concept
fillRelated(Concept) Concept
Powered by yFiles

Figure A.2.: Class diagram of the decomposition.

Prime	Context	Synonym
KIND	being of one type	genre, derivative, sort, type, child
PART	meronym	shed, role, portion, component, parting, piece, constituent, faction, shoad, position, region, component part, shode, party, element

Semantic primes in the category relational in the English language [140].

Prime	Context	Synonym
THIS	direct reference	
THE SAME	equivalence relation	identical, equal, similar, alike, equivalent
OTHER	discrimination	farther, additional, another, unalike, former, dissimilar, disparate, distinctive, else, unlike, distinguishable, diverse, further, different, early

Semantic primes in the category determiners in the English language [140].

A.3. Algorithms

This section contains the detailed descriptions of the algorithms mentioned in this work which have been placed here to gain on readability.

A.3.1. Marker Passing algorithm

This section holds a abstraction of the maker passing algorithm in pseudo code.

Algorithm 22 Spreading Activation - adapted from Crestani [61].

Name: Spreading Activation
Input: Marking
Output: Marking

```
function SPREADINGACTIVATION($M_{in}$)
    $M \leftarrow M_{in}$
    while $\neg$ TERMINATIONCONDITION($M$) do
        $M \leftarrow$ PREADJUSTMENT($M$);
        $M \leftarrow$ PASSING($M$);
        $M \leftarrow$ POSTADJUSTMENT($M$);
    end while
    return $M$
end function
```

A.4. Natural Semantic Metalanguage

This section lists the semantic primes with their synonyms and their domain if available.

Prime	Context	Synonym
ONE	Number	single, unity, 1, ace,
TWO	Number	II, 2
MANY	Number	a lot, loads, plenty, a great deal, greatly, much, highly, very much
SOME	quantifiers	a few
ALL	quantifiers	completely

Semantic primes in the category quantifiers in the English language [140].

Prime	Context	Synonym
GOOD	evaluators	commodity, not bad, accomplished, well, satisfactory, decent, good, all right, trade good, goodness
BAD	evaluators	badness, spurious, malodorous, wanting, bad, odious, awful, faulty, foul, wretched, coarse, poor, horrible, dreadful, unsatisfactory, abandoned, badass, terrible, lousy, inadequate, vicious, miserable, ill, dismal, abysmal, horrid, hideous, detestable, punk, unfavorable, loathsome, evil, rotten, substandard, incompetent, inelegant, negative, dire, disgusting, vile, urgent, ungodly, crummy, deficient, wicked, sloppy, unacceptable, corrupt, inferior, abominable, false, disagreeable, off, wrong, severe, hopeless, intolerable, base

Semantic primes in the category evaluators in the English language [140].

Prime	Context	Synonym
BIG	size	big, prominent, jumbo, bad, large, grown up, heavy, massive, ample, major leagues, huge, sizeable, stoor, adult
SMALL	size	slight, mini, compact, lowercase, insignificant, young, slim, minuscule, minute, modest, wee, tiny, miniature, small-scale, microscopic, petty, little

Semantic primes in the category descriptors in the English language [140].

Prime	Context	Synonym
THINK	cognition	imagine, cogitate, deem, ponder, suppose, opine, reflect, guess, find, regard, ruminate, judge, consider, reckon
KNOW	cognition	
WANT	cognition	wish, set one's heart on, deficiency, need, would like, deprivation, neediness, privation, wishing, lack
FEEL	cognition	flavor, flavour, tone, tactile property, spirit, feel, feeling, smell, look
SEE	cognition	descry, view, get, espy, understand, behold, follow, observe
HEAR	cognition	get word, discover, find out, pick up, see, get wind, get a line, learn
TOUCH	cognition	trace, touching, cutaneous senses, skin senses, sense of touch, touch modality

Semantic primes in the category mental predicates in the English language [140].

Prime	Context	Synonym
SAY	speech	
WORDS	speech	wrangle, lyric, dustup, quarrel, language
TRUE	logic	

Semantic primes in the category speech in the English language [140].

Prime	Context	Synonym
DO	action	to-do, bash, do, DO, brawl
HAPPEN	event	fall out, befall, pass, hap, occur, go on, come about, take place, pass off
MOVE	action	motion, affect, instigate, impel, actuate, relocation, propose, persuade, influence, offer, rouse, induce, transfer, incite, removal, motility, trouble, incline, stir, agitate, movement, prompt

[
Semantic primes in the category actions, events in the English language [140].]Semantic primes in the category actions, events, movement, contact in the English language [140].

Prime	Context	Synonym
BE_{at}	location	
THERE IS	exsistence	
HAVE	possession	
BE_{is-a}	specification	

[
Semantic primes in the category location, possession, in the English language [140].]Semantic primes in the category location, existence, possession, specification in the English language [140].

Prime	Context	Synonym
LIVE		survive, living, last, alive, inhabit, in the flesh, dwell, in person, live on, populate, endure
DIE	live	pass away, die, join the choir invisible, pass on, dice, bite the big one, lose one's life, snuff it, pop one's clogs, cross over, keel over, decease, check out, go the way of all flesh, bite the dust, cash in, perish, go the way of the dinosaurs, go the way of the dodo, croak, shuffle off this mortal coil, pass, buy it, meet one's maker, kick the bucket, draw one's last breath, give up the ghost, forfare, disincarnate, knock off, expire, buy the farm, bite the biscuit, cash in one's chips, cark, yield up the ghost, take a dirt nap, succumb, give one's all, cease to be

Semantic primes in the category life and death in the English language [140].

Prime	Context	Synonym
WHEN		
NOW		
BEFORE		earlier, previously, before, ahead, in front, by, in front, lest, in front of, ahead of
AFTER		
A LONG TIME		
A SHORT TIME		
FOR SOME TIME		
MOMENT		stound, bit, second, moment, minute, instant, tic, sec, overnight, trice, blink of an eye, here and now, present moment, moment of force, jiffy, nothing flat, flash

Semantic primes in the category time in the English language [140]

Prime	Context	Synonym
WHERE		
HERE		
AVOCE		
BELOW		down the stairs, beneath, on a lower floor, to a lower place, below, at a lower place, downwards, downstream, infra, under, downstairs, underneath
FAR		
NEAR		near side, come on, come near, near, go up, approach, draw close, draw near
SIDE		
INSIDE		indoors, inside, interior

Semantic primes in the category space in the English language [140].

Prime	Context	Synonym
NOT		non
MAYBE		perhaps, perchance, mayhaps, peradventure, mayhap, possibly
CAN		may, be able to,
BECAUSE	causality	as, for that, for, therefore, forthy, inasmuch as, since, forwhy
IF		

Semantic primes in the category logical in the English language [140].

Prime	Context	Synonym
VERY		identical, very, swith, selfsame, ilk, main, sore, ever so
MORE		

Semantic primes in the category intensifier in the English language [140].

Prime	Context	Synonym
LIKE		such as

Semantic primes in the category similarity in the English language [140].

Bibliography

[1] Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016. In Amanda Coles, Andrew Coles, Stefan Edelkamp, Daniele Magazzeni, and Scott Sanner, editors, *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016*. AAAI Press, 2016.

[2] Sunitha Abburu. A survey on ontology reasoners and comparison. *International Journal of Computer Applications*, 57(17), 2012.

[3] Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. A study on similarity and relatedness using distributional and WordNet-based approaches. In *NAACL '09: Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 19–27. Charles University in Prague, Association for Computational Linguistics, 2009.

[4] Eneko Agirre, Oier de Lacalle, and Aitor Soroa. Random Walks for Knowledge-Based Word Sense Disambiguation. *Computational Linguistics*, 40(1):57–84, 2014.

[5] Eneko Agirre and Philip Edmonds. *Word sense disambiguation: Algorithms and applications*, volume 33. Springer Science & Business Media, 2007.

[6] Rama Akkiraju, Biplav Srivastava, Anca-Andreea Ivan, Richard Goodwin, and Tanveer Syeda-Mahmood. SEMAPLAN: Combining Planning with Semantic Matching to Achieve Web Service Composition. In *2006 IEEE International Conference on Web Services (ICWS'06)*, pages 37–44. IEEE, 2006.

[7] Harith Alani, Sanghee Kim, D Millard, M Weal, W Hall, P Lewis, and N Shadbolt. Automatic ontology-based knowledge extraction from Web documents. *IEEE INTELLIGENT SYSTEMS*, 18(1):14–21, 2003.

[8] Jens Allwood and Peter Gärdenfors. *Cognitive Semantics*, volume 55 of *Meaning and Cognition*. John Benjamins Publishing, 1999.

[9] Gahadah Altaiari. Editor zur semantischen Dekomposition. Bachelors thesis, 2014. Technische Universität Berlin.

[10] John Anderson. A spreading activation theory of memory. *Journal of Verbal Learning and Verbal Behavior*, 22(3):261–295, 1983.

[11] Avery Andrews. Reconciling NSM and Formal Semantics. *Australian Journal of Linguistics*, 36(1):79–111, 2015.

[12] Mark Aronoff and Janie Rees-Miller. *The handbook of linguistics.* Blackwell Publishers Ltd, 2003.

[13] Christer Bäckström and Bernhard Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[14] Jay Bagga and Adrian Heinz. JGraph— A Java Based System for Drawing Graphs and Running Graph Algorithms. In *Graph Drawing*, pages 459–460. Springer, Berlin, Heidelberg, Berlin, Heidelberg, 2001.

[15] Tina Balke and Nigel Gilbert. How Do Agents Make Decisions? A Survey. *Journal of Artificial Societies and Social Simulation*, 17(4), 2014.

[16] Andrea Ballatore, Michela Bertolotto, and David Wilson. An evaluative baseline for geo-semantic relatedness and similarity. *arXiv preprint arXiv:*, cs.CL(4):747–767, 2014.

[17] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247, Stroudsburg, PA, USA, 2014. Association for Computational Linguistics.

[18] Alistair Barros and Daniel Oberle. *Handbook of Service Description*. USDL and Its Methods. Springer Science & Business Media, Boston, MA, 2012.

[19] Jon Barwise and Robin Cooper. Generalized Quantifiers and Natural Language. In *Philosophy, Language, and Artificial Intelligence*, pages 241–301. Springer Netherlands, Dordrecht, 1988.

[20] Osman Başkaya and David Jurgens. Semi-supervised Learning with Induced Word Senses for State of the Art Word Sense Disambiguation. *Journal of Artificial Intelligence Research*, 55:1025–1058, 2016.

[21] Nicole Beckage and Eliana Colunga. Language Networks as Models of Cognition: Understanding Cognition through Language. In *Towards a Theoretical Framework for Analyzing Complex Linguistic Networks*, pages 3–28. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[22] Richard Bellman. *Dynamic programming*. Rand research study. Princeton Univ. Press, Princeton, NJ, 1957.

[23] Ayse Bener, Volkan Ozadali, and Erdem Ilhan. Semantic matchmaker with precondition and effect matching using SWRL. 36(5):9371–9377, 2009.

[24] Tim Berners-Lee. Semantic Web Road map: A road map for the future, an architectural plan untested by anything except thought experiments, 1998. `http://www.w3.org/DesignIssues/Semantic.html` Last visited: 07.07.2017.

[25] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 2001.

[26] Michael Berthold, Ulrik Brandes, Tobias Kötter, Martin Mader, Uwe Nagel, and Kilian Thiel. Pure spreading activation is pointless. *CIKM*, pages 1915–1918, 2009.

[27] Leonora Bianchi, Marco Dorigo, Luca Gambardella, and Walter Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2):239–287, 2009.

[28] Chris Biermann. Ontology Learning from Text: A Survey of Methods. *LDV-Forum*, 20(2):75–93, 2005.

[29] Leonard Bloomfield. Sentence and Word. *Transactions and Proceedings of the American Philological Association*, 45:65, 1914.

[30] Leonard Bloomfield. *Language*. University of Chicago Press, New York: Holt, 1933.

[31] Leonard Bloomfield. Language or Ideas? *Language*, 12(2):89–95, 1936.

[32] Leonard Bloomfield. Meaning. *Monatshefte für deutschen Unterricht*, 35(3/4):101–106, 1943.

[33] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase - a collaboratively created graph database for structuring human knowledge. *SIGMOD Conference*, page 1247, 2008.

[34] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[35] Paolo Bouquet, Fausto Giunchiglia, Frank Van Harmelen, Luciano Serafini, and Heiner Stuckenschmidt. C-OWL: Contextualizing ontologies. In *International Semantic Web Conference*, volume 2870, pages 164–179, 2003.

[36] Ros Bramwell. Education and some aspects of meaning: A background study. *British Journal of Educational Studies*, 20(1):12–26, 1972.

[37] Benjamin Brand. Semantic Distance of Service Descriptions through Activation Spreading. Master's thesis, 2017. Technische Universtität Berlin.

[38] Lars Braubach, Alexander Pokahr, Daniel Moldt, and Winfried Lamersdorf. Goal Representation for BDI Agent Systems. volume 3346, pages 44–65. Springer Berlin Heidelberg, 2004.

[39] Elia Bruni, Nam Tran, and Marco Baroni. Multimodal Distributional Semantics. *Journal of Artificial Intelligence Research*, 49(2014):1–47, 2014.

[40] Edmund Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[41] Johannes Busse, Bernhard Humm, Christoph Lübbert, Frank Moelter, Anatol Reibold, Matthias Rewald, Veronika Schlüter, Bernhard Seiler, Erwin Tegtmeier, and Thomas Zeh. Was bedeutet eigentlich Ontologie? *Informatik Spektrum*, 37(4):286–297, 2014.

[42] José Camacho-Collados, Mohammad Pilehvar, and Roberto Navigli. NASARI - a Novel Approach to a Semantically-Aware Representation of Items. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (HLT-NAACL)*, pages 567–577, 2015.

[43] José Camacho-Collados, Mohammad Pilehvar, and Navigli Roberto. A unified multilingual semantic representation of concepts. In *International Joint Conference on Natural Language Processing (ACL)*, pages 741–751, 2015.

[44] Erik Cambria and Bebo White. Jumping nlp curves: A review of natural language processing research. *IEEE Computational intelligence magazine*, 9(2):48–57, 2014.

[45] Ronnie Cann. *Formal Semantics*. An introduction. Cambridge University Press, Cambridge, 2009.

[46] Cristiano Castelfranchi. Simulating with Cognitive Agents - The Importance of Cognitive Emergence. *MABS*, 1534(Chapter 3):26–44, 1998.

[47] Luca Cavallaro, Pete Sawyer, Daniel Sykes, Nelly Bencomo, and Valérie Issarny. Satisfying requirements for pervasive service compositions. In *Workshop on Models at run. time*, pages 17–22, New York, New York, USA, 2012. ACM Press.

[48] Eugene Charniak. A Neat Theory of Marker Passing. *Association for the Advancement of Artificial Intelligence*, pages 584–588, 1986.

[49] Dong Chen, Yan Jianzhuo, Fang Liying, and Shi Bin. Measure semantic distance in wordnet based on directed graph search. *2009 International Conference on E-learning, E-Business, Enterprise Information Systems, and E-Government, EEEE 2009*, pages 57–60, 2009.

[50] Jingwei Chen, Robert Holte, Sandra Zilles, and Nathan Sturtevant. Front-to-End Bidirectional Heuristic Search with Near-Optimal Node Expansions. *arXiv.org*, page arXiv:1703.03868, 2017.

[51] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. Web services description language (wsdl) version 2.0 part 1: Core language. *W3C recommendation*, 2007.

[52] Noam Chomsky. Logical Syntax and Semantics: Their Linguistic Relevance. *language*, 31(1):36, 1955.

[53] Noam Chomsky. Syntactic Structures. *Language*, 33(3 Part 1):375–408, 1957.

[54] Noam Chomsky. *Aspects of the Theory of Syntax*, volume 119. MIT press, 1969.

[55] Noam Chomsky. *Syntactic Structures*. Walter de Gruyter, 2002.

[56] William Cole. Understanding Bayesian reasoning via graphical displays. volume 20, pages 381–386, New York, NY, USA, 1989. ACM.

[57] Joe Coleman, Michael O Rourke, and Dean Edwards. Pragmatics and Agent Communication Languages. In *Proceedings of the ESSLLI Workshop on Formal Ontologies for Communicating Agents*, pages 1–13, 2006.

[58] Allan Collins and Elizabeth Loftus. A spreading-activation theory of semantic processing. *Psychological Review*, 82(6):407–428, 1975.

[59] Allan Collins and Ross Quillian. Retrieval time from semantic memory. *Journal of Verbal Learning and Verbal Behavior*, 8(2):240–247, 1968.

[60] Courtney Corley and Rada Mihalcea. Measuring the semantic similarity of texts. In *Proceedings of the ACL workshop on empirical modeling of semantic equivalence and entailment*, pages 13–18, Morristown, NJ, USA, 2005. Association for Computational Linguistics.

[61] Fabio Crestani. Application of Spreading Activation Techniques in Information Retrieval. *Artificial Intelligence Review*, 11(6):453–482, 1997.

[62] William Croft and Alan Cruse. *Cognitive Linguistics*. Cambridge University Press, 2004.

[63] Mohamed Dahab, Hesham Hassan, and Ahmed Rafea. TextOntoEx: Automatic ontology construction from natural English text. *Expert Systems with Applications*, 34(2):1474–1480, 2008.

[64] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50, 1993.

[65] Simon De Deyne, Yoed Kenett, David Anaki, and Miriam Faust. Large-scale network representations of semantics in the mental lexicon. *Big data in cognitive science: From methods to insights*, pages 174–202, 2016.

[66] Juan de Lara, Roswitha Bardohl, Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. Attributed graph transformation with node type inheritance. *Theor. Comput. Sci. ()*, 376(3):139–163, 2007.

[67] Ferdinand de Saussure, Charles Bally, Albert Sechehaye, Albert Reidlinger, and Wade Baskin. Course in General Linguistics. *The Journal of American Folklore*, 73(289):274, 1960.

[68] Joost de Winter and Dimitra Dodou. Why the Fitts list has persisted throughout the history of function allocation. *Cognition, Technology & Work*, 16:1–11, 2014.

[69] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality af A*. *Journal of the ACM*, 32(3):505–536, 1985.

[70] Emanuele Della Valle, Dario Cerizza, and Irene Celino. The mediators centric approach to automatic web service discovery of glue. *International Workshop on. Mediation in Semantic Web Services (MEDIATE2005)*, 168:35–50, 2005.

[71] Anind Dey. Understanding and Using Context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.

[72] Michel Deza and Elena Deza. *Encyclopedia of distances*. Springer-Verlag, Berlin, Berlin, Heidelberg, 2009.

[73] Antonio Di Marco and Roberto Navigli. Clustering and Diversifying Web Search Results with Graph-Based Word Sense Induction. *Computational Linguistics*, 39(3):709–754, 2013.

[74] Patrick Doherty, Witold Lukaszewicz, and Andrzej Szalas. Efficient Reasoning Using the Local Closed-World Assumption. In *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, pages 49–58. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[75] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault. In *the 20th ACM SIGKDD international conference*, pages 601–610, New York, New York, USA, 2014. ACM Press.

[76] David Dowty. Mantague Grammar and The Lexical Decomposition of causative Verbs. In *Montague Grammar*, pages 201–245. Academic Press Inc., 1976.

[77] David Dowty. Lexical Decomposition in Montague Grammar. In *Word Meaning and Montague Grammar*, pages 193–234. Springer Netherlands, Dordrecht, 1979.

[78] Zvi Drezner, Peter Hahn, and Éeric Taillard. Recent Advances for the Quadratic Assignment Problem with Special Emphasis on Instances that are Difficult for Meta-Heuristic Methods. *Annals of Operations Research*, 139(1):65–94, 2005.

[79] Marek Druzdzel. Qualitative Verbal Explanations in Bayesian Belief Networks. *Artificial Intelligence and Simulation of Behaviour Quarterly*, 94:43–54, 1996.

[80] Richard Duda, Peter Hart, and David Stork. *Pattern classification*. Wiley-Interscience, New York, second edition, 2001.

[81] Roman Dumitru, Jacek Kopecký, Tomas Vitvar, John Domingue, and Dieter Fensel. WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs. *Web Semantics: Science, Services and Agents on the World Wide Web*, 31:39–58, 2015.

[82] Uwe Durst. The Natural Semantic Metalanguage approach to linguistic meaning. *Theoretical Linguistics*, 29(3):157–200, 2003.

[83] Mnacho Echenim and Nicolas Peltier. A Superposition Calculus for Abductive Reasoning. *J. Autom. Reasoning*, 57(2):97–134, 2016.

[84] Philip Edmonds. SENSEVAL: The evaluation of word sense disambiguation systems. *European Language Resources Association (ELRA) newsletter*, 7, 2002.

[85] Hartmut Ehrig and Bernd Mahr. *Fundamentals of algebraic specification 1* . Springer Science & Business Media, 1985.

[86] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. A logic programming approach to knowledge-state planning: Semantics and complexity. *ACM Transactions on Computational Logic (TOCL)*, 5(2):206–263, 2004.

[87] Mohamad El Falou, Maroua Bouzid, Abdel-Illah Mouaddib, and Thierry Vidal. Automated Web Service Composition: A Decentralised Multi-agent Approach. 1:387–394, 2009.

[88] Chris Elsaesser. Explanation of Probabilistic Inference. In *Association for Uncertainty in Artificial Intelligence*, pages 387–400, 1987.

[89] Thomas Erl. *Service-oriented architecture*. concepts, technology, and design. Prentice Hall, 2005.

[90] Andre Esteva, Brett Kuprel, Roberto Novoa, Justin Ko, Susan Swetter, Helen Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, 2017.

[91] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching*. Springer-Verlag Berlin Heidelberg, 2007.

[92] Scott Fahlman. Marker-Passing Inference in the Scone Knowledge-Base System. In *Knowledge Science, Engineering and Management*, pages 114–126. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[93] Johannes Fähndrich. Best First Search Planning of Service Composition Using Incrementally Refined Context-Dependent Heuristics. In *14th German Conference Multiagent System Technologies*, pages 404–407, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[94] Johannes Fähndrich, Sebastian Ahrndt, and Sahin Albayrak. Mining Application Development for Research. In *Highlights in Practical Applications of Agents and Multi-Agent Systems. 10th International Conference on Practical Applications of Agents and Multi-Agent Systems*, pages 171–178. DAI-Labor, Technische Universität Berlin, Ernst-Reuter-Platz 7, 10587 Berlin, Germany, Springer Berlin / Heidelberg, 2012.

[95] Johannes Fähndrich, Sebastian Ahrndt, and Sahin Albayrak. Self-Explaining Agents. *Jurnal Teknologi (Science & Engineering)*, 63(3):53–64, 2013.

[96] Johannes Fähndrich, Sebastian Ahrndt, and Sahin Albayrak. Towards Self-Explaining Agents. *International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS)*, 221:147–154, 2013.

[97] Johannes Fähndrich, Sebastian Ahrndt, and Sahin Albayrak. Are There Semantic Primes in Formal Languages? *11th International Conference Distributed Computing and Artificial Intelligence (DCAI)*, 290:397–405, 2014.

[98] Johannes Fähndrich, Sebastian Ahrndt, and Sahin Albayrak. Formal Language Decomposition into Semantic Primes. *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, 3(8):56, 2014.

[99] Johannes Fähndrich, Sebastian Ahrndt, and Sahin Albayrak. Self-Explanation through Semantic Annotation and (automated) Ontology Creation: A Survey. In *10th International Symposium Advances in Artificial Intelligence and Applications*, pages 1–15. ACM, 2015.

[100] Johannes Fähndrich, Tobias Küster, and Nils Masuch. Semantic Service Management and Orchestration for Adaptive and Evolving Processes. *International Journal on Advances in Internet Technology*, 9(4):75–88, 2016.

[101] Johannes Fähndrich, Tobias Küster, and Nils Masuch. Semantic Service Management for Enabling Adaptive and Evolving Processes. In *The 11th International Conference on Internet and Web Applications and Services*, pages 46–53, Valencia, 2016.

[102] Johannes Fähndrich, Nils Masuch, Lars Borchert, and Sahin Albayrak. Learning Mechanisms on OWL-S Service Descriptions for Automated Action Selection. In *16th Conference on Autonomous Agents and MultiAgent Systems*, pages 41–57, 2017.

[103] Johannes Fähndrich, Nils Masuch, Hilmi Yildirim, and Sahin Albayrak. Towards Automated Service Matchmaking and Planning for Multi-Agent Systems with OWL-S - Approach and Challenges. In *Service-Oriented Computing – ICSOC 2013 Workshops*, pages 240–247. Springer International Publishing, Cham, 2014.

[104] Johannes Fähndrich, Sabine Weber, and Sebastian Ahrndt. Design and Use of a Semantic Similarity Measure for Interoperability Among Agents. In *Multiagent System Technologies*, pages 41–57. Springer International Publishing, 2016.

[105] Yong-Yi FanJiang and Yang Syu. Semantic-based automatic service composition with functional and non-functional requirements in design time: A genetic algorithm approach. *Information and Software Technology*, 56(3):352–373, 2014.

[106] Jean-Mari Favre. Towards a basic theory to model model driven engineering. In *Workshop in Software Model Engineering, WiSME*, pages 262–271, 2004.

[107] Jean-Marie Favre. Foundations of meta-pyramids: languages vs. metamodels. In *Episode II. Story of Thotus the Baboon, Procs. Dagstuhl Seminar*, 2004.

[108] Matthias Felleisen. On the expressive power of programming languages. *Science of Computer Programming*, 17(1-3):35–75, 1991.

[109] Dieter Fensel, Federico Facca, Elena Simperl, and Ioan Toma. *Semantic Web Services*. Springer Science & Business Media, Berlin, Heidelberg, 2011.

[110] Jacques Ferber. *Multi-agent systems - an introduction to distributed artificial intelligence.*, volume 1. Addison Wesley Longman, 1999.

[111] Innes Ferguson. Touring Machines: autonomous agents with attitudes. *Computer*, 25(5):51–55, 1992.

[112] Alan Fern, Roni Khardon, and Prasad Tadepalli. The first learning track of the international planning competition. *Machine Learning*, 84(1-2):81–107, 2011.

[113] Juan Fernández-Olivares, Tomás Garzón, Luis Castillo, Óscar García-Pérez, and Francisco Palao. A Middle-Ware for the Automated Composition and Invocation of Semantic Web Services Based on Temporal HTN Planning Techniques. *CAEPIA*, 4788(Chapter 8):70–79, 2007.

[114] Ludovic Ferrand and Boris New. Semantic and Associative Priming in the Mental Lexicon. *Mental lexicon: Some words to talk about words*, pages 25–43, 2003.

[115] Roy Fielding and Richard Taylor. Principled design of the modern Web architecture. *ACM Transactions on Internet Technology (TOIT)*, 2(2):115–150, 2002.

[116] Charles Fillmore, C Wooters, and C Baker. Building a Large Lexical Databank which Provides Deep Semantics. In *Proceedings of the Pacific Asian conference on language, information and computation*, pages 3–25, 2001.

[117] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing search in context: the concept revisited. *ACM Trans. Inf. Syst. ()*, 20(1):116–131, 2002.

[118] John Firth. A Synopsis of Linguistic Theory, 1930-1955. *Studies in linguistic analysis*, 1952-59:1–32, 1957.

[119] Wolf Fischer and Bernhard Bauer. Ontology based Spreading Activation for NLP related Scenarios. In *The Fifth International Conference on Advances in Semantic Processing (SEMAPRO)*, pages 56–61, 2011.

[120] Jerry Fodor and Zenon Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1):3–71, 1988.

[121] Howard Foster, S Uchitel, J Magee, and J Kramer. An Integrated Workbench for Model-Based Engineering of Service Compositions. *Services Computing, IEEE Transactions on*, 3(2):131–144, 2010.

[122] Maria Fox and Derek Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. 20:61–124, 2003.

[123] Valentina Franzoni, Marco Mencacci, Paolo Mengoni, and Alfredo Milani. Semantic Heuristic Search in Collaborative Networks: Measures and Contexts. In *IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 1, pages 141–148. IEEE, 2014.

[124] Gottlob Frege. *Funktion, Begriff, Bedeutung.* Vandenhoeck und Ruprecht, Göttingen, 1892.

[125] Stefan Fricke. *Symbolische Künstliche Intelligenz Probleme repräsentieren und lösen (lassen)* , volume 1. 2016.

[126] Keita Fujii and Tatsuya Suda. Component service model with semantics (CoSMoS): a new component model for dynamic service composition. In *International Symposium on Applications and the Internet Workshops*, pages 348–354. IEEE, 2004.

[127] Keita Fujii and Tatsuya Suda. Semantics-based context-aware dynamic service composition. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):1–31, 2009.

[128] Ulrich Furbach and Claudia Schon. Commonsense Reasoning Meets Theorem Proving. In *Agents and Computational Autonomy*, pages 3–17. Springer International Publishing, Cham, 2016.

[129] Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. volume 7, pages 1606–1611, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[130] Jian-Bo Gao, Bao-Wen Zhang, and Xiao-Hua Chen. A WordNet-based semantic similarity measurement combining edge-counting and information content theory. *Engineering Applications of Artificial Intelligence*, 39:80–88, 2015.

[131] Dedre Gentner and Susan Goldin-Meadow. *Language in mind: Advances in the study of language and thought*. MIT Press, 2003.

[132] Malik Ghallab, Adele Howe, Drew Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL – The Planning Domain Definition Language. Technical report, Yale Center for Computational Vision and Control, 1998.

[133] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann, San Francisco, CA, USA, 2004.

[134] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann, San Francisco, CA, USA, 2008.

[135] Malik Ghallab, Dana Nau, and Paolo Traverso. The actor's view of automated planning and acting: A position paper. *Artificial Intelligence*, 208:1–17, 2014.

[136] Thomas Gil. *Strukturen sprachlicher Bedeutung*. Wehrhan Verlag, Hannover, 2011.

[137] Cliff Goddard. The universal syntax of semantic primitives. *Language Sciences*, 19(3):197–207, 1996.

[138] Cliff Goddard. The semantics of coming and going. *Pragmatics*, 7(2):147–162, 1997.

[139] Cliff Goddard. A culture-neutral metalanguage for mental state concepts. In *Mental States*, volume 2, pages 11–35. John Benjamins Publishing Company, Amsterdam, 2008.

[140] Cliff Goddard. *Cross-linguistic Semantics*. John Benjamins Publishing, 2008.

[141] Cliff Goddard. Semantic molecules and semantic complexity (with special reference to environmental molecules). *Review of Cognitive Linguistics*, 8(1):123–155, 2010.

[142] Cliff Goddard. The Natural Semantic Metalanguage Approach. *The Oxford handbook of linguistic analysis*, pages 459–484, 2010.

[143] Cliff Goddard. Semantic primes, semantic molecules, semantic templates: Key concepts in the NSM approach to lexical typology. *Linguistics*, 50(3):711–743, 2012.

[144] Cliff Goddard and Anna Wierzbicka. *Semantic and Lexical Universals*. Theory and Empirical Findings. John Benjamins Publishing, 1994.

[145] Cliff Goddard and Anna Wierzbicka. *Semantic and lexical universals: Theory and empirical findings*, volume 25. John Benjamins Publishing Company, 1994.

[146] Edited Goddard. Cross-Linguistic Semantics (Studies in Language Companion Series). Technical report, 2008.

[147] Wael Gomaa and Aly Fahmy. A Survey of Text Similarity Approaches. *International Journal of Computer Applications*, 68(13):13–18, 2013.

[148] Bernardo Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. OWL 2: The next step for OWL. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4):309–322, 2008.

[149] Georgia Green. *Pragmatics and Natural Language Understanding*. Psychology Press, 1996.

[150] Peter Gregory and Alan Lindsay. Domain model acquisition in domains with action costs. In *Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling*, pages 149–157. AAAI Press, 2016.

[151] Herbert Grice. Meaning. *The Philosophical Review*, 66(3):377–388, 1957.

[152] Tom Gruber. Ontology. *Encyclopedia of Database Systems*, 2008.

[153] Michael Grüninger, Richard Hull, and Sheila Mcilraith. A Short Overview of FLOWS: A First-Order Logic Ontology for Web Services. *IEEE Data Engineering Bulletin*, 31(3):3–7, 2008.

[154] Nicola Guarino. Formal ontology, conceptual analysis and knowledge representation. *International Journal of Human-Computer Studies*, 43(5-6):625–640, 1995.

[155] Sebastien Harispe, Sylvie Ranwez, Stefan Janaqi, and Jacky Montmain. Semantic Measures for the Comparison of Units of Language, Concepts or Entities from Text and Knowledge Base Analysis. *Computing Research Repository (CoRR)*, pages 1–102, 2013.

[156] Zellig Harris. Distributional Structure. *Word*, 10(2-3):146–162, 1954.

[157] Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Association for the Advancement of Artificial Intelligence (AAAI)*, volume 7, 2007.

[158] Patrik Haslum and Hector Geffner. Admissible heuristics for optimal planning. In *Proceedings of the 5th International Conference of AI Planning Systems (AIPS)*, pages 140–149, 2000.

[159] Ourania Hatzi, Dimitris Vrakas, Nick Bassiliades, Anagnostopoulos Dimosthenis, and Ioannis Vlahavas. The PORSCE II framework: Using AI planning for automated semantic web service composition. *The Knowledge Engineering Review*, 28(02):137–156, 2013.

[160] Barbara Hayes-Roth. An architecture for adaptive intelligent systems. *Artificial Intelligence*, 72(12):329–365, 1995.

[161] David Heckerman, Eric Horvitz, Stanford University. Computer Science Dept. Knowledge Systems Laboratory, and Bharat Nathwani. Toward Normative Expert Systems: Part I, the Pathfinder Project, 1992.

[162] Irene Heim. Decomposing antonyms. In *Proceedings of Sinn und Bedeutung*, volume 12, pages 212–225, 2008.

[163] Malte Helmert. A Planning Heuristic Based on Causal Graph Analysis. *International Conference on Automated Planning and Scheduling (ICAPS)*, 4:161–170, 2004.

[164] Malte Helmert, Gabriel Röger, and Erez Karpas. Fast Downward Stone Soup: A baseline for building planner portfolios. In Erez Karpas, Sergio Jimenéz Celorrio, and Subbarao Kambhampati, editors, *International Conference on Automated Planning and Scheduling*, pages 28–35. International Conference on Automated Planning and Scheduling, Proceedings of the ICAPS-2011 Workshop on Planning and Learning (PAL 2011), 2011.

[165] James Hendler. Integrating marker-passing and problem-solving: A spreading activation approach to improved choice in planning. Hillsdale, N.J. : Lawrence Erlbaum Associates, 1988.

[166] Ehsan Hessami, Faribourz Mahmoud, and Amir Jadidinejad. Unsupervised Graph-based Word Sense Disambiguation Using lexical relation of WordNet. *International Journal of Computer Science*, 8(3), 2011.

[167] Graeme Hirst. *Semantic Interpretation and the Resolution of Ambiguity*. Studies in Natural Language Processing. Cambridge University Press, 1992.

[168] Graeme Hirst and David St-Onge. Lexical chains as representations of context for the detection and correction of malapropisms. *WordNet An Electronic Lexical Database*, pages 305–332, 1998.

[169] Sir Charles Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[170] Johannes Hoffart, Fabian Suchanek, Klaus Berberich, and Gerhard Weikum. YAGO2 - A spatially and temporally enhanced knowledge base from Wikipedia. *Artif. Intell.*, 194:28–61, 2013.

[171] Jorg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heristic search. *Ammerican Association for Artificial Intelligence (AAAI) AI Magazine*, 14(1):253–302, 2001.

[172] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278, 2004.

[173] Matthew Horridge and Peter Patel-Schneider. OWL 2 Web Ontology Language: Manchester Syntax. W3C Working Group Note. Technical report, 2009.

[174] Ian Horrocks and Peter Patel-Schneider. A proposal for an owl rules language. In *Proceedings of the 13th international conference on World Wide Web (WWW)*, pages 723–731, New York, New York, USA, 2004. ACM Press.

[175] Ian Horrocks, Peter Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Technical report, 2004.

[176] Eric Huang, Richard Socher, Christopher Manning, and Andrew Ng. Improving word representations via global context and multiple word prototypes. *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, 1:873–882, 2012.

[177] Thad Hughes and Daniel Ramage. Lexical Semantic Relatedness with Random Graph Walks. *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 581–589, 2007.

[178] Ignacio Iacobacci, Mohammad Pilehvar, and Roberto Navigli. Embeddings for Word Sense Disambiguation: An Evaluation Study. In *Proceedings of the th Annual Meeting of the Association for Computational Linguistics*, pages 897–907, 2016.

[179] Nancy Ide and Jean Veronis. Introduction to the special issue on word sense disambiguation: the state of the art. *Computational Linguistics*, 24(1):2–40, 1998.

[180] Aminul Islam and Diana Inkpen. Semantic text similarity using corpus-based word similarity and string similarity. *ACM Transactions on Knowledge Discovery from Data*, 2(2):1–25, 2008.

[181] Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, 37(142):547–579, 1901.

[182] Ray Jackendoff. Parts and boundaries. *Cognition*, 41(1-3):9–45, 1991.

[183] Mario Jarmasz and Stan Szpakowicz. Roget ' s Thesaurus and Semantic Similarity Roget ' s Thesaurus Relations as a Measure of Semantic Distance. *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2003)*, pages 212–219, 2003.

[184] Nicholas Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.

[185] Jay Jiang and David Conrath. Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy. 1997.

[186] Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011.

[187] Subbarao Kambhampati and James Hendler. A Validation-Structure-Based Theory of Plan Modification and Reuse. *Artif. Intell.*, 55(2-3):193–258, 1992.

[188] Hans Kamp and Uwe Reyle. *From discourse to logic: Introduction to modeltheoretic semantics of natural language, formal logic and discourse representation theory*, volume 42. Kluwer Academic Publischers, 2013.

[189] Thomas Karbe. *Context and Context Management*. PhD thesis, Shaker Verlag, 2014. Technische Universtität Berlin.

[190] Erez Karpas and C Domshlak. Cost-Optimal Planning with Landmarks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1728–1733, 2009.

[191] Michael Katz and Carmel Domshlak. Structural Patterns Heuristics via Fork Decomposition. *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 182–189, 2008.

[192] Alistair Kennedy and Stan Szpakowicz. Evaluation of automatic updates of Roget's Thesaurus. *Journal of Language Modelling*, 2(1):1–49, 2014.

[193] Jeffrey Kephart and David Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, 2003.

[194] Emil Keyder and Héctor Geffner. Heuristics for Planning with Action Costs. *CAEPIA*, 4788(Chapter 15):140–149, 2007.

[195] Matthias Klusch, Benedikt Fries, and Katia Sycara. OWLS-MX: A hybrid Semantic Web service matchmaker for OWL-S services. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(2):121–133, 2009.

[196] Matthias Klusch, Andreas Gerber, and Marcus Schmidt. Semantic web service composition planning with owls-xplan. In *International Conference on Web Intelligence and Intelligent Agent Technology Workshops*, pages 55–62, 2005.

[197] Matthias Klusch and Patrick Kapahnke. The iSeM matchmaker: A flexible approach for adaptive hybrid semantic service selection. *Web Semantics: Science, Services and Agents on the World Wide Web*, 15:1–14, 2012.

[198] Matthias Klusch, Patrick Kapahnke, Stefan Schulte, Freddy Lecue, and Abraham Bernstein. Semantic Web Service Search: A Brief Survey. *KI - Künstliche Intelligenz*, 30(2):139–147, 2015.

[199] Matthias Klusch, Patrick Kapahnke, and Ingo Zinnikus. SAWSDL-MX2: A Machine-Learning Approach for Integrating Semantic Web Service Matchmaking Variants. In *2009 IEEE International Conference on Web Services (ICWS)*, pages 335–342. IEEE Computer Society, IEEE, 2009.

[200] Matthias Klusch, Ulrich Küster, Alain Leger, David Martin, and Massimo Paolucci. 5*th* International Semantic Service Selection Contest - Performance Evaluation of Semantic Service Matchmakers. In *Semantic web services*, pages 17–34. Springer, 2012.

[201] Mattias Klusch and Andreas Gerber. Fast Composition Planning of OWL-S Services and Application. In *Fast Composition Planning of OWL-S Services and Application*, pages 181–190. IEEE, 2006.

[202] Craig Knoblock. *Generating Abstraction Hierarchies: An Automated Approach to Reducing Search in Planning*. Springer Science & Business Media, LLC, 1993.

[203] Tom König. Untersuchung semantischer Distanzmaße auf der Grundlage von Aktivierungsausbreitung über ontologiebasierte Hypergraphen. Bachelors thesis, 2017. Technische Universtität Berlin.

[204] Thomas Konnerth. *An Agent-Based Approach to Service-Oriented Architectures*. PhD thesis, 2012. Technische Universtität Berlin.

[205] Jacek Kopecký, Tomas Vitvar, Carine Bournez, and Joel Farrell. SAWSDL: Semantic Annotations for WSDL and XML Schema. *IEEE Internet Computing ()*, 11(6):60–67, 2007.

[206] Richard Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189–211, 1990.

[207] Richard Korf and Ariel Felner. Recent Progress in Heuristic Search - A Case Study of the Four-Peg Towers of Hanoi Problem. *International Joint Conference on Artificial Intelligence*, pages 2324–2329, 2007.

[208] András Kornai and Marcus Kracht. Lexical Semantics and Model Theory: Together at Last? In *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015)*, pages 51–61, Stroudsburg, PA, USA, 2015. Association for Computational Linguistics.

[209] Marcus Kracht. Are logical languages compositional? *Studia Logica. An International Journal for Symbolic Logic*, 101(6):1319–1340, 2013.

[210] Saul Kripke. Naming and Necessity. In *Semantics of Natural Language*, pages 253–355. Springer Netherlands, Dordrecht, 1973.

[211] Patricia Kuhl, Barbara Conboy, Denise Padden, Tobey Nelson, and Jessica Pruitt. Early speech perception and later language development: Implications for the critical period. *Language Learning and Development*, 1(3):237–264, 2005.

[212] Ram Kumar, Shailesh Jaloree, and Thakur. Developing Context Ontology using Information Extraction. *nternational Journal of Computer Science and Information Security (IJCSIS)*, 14(3), 2016.

[213] Ray Kurzweil. *The singularity is near*. Gerald Duckworth & Co, 2005.

[214] Ulrich Küster, Birgitta König-Ries, Michael Klein, and Mirco Stern. Diane: A matchmaking-centered framework for automated service discovery, composition, binding, and invocation on the web. *International Journal of Electronic Commerce*, 12(2):41–68, 2007.

[215] Ugur Kuter, Evren Sirin, Dana Nau, Bijan Parsia, and James Hendler. Information Gathering During Planning for Web Service Composition. In *International Semantic Web Conference (ISWC)*, pages 335–349. Elsevier, 2005.

[216] Mehmet Kuzu and Nihan Cicekli. Dynamic planning approach to automated web service composition. *Applied Intelligence*, 36(1):1–28, 2012.

[217] William Labov. The boundaries of words and their meanings. *New Ways of analyzing variation in English*, pages 340–373, 1973.

[218] George Lakoff and Mark Johnson. *Metaphors We Live By*. University of Chicago Press, 2008.

[219] Steffen Lamparter and Anupriya Ankolekar. Automated selection of configurable web services. In *Internationale Tagung Wirtschaftsinformatik*, pages 441–458, 2007.

[220] Ora Lassila and Ralph Swick. Resource description framework (RDF) model and syntax specification, 1999.

[221] Juan Lastra-Díaz and Ana García-Serrano. A novel family of IC-based similarity measures with a detailed experimental survey on WordNet. *Engineering Applications of Artificial Intelligence*, 46:140–153, 2015.

[222] Steven LaValle. *Planning algorithms*. Cambridge University Press, Cambridge, Cambridge, 2006.

[223] Claudia Leacock, George Miller, and Martin Chodorow. Using corpus statistics and WordNet relations for sense identification. *Computational Linguistics*, 24(1):147–165, 1998.

[224] David Leake. Goal-based explanation evaluation. *Cognitive science*, 15(4):509–545, 1991.

[225] David Leake. *Evaluating Explanations A Content Theory*. Psychology Press, 1992.

[226] Freddy Lecue, Alain Leger, and Alexandre Delteil. DL Reasoning and AI Planning for Web Service Composition. In *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pages 445–453. IEEE, 2008.

[227] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[228] Marta Lenartowicz. Creatures of the semiosphere: A problematic third party in the 'humans plus technology' cognitive architecture of the future global superintelligence. *Technological Forecasting and Social Change*, 114:35–42, 2017.

[229] Michael Lesk. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual international conference on Systems documentation*, how to tell a pine code from an ice cream cone, pages 24–26, New York, New York, USA, 1986. ACM.

[230] Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, volume 46, pages 552–561, 2011.

[231] Yuhua Li, David McLean, Zuhair Bandar, James O'shea, and Keeley Crockett. Sentence similarity based on semantic nets and corpus statistics. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1138–1150, 2006.

[232] Dekang Lin. An information-theoretic definition of similarity. In *International Conference on Machine Learning (ICML)*, volume 98, pages 296–304, 1998.

[233] Dekang Lin, Shaojun Zhao, Lijuan Qin, and Ming Zhou. Identifying synonyms among distributionally similar words. *IJCAI International Joint Conference on Artificial Intelligence*, (4):1492–1493, 2003.

[234] Robert Lindsay. *Understanding Understanding*. Natural and Artificial Intelligence. CreateSpace Independent Publishing Platform, 2012.

[235] Angelika Linke, Markus Nussbaumer, and Paul Protmann. *Studienbuch Linguistik*. Reihe Germanistische Linguisitk, 5. erweiterte Auflage. De Gruyter, 2004.

[236] Nir Lipovetzky and Hector Geffner. Best-First Width Search: Exploration and Exploitation in Classical Planning. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 3590–3569, 2017.

[237] Baoding Liu. Uncertain entailment and modus ponens in the framework of uncertain logic. *Journal of Uncertain Systems*, 3(4):243–251, 2009.

[238] Hugo Liu and Push Singh. ConceptNet - A Practical Commonsense Reasoning Tool-Kit. *BT technology journal*, 22(4):211–226, 2004.

[239] Wei Liu, Albert Weichselbraun, Arno Scharl, and Elizabeth Chang. Semi-automatic ontology extension using spreading activation. *Journal of Universal Knowledge Management*, 1(1):50–58, 2005.

[240] Sebastian Löbner. *Semantik: Eine Einführung*. Walter de Gruyter GmbH & Co KG, 2013.

[241] Nikolaos Loutas, Vassilios Peristeras, and Konstantinos Tarabanis. Towards a reference service model for the Web of Services. *Data & Knowledge Engineering*, 70(9):753–774, 2011.

[242] Marco Luetzenberger, Tobias Kuester, Nils Masuch, and Johannes Fähndrich. Multi-Agent System in Practice – When Research Meets Reality. In John Thangarajah, Karl Tuyls, Catholijn Jonker, and Stacy Marsella, editors, *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), Singapore*, pages 796–805. IFAAMAS, 2016.

[243] Pascal Lukanek. Ein auf Marker-Passing basierender Word-Sense-Disambiguation Ansatz. Bachelors thesis, Berlin, 2017. Technische Universtität Berlin.

[244] Thang Luong, Richard Socher, and Christopher Manning. Better word representations with recursive neural networks for morphology. *The SIGNLL Conference on Computational Natural Language Learning*, pages 104–113, 2013.

[245] Alexander Maedche. Ontology Learning for the Semantic Web. *IEEE Intelligent systems*, 16(2):72–79, 2001.

[246] Alexander Maedche and Steffen Staab. Learning ontologies for the semantic web. In *Proceedings of the Second International Conference on Semantic Web*, pages 51–60. CEUR-WS.org, 2001.

[247] Pattie Maes. How to do the Right Thing. *Connection Science*, 1(3):291–323, 1989.

[248] Farzaneh Mahdisoltani, Joanna Biega, and Fabian Suchanek. YAGO3 - A Knowledge Base from Multilingual Wikipedias. *CIDR*, 2015.

[249] Bernd Mahr. Gegenstand und Kontext - Eine Theorie der Auffassung. In K. Eyferth, Bernd Mahr, R. Posner, and F. Wysotzki, editors, *Prinzipien der Kontextualisierung*. Technische Universität Berlin, 1997.

[250] Bernd Mahr. Intentionality and modeling of conception. Technical report, 2010.

[251] Arun Majumdar, John Sowa, and John Stewart. Pursuing the Goal of Language Understanding. In *Conceptual Structures: Knowledge Visualization and Reasoning*, pages 21–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[252] Florian Marienwald. How Does the Interpretation of Word Relations Help Word Sense Disambiguation Via a Marker Passing Approach? Bachelors thesis, 2017.

[253] Liana Marinescu and Andrew Coles. Heuristic guidance for forward-chaining planning with numeric uncertainty. In *Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling (ICAPS)*, pages 230–234. AAAI Press, 2016.

[254] George Markou and Ioannis Refanidis. Non-deterministic planning methods for automated web service composition. *Artificial Intelligence Research*, 5(1):14, 2016.

[255] David Martin, Massimo Paolucci, Sheila McIlraith, Mark Burstein, Drew McDermott, Deborah McGuinness, Bijan Parsia, Terry Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, and Katia Sycara. Bringing Semantics to Web Services: The OWL-S Approach. In *Semantic Web Services and Web Process Composition*, pages 26–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[256] Cynthia Matuszek, John Cabral, Michael Witbrock, and John DeOliveira. An Introduction to the Syntax and Content of Cyc. In *AAAI Spring Symposium - Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, 2006.

[257] Sylvester Mawson and Peter Roget. Roget's international thesaurus. 1911.

[258] John McCarthy. Notes on formalizing context. Technical report, 1993.

[259] Deborah McGuinness, Frank Van Harmelen, and others. OWL web ontology language overview. *W3C recommendation*, 10(2004-03):10, 2004.

[260] Anupam Mediratta and Biplav Srivastava. Applying planning in composition of web services with a user-driven contingent planner. Technical report, IBM Research Division IBM India Research Lab, 2006.

[261] Igor Mel čuk. Explanatory Combinatorial Dictionary. *Open problems in linguistics and lexicography*, pages 225–355, 2006.

[262] Igor Mel'čuk and Alain Polguère. A Formal Lexicon in the Meaning-Text Theory (or How to Do Lexica with Words). *Computational Linguistics*, 13(3-4):261–275, 1987.

[263] Christian Meyer and Iryna Gurevych. Wiktionary: A new rival for expert-built lexicons? Exploring the possibilities of collaborative lexicography. In *Electronic Lexicography*, pages 259–292. Oxford University Press, 2012.

[264] Harald Meyer and Mathias Weske. Automated Service Composition Using Heuristic Search. In *International Conference on Business Process Management*, volume 4102, pages 81–96. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[265] Loizos Michael. Jumping to Conclusions. In *Proceedings of the 2015 International Conference on Defeasible and Ampliative Reasoning*, volume 1423, pages 43–49, Buenos Aires, 2015.

[266] Zbigniew Michalewicz and David Fogel. *How to Solve It: Modern Heuristics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[267] Rada Mihalcea, Timothy Chklovski, and Adam Kilgarriff. The Senseval-3 English lexical sample task. In *Proceedings of Senseval-3, the third international workshop on the evaluation of systems for the semantic analysis of text*. Association for Computational Linguistics, 2004.

[268] Rada Mihalcea, Courtney Corley, and Carlo Strapparava. Corpus-based and Knowledge-based Measures of Text Semantic Similarity. *Association for the Advancement of Artificial Intelligence*, pages 775–780, 2006.

[269] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv.org*, page arXiv:1301.3781, 2013.

[270] George Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[271] George Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller. Introduction to WordNet: An On-line Lexical Database*. *International Journal of Lexicography*, 3(4):235–244, 1989.

[272] Joaquin Miller and Jishnu Mukerji. Model Driven Architecture (MDA). Technical report, 2001.

[273] Marvin Minsky. Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI Magazine*, 12(2):34, 1991.

[274] Sonam Mittal and Guarav Sahu. Ontology Learning. *International Research Journal of Engineering and Technology (IRJET)*, 3(10), 2016.

[275] Charles Morris. *Foundations of the Theory of Signs*. University of Chicago Press, 1938.

[276] Erik Mueller. *Commonsense Reasoning*. An Event Calculus Based Approach. Morgan Kaufmann, 2014.

[277] Jörg Müller and Markus Pischel. The agent architecture InteRRaP : concept and application. Technical report, 2011.

[278] Christian Müller-Schloer, Christoph Schmeck, and Hartmut Ungerer. *Organic Computing*. Springer-Verlag, 2004.

[279] Christian Müller-Schloer and Hartmut Schmeck. Organic Computing: A Grand Challenge for Mastering Complex Systems. *IT – Information Technology*, 52(3):135–141, 2010.

[280] Letitia Naigles, Anne Fowler, and Atessa Helm. Developmental shifts in the construction of verb meanings. *Cognitive Development*, 7(4):403–427, 1992.

[281] Dana Nau, Malik Ghallab, and Paolo Traverso. Blended Planning and Acting: Preliminary Approach, Research Challenges. *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 4047–4051, 2015.

[282] Roberto Navigli. Word sense disambiguation: A survey. *ACM Computing Surveys (CSUR)*, 41(2):1–69, 2009.

[283] Roberto Navigli and Simone Ponzetto. BabelNet - The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence*, 193:217–250, 2012.

[284] Srinivas Nedunuri, William Cook, and Douglas Smith. Cost-based learning for planning. In *International Conference on Automated Planning and Scheduling*, pages 68–75, 2011.

[285] Nils Nilsson and Richard Fikes. Problem-Solving Methodes in Artificial Intelligence. *Artificial Intelligence*, 2(3-4):189–208, 1971.

[286] Peter Norvig. Marker Passing as a Weak Method for Text Inferencing. *Cognitive science*, 13(4):569–620, 2010.

[287] Vilém Novák. Antonyms and linguistic quantiers in fuzzy logic. *Journal of Web Seamantics*, 124(3):335–351, 2001.

[288] Natasha Noy, Alan Rector, Pat Hayes, and Chris Welty. Defining N-ary Relations on the Semantic Web. Technical report, 2006.

[289] Ervin Nutter. Epistemology. *Encyclopedia of Artificial Intelligence*, 1:460–468, 1987.

[290] Phillipa Oaks, Arthur ter Hofstede, and David Edmond. Capabilities: Describing What Services Can Do. In *Service-Oriented Computing - ICSOC 2003*, pages 1–16. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[291] Charles Ogden and Ivor Richards. *The Meaning of Meaning*. Harcourt Brace Jovanovich, Publishers, 1923.

[292] Seog-Chan Oh, Ju-Yeon Lee, Seon-Hwa Cheong, Soo-Min Lim, Min-Woo Kim, Sang-Seok Lee, Jin-Bum Park, Sang-Do Noh, and Mye Sohn. WSPR*: Web-Service Planner Augmented with A* Algorithm. In *EEE Conference on Commerce and Enterprise Computing*, pages 515–518. IEEE, 2009.

[293] Cagla Okutan and Nihan Cicekli. A monolithic approach to automated composition of semantic web services with the Event Calculus. *Knowledge-Based Systems*, 23(5):440–454, 2010.

[294] Jesús Oliva, José Serrano, María del Castillo, and Ángel Iglesias. SyMSS: A syntax-based measure for short-text semantic similarity. *Data & Knowledge Engineering*, 70(4):390–405, 2011.

[295] Michael Öllinger and Albrecht von Müller. Search and Coherence-Building in Intuition and Insight Problem Solving. *Frontiers in psychology*, 8:827, 2017.

[296] Charles Osgood. The nature and measurement of meaning. *Psychological bulletin*, 49(3):197–237, 1952.

[297] Charles Osgood, George Suci, and Percy Tannenbaum. *The Measurement of Meaning*. University of Illinois Press, 1957.

[298] Ibrahim Osman and Gilbert Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63(5):511–623, 1996.

[299] Petros Papadopoulos, Huaglory Tianfield, David Moffat, and Peter Barrie. Decentralized multi-agent service composition. *Multiagent and Grid Systems An International Journal*, 9(1):45–100, 2013.

[300] Evan Patterson. Knowledge Representation in Bicategories of Relations. Technical report, 2017.

[301] Karalyn Patterson, Peter Nestor, and Timothy Rogers. Where do you know what you know? The representation of semantic knowledge in the human brain. *Nature Reviews Neuroscience*, 8(12):976–987, 2007.

[302] Siddharth Patwardhan and Ted Pedersen. Using WordNet-based context vectors to estimate the semantic relatedness of concepts. In *Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linquistics*, volume 1501, pages 1–8, 2006.

[303] Christopher Peacocke. *A Study of Concepts.* The MIT Press, 1992.

[304] Judea Pearl. Heuristics. Intelligent search strategies for computer problem solving. *The Addison-Wesley Series in Artificial Intelligence*, 1985.

[305] Joachim Peer. A PDDL based tool for automatic web service composition. *Principles and Practice of Semantic Web Reasoning*, 3208(Chapter 11):149–163, 2004.

[306] Carl Petri. *Kommunikation mit Automaten.* PhD thesis, Technische Hochschule Darmstadt, 1962.

[307] Giulio Petrucci. Information Extraction for Learning Expressive Ontologies. In *Agents and Computational Autonomy*, pages 740–750. Springer International Publishing, Cham, 2015.

[308] Mohammad Pilehvar and Roberto Navigli. From senses to texts: An all-in-one graph-based approach for measuring semantic similarity. *Artificial Intelligence*, 228:95–128, 2015.

[309] Martha Pollack and Marc Ringuette. Introducing the Tileworld: Experimentally Evaluating Agent Architectures. In *Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence*, volume 90, pages 183–189, 1990.

[310] Julie Porteous, Laura Sebastia, and Joerg Hoffmann. On the Extraction, Ordering, and Usage of Landmarks in Planning. In *Sixth European Conference on Planning*, pages 174–182, 2014.

[311] Uta Priss. Classification of meronymy by methods of relational concept analysis. In *Midwest Artificial Intelligence and Cognitive Science Conference*. unknown, 1996.

[312] Paul Procter. *Longman dictionary of contemporary English*. Harlow [Eng.] : Longman, 1978.

[313] Uwe Quasthoff, Dirk Goldhahn, and Thomas Eckart. Building Large Resources for Text Mining: The Leipzig Corpora Collection. In *Text Mining*, pages 3–24. Springer International Publishing, Cham, 2014.

[314] Ross Quillian. *Semantic Memory*. PhD thesis, Bolt Beranek And Newman Inc. Unpublished doctoral dissertation, Carnegie Institute of Technology. (Reprinted in part in M. Minsky [Ed.], Semantic information processing. Cambridge, Mass.: M.I.T. Press, 1968.). Carnegie Institute of Technology.

[315] Rodrigo Quian Quiroga. Concept cells: the building blocks of declarative memory functions. *Nature reviews. Neuroscience*, 13(8):587, 2012.

[316] Jeroen Raaijmakers and Richard Shiffrin. Search of associative memory. *Psychological Review*, 88(2):93–134, 1981.

[317] Roy Rada, Hafedh Mili, Ellen Bicknell, and Maria Blettner. Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(1):17–30, 1989.

[318] Kira Radinsky, Eugene Agichtein, Evgeniy Gabrilovich, and Shaul Markovitch. A word at a time: computing word relatedness using temporal semantic analysis. In *Proceedings of the 20th international conference on World wide web*, computing word relatedness using temporal semantic analysis, pages 337–346, New York, New York, USA, 2011. ACM.

[319] Nairán Ramírez-Esparza, Adrián García-Sierra, and Patricia Kuhl. Look who's talking: speech style and social context in language input to infants are linked to concurrent and future speech development. *Developmental science*, 17(6):880–891, 2014.

[320] Anand Rao and Michael Georgeff. BDI Agents: From Theory to Practice. In Victor Lesser and Les Gasser, editors, *Proceedings of the First International Conference on Mutiagent Systems (ICMAS)*, pages 312–319. The MIT Press, 1995.

[321] Jinghai Rao, Peep Küngas, and Mihhail Matskin. Composition of semantic web services using linear logic theorem proving. *Information Systems*, 31(4-5):340–360, 2006.

[322] Daniel Ratner and Manfred Warmuth. Finding a Shortest Solution for the NxN Extension of the 15-Puzzle Is Intractable. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 168–172, 1986.

[323] Domenico Redavid, Stefano Ferilli, and Floriana Esposito. SWRL Rules Plan Encoding with OWL-S Composite Services. In *Agents and Computational Autonomy*, pages 476–482. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[324] Philip Resnik. Using Information Content to Evaluate Semantic Similarity in a Taxonomy. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, page 6, 1995.

[325] Silvia Richter, Malte Helmert, and Matthias Westphal. Landmarks Revisited. In *Association for the Advancement of Artificial Intelligence*, pages 975–982, 2008.

[326] Silvia Richter and Matthias Westphal. The LAMA Planner - Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research*, 39(1):127–177, 2014.

[327] Burghard Rieger. *Unscharfe Semantik*. Die empirische Analyse, quantitative Beschreibung, formale Repräsentation und prozedurale Modellierung vager Wortbedeutungen in Texten. Peter Lang, Frankfurt am Main, 1990.

[328] Nick Riemer. *The Routledge Handbook of Semantics*. Routledge, 2015.

[329] Christopher Riesbeck and Charles Martin. Direct Memory Access Parsing. *Experience, Memory, and Reasoning*, pages 209–226, 1986.

[330] Paul Robertson, Howard Shrobe, and Robert Laddaga. *Self-adaptive Software*, volume 1936 of *Revised Papers. Oxford, UK, April 17-19, 2000.* Springer Science & Business Media, Berlin, Heidelberg, 2001.

[331] Guillermo Rodríguez, Álvaro Soria, and Marcelo Campo. Artificial intelligence in service-oriented software design. *Engineering Applications of Artificial Intelligence*, 53(C):86–104, 2016.

[332] Pablo Rodriguez-Mier, Manuel Mucientes, and Manuel Lama. Automatic Web Service Composition with a Heuristic-Based Search Algorithm. In *2011 IEEE International Conference on Web Services (ICWS)*, pages 81–88. IEEE, 2011.

[333] Timothy Rogers, John Hodges, Karalyn Patterson, and Matthew Lambon Ralph. Object recognition under semantic impairment: The effects of conceptual regularities on perceptual decisions. *Language and Cognitive Processes*, 18(5-6):625–662, 2003.

[334] Chuitian Rong, Yasin Silva, and Chunqing Li. String similarity join with different similarity thresholds based on novel indexing techniques. *Frontiers of Computer Science*, 11(2):307–319, 2016.

[335] Louise Röska-Hardy. *Die Bedeutung in natürlicher Sprache*. Athenäums monografien: Philosophie. Athenäum Verlag GmbH, Frankfurt am Main, 1988.

[336] Herbert Rubenstein and John Goodenough. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633, 1965.

[337] Stuart Russel and Peter Norvig. *Artifical Intelligence: A Modern Approach*, volume 2. Prentice Hall, 2002.

[338] Stuart Russell and Peter Norvig. *Artificial Intelligence*. A Modern Approach, Global Edition. 2016.

[339] Luca Sabatucci and Massimo Cossentino. From Means-End Analysis to Proactive Means-End Reasoning. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 2–12. IEEE, 2015.

[340] Hadi Saboohi and Sameem Kareem. A Resemblance Study of Test Collections for World-altering Semantic Web Services. In *Proceedings of the International MultiConference of Engineers and Computer Scientists IMECS*, volume 1, pages 1–5, 2011.

[341] Saïd Salhi. *Heuristic Search: The Emerging Science of Problem Solving*. Springer, 2017.

[342] David Sánchez, Montserrat Batet, David Isern, and Aida Valls. Ontology-based semantic similarity: A new feature-based approach. *Expert Systems With Applications*, 39(9):7718–7728, 2012.

[343] Pete Sawyer, Nelly Bencomo, Jon Whittle, Emmanuel Letier, and Anthony Finkelstein. Requirements-Aware Systems: A Research Agenda for RE for Self-adaptive Systems. *2010 IEEE 18th International Conference on Requirements Engineering (RE)*, pages 95–103, 2010.

[344] Marco Sbodio, David Martin, and Claude Moulin. Discovering Semantic Web services using SPARQL and intelligent agents. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(4):310–328, 2010.

[345] Roger Schank and Robert Abelson. *Scripts, Plans, Goals, and Understanding* . An Inquiry into Human Knowledge Structures. 1977.

[346] Hartmut Schmeck. Organic computing - a new vision for distributed embedded systems. pages 201–203, 2005.

[347] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[348] David Schmidt. *Denotational Semantics*. A Methodology for Language Development. 1997.

[349] Hedda Schmidtke. Contextual Reasoning in Context-Aware Systems. In *Workshop Proceedings of the 8th International Conference on Intelligent Environments*, pages 82–93, 2012.

[350] Luc Schneider. How to Build a Foundational Ontology. In *Agents and Computational Autonomy*, pages 120–134. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[351] Nico Schneider. Quantifizierung semantischer Satzähnlichkeit basierend auf lexikalischer Deckomposition und Marker Passing. Bachelors thesis, 2017. Technische Universtität Berlin.

[352] Susan Schneider. Non-Reductive Physicalism and the Mind Problem. *Noûs*, 47(1):135–153, 2013.

[353] John Searle. Minds, brains, and programs. *Behavioral and brain sciences*, 3(3):417–424, 1980.

[354] David Shanks and Koen Lamberts. *Knowledge, Concepts, and Categories*. Psychology Press, 1997.

[355] Raj Sharman, Rajiv Kishore, and Ram Ramesh. *Ontologies: A Handbook of Principles, Concepts and Applications in Information Systems*. Springer Science + Business Media, LLC, New York, 2007.

[356] Lokendra Shastri and Venkat Ajjanagadde. From simple associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony. *Behavioral and brain sciences*, 16(03):417–451, 2010.

[357] Hui Shen, Razvan Bunescu, and Rada Mihalcea. Coarse to Fine Grained Sense Disambiguation in Wikipedia. In *Conference of the North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL HLT)*, pages 22–31, 2013.

[358] Pavel Shvaiko and Jérôme Euzenat. Ontology Matching: State of the Art and Future Challenges. *IEEE Transactions on knowledge and data engineering*, 25(1):158–176, 2013.

[359] Josefina Sierra-Santib ez. Heuristic planning: A declarative approach based on strategies for action selection. *Artificial Intelligence*, 153(1-2):307–337, 2004.

[360] Evren Sirin and Bijan Parsia. Planning for semantic web services. In *Semantic Web Services Workshop at 3rd International Semantic Web Conference*, pages 33–40, 2004.

[361] Barry Smith. Basic Concepts of Formal Ontology. *Formal Ontology in Information Systems*, pages 19–28, 1998.

[362] Edward Smith, Edward Shoben, and Lance Rips. Structure and process in semantic memory: A featural model for semantic decisions. *Psychological Review*, 81(3):214–241, 1974.

[363] Jeffrey Smith, Scott DeLoach, Mieczyslaw Kokar, and Ken Baclawski. Category theoretic approaches of representing precise UML semantics. In *Proceedings of the ECOOP Workshop on Defining Precise Semantics for UML*, 2000.

[364] Raja Sooriamurthi and David Leake. Towards Situated Explanation. In *Midwest Artificial Intelligence and Cognitive Science Conference*, page 1492, 1994.

[365] John Sowa. *Conceptual Structures*. Information Processing in Mind and Machine. Addison Wesley Publishing Company, 1984.

[366] John Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Thomson Learning, 2000.

[367] Biplav Srivastava and Subbarao Kambhampati. Scaling up Planning by Teasing out Resource Scheduling. *ECP*, 1809(Chapter 14):172–186, 1999.

[368] Lubomir Stanchev. Semantic Document Clustering Using a Similarity Graph. In *IEEE Tenth International Conference on Semantic Computing (ICSC)*, pages 1–8. IEEE, 2016.

[369] Luc Steels. Modeling the cultural evolution of language. *Physics of Life Reviews*, 8(4):339–356, 2011.

[370] Peter Strawson. *Meaning and Truth.* Logico-Linguistic Papers. London: Methuen, 1971.

[371] Fabian Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago. In *the 16th international conference*, pages 697–706, New York, New York, USA, 2007. ACM Press.

[372] Katia Sycara. Multiagent Systems. *AI Magazine*, 19(2):79–92, 1998.

[373] Katia Sycara, Matthias Klusch, Seth Widoff, and Jianguo Lu. Dynamic service matchmaking among agents in open information environments. *SIGMOD Record*, 28(1):47–53, 1999.

[374] Mohamed Taieb, Mohamed Ben Aouicha, and Abdelmajid Ben Hamadou. A new semantic relatedness measurement using WordNet features. *Knowledge and Information Systems*, 41(2):467–497, 2014.

[375] Mohamed Taieb, Mohamed Ben Aouicha, and Yosra Bourouis. FM3S: Features-Based Measure of Sentences Semantic Similarity. In *Hybrid Artificial Intelligent Systems*, pages 515–529, Cham, 2015. Springer International Publishing.

[376] Electra Tamani and Paraskevas Evripidou. *Combining Pragmatics and Intelligence in Semantic Web Service Discovery*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[377] Alfred Tarski. Der Wahrheitsbegriff in den formalisierten Sprachen. Leopoli, 1936.

[378] Alfred Tarski. The Semantic Conception of Truth: and the Foundations of Semantics. *Philosophy and phenomenological research*, 4(3):341, 1944.

[379] Kilian Thiel and Michael Berthold. Node Similarities from Spreading Activation. *Bisociative Knowledge Discovery*, 7250(Chapter 17):246–262, 2012.

[380] George Tsatsaronis, Iraklis Varlamis, and Kjetil Nørvåg. An Experimental Study on Unsupervised Graph-based Word Sense Disambiguation. *CICLing*, 6008(Chapter 16):184–198, 2010.

[381] George Tsatsaronis, Iraklis Varlamis, and Michalis Vazirgiannis. Word Sense Disambiguation with Semantic Networks. In *Text, Speech and Dialogue*, pages 219–226. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[382] George Tsatsaronis, Michalis Vazirgiannis, and Ion Androutsopoulos. Word Sense Disambiguation with Spreading Activation Networks Generated from Thesauri. In *International Joint Conference on Artificial Intelligence*, pages 1725–1730, 2007.

[383] Endel Tulving and Daniel Schacter. Priming and Human Memory Systems. *Science*, 247(4940):301–306, 1990.

[384] Alan Turing. Computing Machinery and Intelligence. *Mind*, LIX(236):433–460, 1950.

[385] Peter Turney. Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 491–502, 2002.

[386] Amos Tversky and Daniel Kahneman. Judgment under Uncertainty: Heuristics and Biases. 11(4157):141–162, 1974.

[387] Ricardo Usón and Pamela Faber. Decomposing semantic decomposition: Towards a semantic metalanguage in RRG. In *International Course and Conference on Role and Reference Grammar. Institute of Linguistics, Academia Sinica, Nankang, Taipei, Taiwan*, pages 1–28, 2005.

[388] Robert Van Valin Jr. An Overview of Role and Reference Grammar. Technical report, Department of General Linguistics Institute for Language and Information Heinrich-Heine-University Düsseldorf Universitätsstr. 1 40225 Düsseldorf, 2008.

[389] Jean Veronis and Nancy Ide. Word sense disambiguation with very large neural networks extracted from machine readable dictionaries. In *Proceedings of the 13th conference on Computational linguistics-Volume 2*, pages 389–394, Morristown, NJ, USA, 1990. Association for Computational Linguistics.

[390] Tomas Vitvar, Jacek Kopecký, Jana Viskova, and Dieter Fensel. WSMO-Lite Annotations for Web Services. In *The Semantic Web: Research and Applications*, pages 674–689. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[391] Stefan Voß. Meta-heuristics: The State of the Art. In *Workshop on Local Search for Planning and Scheduling*, pages 1–23. Springer, Berlin, Heidelberg, 2001.

[392] Denny Vrandečić and Markus Krötzsch. Wikidata. *Communications of the ACM*, 57(10):78–85, 2014.

[393] Florian Wagner, Fuyuki Ishikawa, and Shinichi Honiden. QoS-Aware Automatic Service Composition by Applying Functional Clustering. *International Conference on Web Services (ICWS)*, pages 89–96, 2011.

[394] Elizabeth Walter. *Cambridge advanced learner's dictionary*. Ernst Klett Sprachen, 2008.

[395] Fei-Yue Wang, Jun Zhang, Xinhu Zheng, Xiao Wang, Yong Yuan, Yiaoxiao Dai, Jie Zhang, and Liuqing Yang. Where does AlphaGo go: from church-turing thesis to AlphaGo thesis and beyond. *IEEE/CAA Journal of Automatica Sinica*, 3(2):113–120, 2016.

[396] Hongbing Wang, Xuan Zhou, Xiang Zhou, Weihong Liu, Wenyo Li, and Athman Bouguettaya. Adaptive Service Composition Based on Reinforcement Learning. In *International Conference on Service-Oriented Computing*, pages 92–107, 2010.

[397] Pengwei Wang, Zhijun Ding, Changjun Jiang, Mengchu Zhou, and Yuwei Zheng. Automatic Web Service Composition Based on Uncertainty Execution Effects. *IEEE Trans. Services Computing*, 9(4):551–565, 2015.

[398] Ying Wang, Weiru Liu, and David Bell. A Structure-Based Similarity Spreading Approach for Ontology Matching. In *Scalable Uncertainty Management*, pages 361–374. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[399] Yingxu Wang and Y Wang. *Discoveries and Breakthroughs in Cognitive Informatics and Natural Intelligence*. Information Science Reference, 2009.

[400] Lori Watrous-deVersterre, Chong Wang, and Min Song. Concept chaining utilizing meronyms in text characterization. In *Proceedings of the 12th ACM/IEEE-CS joint conference on Digital Libraries*, pages 241–248, New York, New York, USA, 2012. ACM.

[401] Warren Weaver. Translation. *Machine translation of languages*, 14:15–23, 1955.

[402] Sabine Weber. Die Rolle von Synonymen bei der Dekomposition. Bachelors thesis, 2016. Technische Universtität Berlin.

[403] Gerhard Weiss. *Multiagent Systems*. Intelligent Robotics and Autonomous Agents series. MIT Press, 2 edition, 2013.

[404] Heinz Werner and Edith Kaplan. Development of Word Meaning Through Verbal Context: An Experimental Study. *The Journal of Psychology*, 29(2):251–257, 1950.

[405] Tina Wieczorek. *On Foundational Frames for Formal Modelling: Sets, Epsilon-sets and a Model of Conception*. PhD thesis, Technische Universität Berlin, 2009. Technische Universtität Berlin.

[406] Anna Wierzbicka. *Semantic primitives*. Athenäum-Verlag, 1972.

[407] Anna Wierzbicka. *Semantics: Primes and Universals*. Oxford University Press, USA, 1996.

[408] Anna Wierzbicka. *English : Meaning and Culture*. Oxford University Press, 2006.

[409] Anna Wierzbicka. NSM Semantics versus Conceptual Semantics: Goals and standards (A response to Jackendoff). *Intercultural Pragmatics*, 4(4):521–529, 2007.

[410] Anna Wierzbicka. The Theory of the Mental Lexicon. *The Slavic Languages: An international handbook of their history, their structure and their investigation*, pages 848–863, 2009.

[411] Anna Wierzbicka. Common language of all people: The innate language of thought. *Problems of Information Transmission*, 47(4):378–397, 2011.

[412] Wilbur Wilbur and Karl Sirotkin. The automatic identification of stop words. *Journal of information science*, 18(1):45–55, 1991.

[413] Morton Winston, Roger Chaffin, and Douglas Herrmann. A taxonomy of part-whole relations. *Cognitive science*, 11(4):417–444, 1987.

[414] Ludwig Wittgenstein. *Philosophische Untersuchungen* . Akademie Verlag GmbH Berlin, Frankfurt, 1998.

[415] William Woods. Understanding Subsumption and Taxonomy. *The Morgan Kaufmann Series in Representation and Reasoning*, pages 45–94, 1991.

[416] Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition, 2009.

[417] Michael Wooldridge and Nicholas Jennings. Intelligent Agents: Theory and Practice. 10(2):115–152, 1995.

[418] Zhibiao Wu and Martha Palmer. Verbs Semantics and Lexical Selection. In *Proceedings of the 32nd Annual Meeting on Association for Computational Linguistics*, pages 133–138, Stroudsburg, PA, USA, 1994. Association for Computational Linguistics.

[419] Roman Yampolskiy. AI-Complete, AI-Hard, or AI-Easy - Classification of Problems in AI. In *Twenty-third Midwest Artificial Intelligence and Cognitive Science Conference*, pages 94–101, 2012.

[420] Dongqiang Yang and David Powers. Measuring semantic similarity in the taxonomy of WordNet. In *Proceedings of the Twenty-eighth Australasian conference on Computer Science*, pages 315–322. Australian Computer Society, Inc., 2005.

[421] Yeong-Ho Yu and Robert Simmons. Truly Parallel Understanding of Text. In *Association for the Advancement of Artificial Intelligence*, pages 996–1001, 1990.

[422] Edward Zalta. The Stanford Encyclopedia of Philosophy. Winter 2015 Edition. The Metaphysics Research Lab Center for the Study of Language and Information Stanford University, Stanford, 2015.

[423] Torsten Zesch and Iryna Gurevych. Analysis of the Wikipedia category graph for NLP applications. In *TextGraphs-2: Graph-Based Algorithms for Natural Language Processing*, pages 1–8, 2007.

[424] Torsten Zesch and Iryna Gurevych. Wisdom of crowds versus wisdom of linguists - measuring the semantic relatedness of words. *Natural Language Engineering*, 16(1):25–59, 2010.

[425] Torsten Zesch, Christof Müller, and Iryna Gurevych. Using Wiktionary for Computing Semantic Relatedness. In *Association for the Advancement of Artificial Intelligence (AAAI)*, volume 8, pages 861–866, 2008.

[426] Lei Zhang, Chong-Jun Wang, and Jun-Yuan Xie. Cost optimal planning with multi-valued landmarks. *AI Communications*, 28(3):579–590, 2015.

[427] Ziqi Zhang, Anna Gentile, and Fabio Ciravegna. Recent advances in methods of lexical semantic relatedness – a survey. *Natural Language Engineering*, 19(4):411–479, 2013.

[428] Zhi Zhong and Hwee Ng. It makes sense: a wide-coverage word sense disambiguation system for free text. In *Proceedings of the Association for Computational Linguistics, System Demonstrations*, pages 78–83. Association for Computational Linguistics, 2010.

[429] Cai-Nicolas Ziegler, Kai Simon, and Georg Lausen. Automatic computation of semantic proximity using taxonomic knowledge. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 465–474. ACM, 2006.

[430] Guobing Zou, Yixin Chen, Yang Xiang, Ruoyun Huang, and You Xu. AI Planning and Combinatorial Optimization for Web Service Composition in Cloud Computing. In *Annual International Conference on Cloud Computing and Virtualization (CCV 2010)*, pages 28–35. Global Science and Technology Forum, 2010.

[431] Guobing Zou, Yixin Chen, You Xu, Ruoyun Huang, and Yang Xiang. Towards Automated Choreographing of Web Services Using Planning. In *Association for the Advancement of Artificial Intelligence*, pages 178–184, 2012.

[432] Guobing Zou, Yanglan Gan, Yixin Chen, and Bofeng Zhang. Dynamic composition of Web services using efficient planners in large-scale service repository. *Knowledge-Based Systems*, 62:98–112, 2014.

[433] Guobing Zou, Qiang Lu, Yixin Chen, Ruoyun Huang, You Xu, and Yang Xiang. QoS-aware dynamic composition of Web services using numerical temporal planning. *IEEE Transactions on Service Computing*, 5, 2012.

[434] Geoffry Zweig and Christopher Burges. The Microsoft Research sentence completion challenge. Technical report, Technical Report MSR-TR-2011-129, Microsoft, 2011.

Index