Zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften (Dr.-Ing.)
von der KIT-Fakultät für Wirtschaftswissenschaften
des Karlsruher Instituts für Technologie (KIT)
genehmigte Dissertation von
Patrick Raoul Philipp, M.Sc.

Decision-Making with Multi-Step Expert Advice on the Web

Patrick Raoul Philipp

Tag der mündlichen Prüfung: 13.04.2018
Referent: PD Dr. Achim Rettinger
Korreferent: Prof. Dr. Kristian Kersting

Karlsruhe, 2018

Abstract

This thesis deals with solving multi-step tasks by using advice from experts, which are algorithms to solve individual steps of such tasks. We contribute with methods for maximizing the number of correct task solutions by selecting and combining experts for individual task instances and methods for automating the process of solving tasks on the Web, where experts are available as Web services.

Multi-step tasks frequently occur in Natural Language Processing (NLP) or Computer Vision, and as research progresses an increasing amount of exchangeable experts for the same steps are available on the Web. Service provider platforms such as Algorithmia monetize expert access by making expert services available via their platform and having customers pay for single executions. Such experts can be used to solve diverse tasks, which often consist of multiple steps and thus require pipelines of experts to generate hypotheses.

We perceive two distinct problems for solving multi-step tasks with expert services: (1) Given that the task is sufficiently complex, no single pipeline generates correct solutions for all possible task instances. One thus must learn how to construct individual expert pipelines for individual task instances in order to maximize the number of correct solutions, while also taking into account the costs adhered to executing an expert. (2) To automatically solve multi-step tasks with expert services, we need to discover, execute and compose expert pipelines. With mostly textual descriptions of complex functionalities and input parameters, Web automation entails to integrate available expert services and data, interpreting user-specified task goals or efficiently finding correct service configurations.

In this thesis, we present solutions to both problems: (1) We enable to learn well-performing expert pipelines assuming available reference data sets (comprising a number of task instances and solutions), where we distinguish between centralized and decentralized decision-making. We formalize the problem as specialization of a Markov Decision Process (MDP), which we refer to as Expert Process (EP) and integrate techniques from Statistical Relational Learning (SRL) or Multiagent coordination. (2) We develop a framework for automatically discovering, executing and composing expert pipelines by exploiting methods developed for the Semantic Web. We lift the representations of experts with structured vocabularies modeled with the Resource Description Framework (RDF) and extend EPs to Semantic Expert Processes (SEPs) to enable the data-driven execution of experts in Web-based architectures.

We evaluate our methods in different domains, namely Medical Assistance with tasks in Image Processing and Surgical Phase Recognition, and NLP for textual data on the Web, where we deal with the task of Named Entity Recognition and Disambiguation (NERD).

Publications

Text as well as figures in this thesis have partly been published in the following papers:

Philipp Gemmeke, Maria Maleshkova, Patrick Philipp, Michael Götz, Christian Weber, Benedikt Kämpgen, Sascha Zelzer, Klaus Maier-Hein, and Achim Rettinger. Using linked data and web apis for automating the pre-processing of medical images. In *Proceedings of the 5th International Workshop on Consuming Linked Data (COLD'14) co-located with the 13th International Semantic Web Conference (ISWC'14), Riva del Garda, Italy*, pages 25–36, 2014. [51]

Patrick Philipp, Maria Maleshkova, Achim Rettinger, and Darko Katic. A semantic framework for sequential decision making. In *Engineering the Web in the Big Data Era - 15th International Conference, ICWE'15, Rotterdam, The Netherlands*, pages 392–409, 2015. doi:10.1007/978-3-319-19890-3_25. [101]

Patrick Philipp, Maria Maleshkova, Darko Katic, Christian Weber, Michael Götz, Achim Rettinger, Stefanie Speidel, Benedikt Kämpgen, Marco Nolden, Anna-Laura Wekerle, Rüdiger Dillmann, Hannes Kenngott, Beat P. Müller-Stich, and Rudi Studer. Toward cognitive pipelines of medical assistance algorithms. *Int. J. Computer Assisted Radiology and Surgery*, 11(9):1743–1753, 2016. doi: 10.1007/s11548-015-1322-y. [102]

Nicolai Schoch, Patrick Philipp, Tobias Weller, Sandy Engelhardt, Mykola Volovyk, Andreas Fetzer, Marco Nolden, Raffaele De Simone, Ivo Wolf, Maria Maleshkova, Achim Rettinger, Rudi Studer, and Vincent Heuveline. Cognitive tools pipeline for assistance of mitral valve surgery. In *Proceedings of SPIE, Medical Imaging'16: Image-Guided Procedures, Robotic Interventions and Modeling*, pages 9786–9794, 2016. [114]

Patrick Philipp and Achim Rettinger. Reinforcement learning for multi-step expert advice. In *Proceedings of the 16th Conference on Autonomous Agents and Multiagent Systems, AAMAS'17, São Paulo, Brazil*, pages 962–971, 2017. [100]

Patrick Philipp, Achim Rettinger, and Maria Maleshkova. On automating decentralized multi-step service combination. In *IEEE International Conference on Web Services, ICWS'17, Honolulu, HI, USA*, pages 736–743, 2017. doi: 10.1109/ICWS.2017.89. [104]

Patrick Philipp, Maria Maleshkova, Achim Rettinger, and Darko Katic. A semantic framework for sequential decision making. *J. Web Eng.*, 16(5&6):471–504, 2017. [103]

Contents

Part I

Foundations

This part consists of the foundations of the thesis. We first introduce the topic of multi-step tasks with expert advice (Chapter 1) by motivating the latter and defining our research questions, hypotheses and contributions. After presenting the application scenarios used in this thesis in Chapter 2, we explain and define the preliminaries needed for the subsequent content (Chapter 3). We finally discuss related works with respect to the contributions of this thesis (Chapter 4).

Chapter 1

Introduction & Overview

In **this chapter**, we introduce the problem of solving multi-step tasks with expert advice by first motivating the setting in Section 1.1. We then highlight the involved challenges in Section 1.2 with respect to learning and automation. Section 1.3 introduces our research questions and hypotheses, after which we summarize the respective contributions (Section 1.4). Based on the latter, we define the scope of the thesis (Section 1.5) by specifically pointing out which aspects we focus on and which we neglect. We finally give an outline of all parts and chapters in Section 1.6.

1.1 Introduction

An increasing amount of algorithms is made available on the Web - exemplary fields are Computer Vision or Natural Language Processing (NLP). Here, researchers publish their prototypes as code to adhere to the goal of *reproducible research* or wrap them as Web services to enable direct access. To this end, service provider platforms such as *Algorithmia* [1]), which has several advantages for end users:

- End users can reuse *State-of-the-Art* algorithms instead of being forced to develop solutions from scratch. This enables end users to focus on unsolved problems rather than spending an abundance of time in reengineering the State of the Art.
- A side effect of service platforms is an increasing modularity of algorithms, which enables compositions and combinations for heterogeneous applications. Services with higher degrees of modularity as well as higher numbers of exposed hyperparameters can be better adapted and integrated into applications. As a consequence, researchers and developers are incentivized to foster such properties.
- New algorithms are being developed for existing *tasks*, if there is sufficient room for improvement. This results in sets of exchangeable algorithms, where algorithms com-

[1] `https://algorithmia.com/` enable to monetize such algorithms by making them easily accessible via diverse interfaces and having customers pay for single executions. This trend is picked up by influential IT companies, such as Google (by funding Algorithmia [2] (*accessed on 05/01/2018*)

pletely or partially *dominate* others for instances of the task. This drives the development of better approaches, as dominant algorithms are used more frequently and yield higher revenues.

- Hosting algorithms as Web services is a reliable access mechanism for end users, leading the latter to directly integrate respective services into their applications. By monetizing algorithm access, fast and dependable executions of the respective services have to be ensured. As this is especially challenging for algorithms with high *computational complexity* (i.e. the execution requires large amounts of computational resources), end users profit from outsourcing their executions.

- In addition to reliability, service platforms often provide a wide-range of well-known *interfaces* to available algorithms, e.g. directly via *HTTP methods* or via wrappers for common programming languages such as *Java*, *Python* and many more. Providing heterogeneous interfaces is a central requirement, as end users are inhibited to use such platforms if efforts for algorithm access and integration are too high.

Real-world applications often approach to solve *multi-step tasks*, i.e. tasks which require to solve a number of steps and where end users need to *compose* (or *pipeline*) different algorithms to generate hypotheses for task instances.

Example 1.1 (A Multi-Step NLP Task)**.** *Take, for example, the task of structuring text on the Web as available on social media or in news articles. Given piece of text (i.e. a task instance), one needs algorithms to find named entities (step Named Entity Recognition (NER)), algorithms to find relations among the latter (step Relation Extraction) and algorithms to link named entities and relations to knowledge bases (KBs) such as Wikipedia [3], DBpedia [85], Yago [125] or Wikidata [138] (step Named Entity Disambiguation (NED) and step Relation Disambiguation). The sequence of algorithms generates structured pieces of texts (i.e. task solutions).*

Figure 1.1 illustrates such tasks, comprising a number of steps (variable H) which need to be sequentially solved for a task instance to generate a task solution. A number of algorithms (variable N) available in an arbitrary Web repository have to be used to solve the respective steps. Each algorithm needs a set of inputs in a specified format to generate a solution for a task instance.

When trying to find well-performing algorithms for a multi-step task, multiple options (i.e. exchangeable algorithms) are often available for a single step. As a result, numerous algorithm pipelines might be possible to configure, where pipelines solve the same steps but potentially produce different results for task instances. For a sufficiently challenging task, there is a high probability that the set of all possible pipelines produce conflicting hypotheses for at least a subset of task instances. In addition to disagreements among pipelines, it is probable that

[3] `https://en.wikipedia.org/wiki/Main_Page` (accessed on 05/01/2018)

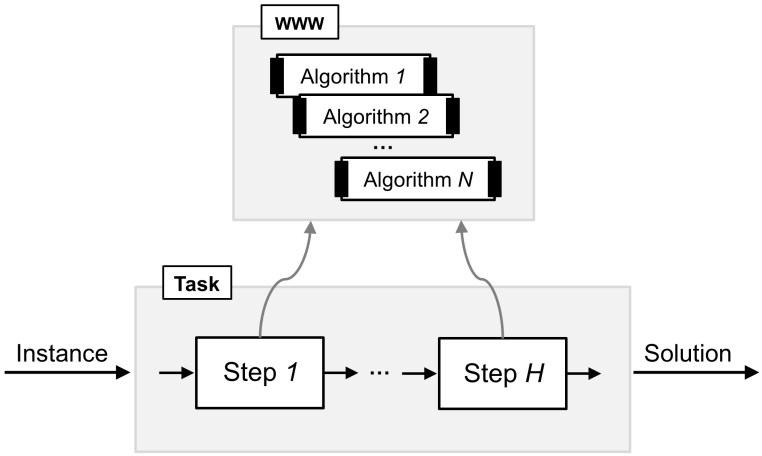

Figure 1.1: General schema for using algorithms to solve multi-step tasks.

there is not a single *correct* pipeline for all task instances, i.e. a pipeline works well for some task instances but produces wrong solutions for others. Assuming that pipelines produce good hypotheses for different task instances, it is sensible to choose different pipelines for different task instances *to maximize the number of correct solutions for the task*.

Choosing among exchangeable algorithms can be generally approached by *supervised machine learning* (short *supervised learning*). One exploits *training sets* to generalize the correctness of an algorithm to new task instances. Training sets contain <task instance, label> tuples (also referred to as *samples*), where the label is the correct solution. The general procedure entails to execute algorithms on task instances in the training set and to compare their outputs to the respective label. While, for single-step tasks, there are advances for learning to assess and combine algorithms (e.g. [3, 27, 149], there is a gap in research for multi-step tasks, especially with additional constraints (e.g. only being able to execute a subset of available algorithms for a step). We refer to the latter as *learning problem*.

Figure 1.2 illustrates the learning problem, where steps and available algorithms are assumed to be known. The correctness of an algorithm for a task instance is, however, unknown and has to be assessed. Arrows from steps to algorithms are labeled with *weights* (i.e. confidence estimate) for the proposed solutions, which are have to be estimated.

Given that algorithms are available as Web services, *automatically* solving diverse multi-step tasks becomes possible. This, however, requires a Web architecture able to find the set of possible steps to solve a multi-step task, discover the set of algorithms for these steps and correctly execute the chosen algorithms. The needed steps to solve a multi-step task usually have to be found based on *textual descriptions* and *taxonomies*. Here, different compositions of steps might solve the task, as services for algorithms wrap overlapping functionalities. The mentioned step NER could, for example, be divided into two smaller steps, where text is tokenized in a dedicated step and then further processed. To this end, textual descriptions for

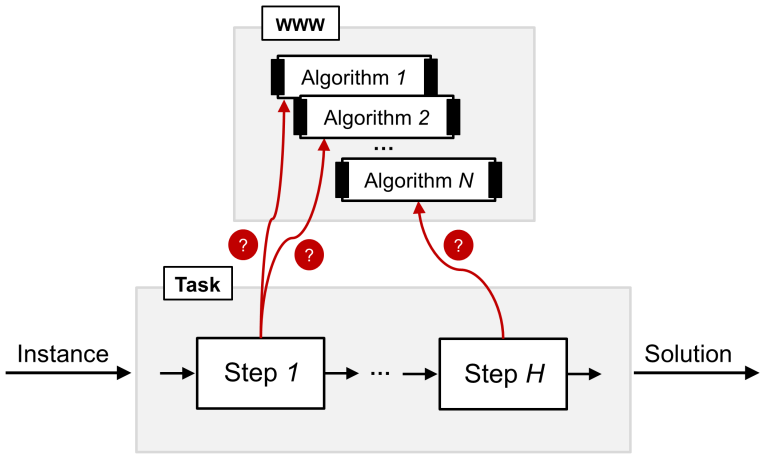

Figure 1.2: Learning problem of multi-step tasks using algorithms to solve steps.

algorithm services significantly complicate their execution, as required inputs and available parameters have to be correctly interpreted. While there is a large body of works with respect to service discovery, -selection, -composition and orchestration, the literature lacks the needed decision-theoretic focus to work towards Web automation for multi-step tasks using algorithm services. More specifically, it is essential to automate solving the learning problem in Web environments, while keeping the architecture flexible for novel algorithms and tasks. We refer to the latter as *automation problem*.

Figure 1.3 illustrates the automation problem, where the needed steps of the task are unknown, and algorithm inputs and outputs need to be interpreted.

Throughout this thesis, we refer to such algorithms as ***experts***, as the latter generate *specialized hypotheses* for a class of task instances. We thereby align our terminology with the seminal work on *prediction with expert advice* [27], where a decision-maker has to learn how to use the advice (i.e. the hypotheses for task instances) of experts for a task with a single step.

Figure 1.4 summarizes our general approach to the learning and the automation problem in terms of a *feedback loop* of an agent, having to choose experts to solve a multi-step task. The figure illustrates the loop for a single step (variable h, where $h \in 1\ldots,H$). As we argue and show throughout this thesis, *Markov Decision Processes* (MDPs) [109] are an adequate framework for both problems. MDPs enable to model *sequential decision-making* problems, where a state (variable s^h) requires to take a single action (variable a^h), which yields a novel state (variable s^{h+1}) and a *numerical* reward (variable r^h). *Reinforcement Learning* (RL) is a paradigm for *learning to act* in MDPs, which we use to solve the learning problem. For the automation problem, we additionally exploit *Semantic Web Technologies* to flexibly describe, discover and execute *expert services*. By *Planning* in a MDP, which returns the best action for a state *after* the learning problem is solved, we can automatically compose experts to solve

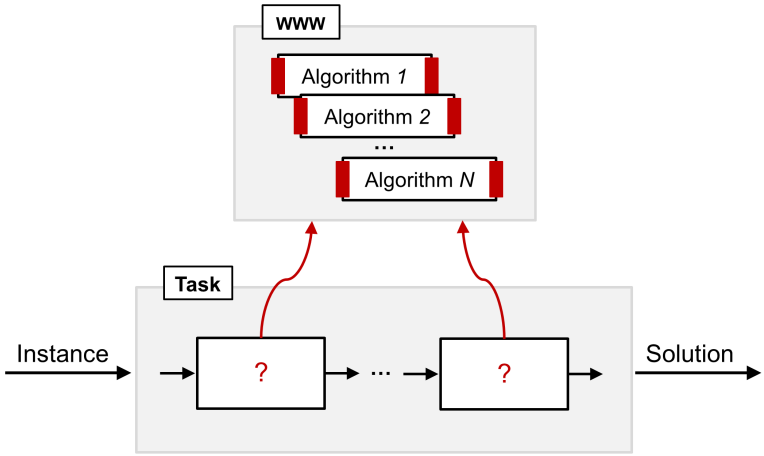

Figure 1.3: Automation problem for solving multi-step tasks using algorithms.

multi-step tasks.

We now describe the respective challenges in detail.

1.2 Challenges

We first discuss challenges for assessing the correctness of expert hypotheses for individual task instances (i.e. the learning problem) and then deal with how to find and correctly execute expert services for a task (i.e. the automation problem).

1.2.1 Learning

To choose the *best* expert pipeline for a task instance, the following observations and challenges for a learning system are central:

1. Querying multiple exchangeable experts is advantageous, as taking the average of their hypotheses (referred to *majority vote*) often results in correct solutions for a large number of task instances (with respect to the available experts). More specifically, the prominent fields of prediction with expert advice [27] and ensemble learning [149] both deal with learning to assess single or sets of experts to optimize their majority vote by assigning different weights to different experts. However, as service platforms use cost models for expert access, it is sensible (e.g. from a monetary perspective) not to execute all available experts for a step. By limiting the executions of experts for individual task instances, one loses information about their outputs and needs to deal with resulting missing information.

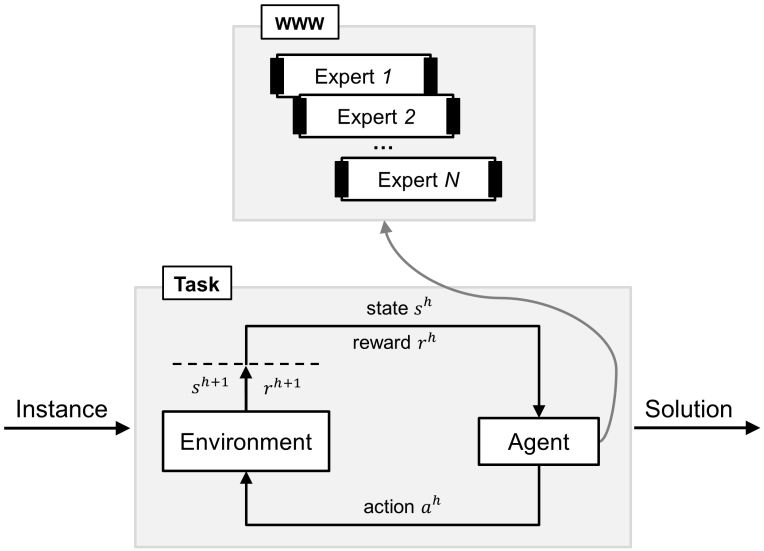

Figure 1.4: General approach for solving multi-step tasks with experts.

2. A possible approach for choosing experts and combining their outputs is to ignore that the task has multiple steps. This, however, does not favor the solution for the task, as future experts might perform better for *inexact* (i.e. *semi-correct*) inputs. For the mentioned task of structuring Web text, for example, an expert for the NED step might correctly link `Michael Jordan plays` to resource `http://dbpedia.org/page/Michael_Jordan` but wrongly link `Michael Jordan` to `http://dbpedia.org/page/Michael_I._Jordan`, as it confuses the basketball player with the computer science researcher – where `Michael Jordan plays` is not a named entity but `Michael Jordan` is. Expert pipelines constructed based on *local optimization* (i.e. only for individual steps) thus often fail to find *global optima* for multi-step tasks.

3. Supervised learning often assumes training sets to be available in a *batch*, where all task instances and labels are available *prior to* learning. For multi-step tasks with experts, we have to generate novel, *expert-specific* training sets by executing experts and comparing their hypotheses with the labeled task instances of task training sets. To use batch learning for expert-specific training sets, one has to either (i) randomly execute experts and randomly choose step solutions, where wrong solutions propagate to later steps or (ii) try to execute experts of all steps with correct inputs and artificially create labels for all steps. Both options introduce a *learning bias*, as generated expert-specific training sets are not representative of the joint *performance* of all experts involved in the task (i.e. how often their eventually generated solutions are correct).

4. Available training sets for tasks often comprise relatively few task instances. The resulting challenge for using supervised learning is that the learning process needs to *converge* sufficiently fast (i.e. for multi-step tasks, the resulting expert weights have to work well for choosing and combining experts). For learning to generalize across task instances, supervised learning entails to define a *feature representation*, e.g. a vector which summarizes characteristics of a task instance. When training sets are small, feature representations for task instances must not be high-dimensional.

5. When combining proposed solutions of multiple exchangeable experts, one has to additionally deal with *expert correlation*. Two experts might propose wrong solutions for a large number of shared task instances, which causes the *weighted majority vote* (i.e. the majority vote with learned expert weights as influence factor for solutions) to overestimate their solutions for new task instances. If two experts only correlate in terms of proposing correct solutions for shared task instances, the weighted majority vote is improved.

1.2.2 Automation

In addition to learning to assess expert hypotheses, Web automation involves to find, execute and compose expert services for a multi-step task. We now discuss the involved challenges in detail.

6. Expert services usually provide *descriptions*, such that end users or application systems can discover and execute them. A description often consists of textual information about the available functionality as well as explanations about the needed inputs and parameters. Executing expert services thus presupposes that end users or application systems sufficiently understand the description, which is usually difficult, as required formats need to be interpreted and converted. The latter often requires to consult *domain experts*, especially when task domains are specialized (e.g. tasks for medical assistance).

7. After having discovered experts and ensured their correct execution, there often is the need to develop additional *glue code* (i.e. conversion mechanisms from data to service inputs) in order to transform generated expert outputs of a step to correct formats for experts of a subsequent step. As a consequence, one often cannot automatically integrate newly discovered experts and has to manually write glue code.

8. Even if inputs and outputs of expert services are aligned and sufficiently integrated, automatically executing expert services remains challenging. The latter is due to the fact that steps often need various inputs, where mapping available data of the task instance to input descriptions of expert services has to be *task-oriented*. When executing a NED expert service, for example, one needs to select an undefined number of annotated named entities, while ensuring that selected named entities do not overlap. In addition,

one has to take into account that NED experts might make mistakes for correct named entities.

9. There might be an *expert class* to integrate the expert into a *taxonomy*. While the latter could be helpful to map experts to steps of the current task, exclusively using it often results in missing possible expert pipelines. This is due to the fact that steps can often be further partitioned, which is hard to automatically (or even manually) discover.

10. As there often exist a number of exchangeable experts for a single step, assessing and weighting them is important to find correct solutions for a multi-step task (as stated for the learning problem). One thus has to integrate adequate *learning* approaches into a Web architecture. Learning approaches might, however, be *tailored* to a specific step or outperformed by newer learning algorithms. Fixed *procedural* workflows thus do not suffice to sustainably solve multi-step tasks, as they lack flexibility.

Based on the challenges for learning and automation with respect to multi-step tasks, we now stipulate our research questions and hypotheses for the contributions of this thesis.

1.3 Research Questions & Hypotheses

We first describe our hypotheses and respective research questions for learning about the performance of experts in a supervised setting. All hypotheses and research questions assume a multi-step task with sets of experts for each step, where for at least one step the respective expert set comprises more than one expert. We begin by dealing with learning challenges 1-3 in Hypothesis 1.

Hypothesis 1. *By learning expert weights for individual task instances which express an expert's ability to support generating the correct task solution, and by choosing experts with respect to the exploration-exploitation problem, we can maximize the number of correct solutions for multi-step tasks.*

The hypothesis tackles Research Question 1:

Research Question 1. *How can we formalize and model the problem of solving multi-step tasks assuming the prevalence of experts for predefined steps and expert execution budgets, where we have access to feedback in terms of training sets for the respective task?*

Hypothesis 1 thus proposes the basic properties of a general framework for weighting experts for multi-step tasks. To this end, it is central to study when to *exploit* experts with high weight estimates (i.e. when to choose a promising expert) and when to *explore* experts with low or unknown expert weights (i.e. when to choose an expert with low or unknown expectations). The framework is the baseline for all residual learning- and automation challenges.
By additionally tackling challenge 4, we make Hypothesis 2:

Hypothesis 2. *RL using relational features for individual task instances enables to deal with expert budgets in order to maximize the number of correct solutions for multi-step tasks with expert advice.*

Hypothesis 2 corresponds to Research Question 2:

Research Question 2. *How can we learn which experts to choose in which steps given an expert execution budget to correctly solve multi-step tasks?*

Hypothesis 2 describes how we approach to learn expert weights for multi-step tasks. Expert weights need to be computed for each task instance, while expressing the expert's global potential to support solving the task. As several experts are not executed for a respective task instance, one does not receive their labels and has to take into account that their hypotheses might have been correct. RL is deemed to be especially suited for this problem, as it specifically studies how to estimate future impacts of actions and how to balance their exploration and exploitation. Relational features enable to compactly integrate diverse feature representations of task instances and experts.
Finally, tackling challenge 5, we make Hypothesis 3:

Hypothesis 3. *Framing expert assessment for multi-step tasks as Multiagent coordination problem and solving the latter can improve the number of correct solutions for multi-step tasks, while keeping the resulting system flexible.*

Hypothesis 3 is connected to Research Question 3:

Research Question 3. *How can we take into account the correlation of experts in order to maximize the number of correct solutions for multi-step tasks?*

The goal of Hypothesis 3 is to extend the learning framework to a Multiagent System (MAS), where coordination approaches can be studied. The resulting coordination process helps to decorrelate expert hypotheses resulting from learned expert weights, while also increasing the flexibility of the system.
We now deal with research questions and hypotheses for the problem of automatically solving multi-step tasks with experts on the Web. Hypothesis 4 approaches challenges 6 and 7.

Hypothesis 4. *Given a multi-step task with available experts, structuring experts and task instances with lightweight annotations modeled with the Resource Description Language (RDF) as well as exploiting HTTP methods to execute the services is sufficient to find and execute expert services.*

The hypothesis is associated with Research Question 4:

Research Question 4. *How can we account for heterogeneous expert functionalities, -implementations and -interfaces for solving multi-step tasks on the Web?*

Hypothesis 4 deals with the problem of structuring textual service descriptions, as the latter are both cumbersome and difficult to understand for end users and machines. Semantically enriched expert services are a first step towards automating their execution.
Hypothesis 5 tackles challenges 8-10.

Hypothesis 5. *Expert services lifted with concepts of the Semantic Web can be executed on a data-driven basis, where decision-theoretic models enable planning their execution as well as integrating expert weight learning approaches to solve multi-step tasks.*

The hypothesis is associated with Research Question 5:

Research Question 5. *How can the needed set of expert services for a multi-step task be discovered, executed, composed and weighted to solve multi-step tasks with expert advice?*

Hypothesis 5 deals with generating task-oriented solutions using expert weight learning techniques for multi-step tasks. By using adequate decision-theoretic models to integrate learning and planning techniques and by using a data-driven approach for expert execution, we can achieve automatic compositions of experts.

1.4 Contributions

We now summarize the contributions for the respective hypotheses.

Contribution 1. *A decision-theoretic model merging MDPs and prediction with expert advice – referred to as Expert Processes (EPs) – to enable the reuse and further development of RL algorithms.*

The contribution to Hypothesis 1 comprises EPs, which are a general purpose framework for multi-step decision making with expert advice. As EPs are a specialization of MDPs, one can easily reuse RL algorithms and only has to find adequate feature spaces for the expert weight learning problem. EPs are introduced and explained in Chapter 5.

Contribution 2. *RL algorithms using relational pairwise expert feature spaces, which enable to learn expert weights to maximize the number of correct task solutions.*

We contribute to Hypothesis 2 by presenting two RL solutions for the expert weight learning problem. The feature spaces comprise diverse relational pairwise expert measures (referred to as *meta dependencies*), which are calculated for experts of same- as well as subsequent steps. Meta dependencies for experts of subsequent steps account for an expert's ability to support solving the task. We present meta dependencies as well as RL algorithms in Chapter 5 after having defined EPs.

Contribution 3. *Multiagent coordination protocols for Multiagent Expert Processes (MEPs) (i.e. EPs framed as MAS) with appropriate agent coordination techniques and feature spaces, which enable to decorrelate expert weights while keeping the resulting system flexible.*

Our contribution for Hypothesis 3 comprises the formalization of MEPs with novel Multiagent coordination protocols, which enable to study *decentralized* multi-step decision-making for multi-step tasks. We present methods for adapting expert weights by coordinating them with other cooperative agents in order to reduce their correlation. Chapter 6 comprises the formalization of MEPs as well as three coordination approaches for two coordination protocols. To this end, we evaluate contributions 1, 2 and 3 in Chapter 7 for the NLP task of *Named Entity Recognition and -Disambiguation* (NERD) with steps NER and NED.

Contribution 4. *A lifting mechanism from experts to semantic experts, using Linked APIs (LAPIs) and lightweight service descriptions, which is sufficiently time-efficient and flexible to solve selected NLP- as well as medical assistance tasks.*

We contribute to Hypothesis 4 by *lifting* expert services (i.e. adding structure to their descriptions and execution mechanisms) to semantic experts, which are lightweight Web services annotated with structured vocabularies. We formally define semantic experts and describe their needed components in Chapter 8. We apply the concept of semantic experts to NERD- and medical assistance experts in Chapter 9 and also evaluate their time-efficiency.

Contribution 5. *The integration of semantics into the EP framework – yielding Semantic Expert Processes (SEPs) – and methods reusing and extending MDP planners and EP learners to discover, execute, compose and weight adequate semantic experts for individual task instances. The methods are applicable to NLP as well as medical assistance tasks.*

Our contribution for Hypothesis 5 comprises SEPs, which are semantic liftings of EPs. SEPs are the baseline to develop *semantic meta components*, which are learning and planning algorithms to weight- and find adequate experts for a given *semantic multi-step task*. We introduce SEPs as well as the concept of semantic meta components in Chapter 8 and present specific applications of the latter to the NERD task as well as two medical assistance tasks in Chapter 9.

1.5 Scope of the Thesis

We define the scope of the thesis by revisiting the main problems we approach and by highlighting as well as confining our goals.

The first main problem we deal with is learning about expert performances for multi-step tasks under possible budget constraints, i.e. one can only execute a subset of experts for a task instance. We approach the problem by supervised learning and thus assume the availability of training sets for tasks. To this end, we exclude *unsupervised machine learning* (short *unsupervised learning*) methods from our scope, as the latter make strong assumptions on the general performance of experts (e.g. their minimal- or average correctness on a series of task instances). As expert performances often significantly vary across different data distributions and experts often correlate in generating wrong solutions, such assumptions do not hold.

In addition to unsupervised learning, we also exclude the theoretical or empirical study of *convergence* (also referred to as *sample complexity*) of our learning approaches, which is prominent for evaluating RL algorithms. RL algorithms are used in this thesis, as they can be applied to the problem of weighting and choosing experts under budget constraints. Our overall goal for the learning problem is to maximize the number of correct solutions for multi-step tasks and we thus assume that, if the respective learning approaches have converged sufficiently fast given the available training sets, the number of correct solutions is maximized.

Our last limitation for the scope of the learning problem deals with the sub-problem of reducing expert correlation. We cast the problem as MAS and develop techniques to coordinate available hypotheses. However, central problems of interest in MAS coordination deal with investigating constrained communication cases, where agents cannot retrieve the global state. In this thesis, we do not deal with constrained communication MAS, as such a constraint only makes sense if the respective learning system is distributed (and thus decentralized). The essential difference of our goal is that we are only interested in exploiting the coordination process among experts, which introduces novel perspectives on their weights and resulting hypotheses. To this end, we again do not investigate the convergence behavior of our expert coordination techniques, where our argumentation remains the same as before, i.e. if our methods converged, the number of correct solutions for the respective multi-step task with expert advice is maximized.

The second main problem approached in this thesis is automatically solving multi-step tasks on the Web, where experts are available as Web services. Our goals are to automatically find adequate expert services for steps, derive plans which only comprise expert services needed for the overall multi-step task, enable automatic querying of the resulting expert services and, finally, integrate expert weight learning methods (as developed for the learning problem of this thesis). Our methods rely on advances in Semantic Web technologies for both structuring data as well as services, but we exclude methods around the topic of *reasoning*. While reasoning is a central means to derive novel information, we aim to express (and explore dealing with) uncertainty about the *predictive performance* (i.e. estimated correctness) of an expert by means of supervised learning. More specifically, we argue that manually created structured annotations cannot compete with learned *expert weight approximations* based on supervised learning methods. We thus aim to integrate the latter into the automation approach.

We also do not deal with reasoning with respect to *non-matching* semantic annotations for task instances and expert services, which could be successfully used to *align* them by *ontology matching*. We assume that a single centralized vocabulary is used for all experts and task instances.

Finally, when lifting expert service descriptions to semantic representations and enabling to automatically execute expert services, we exclude problems with respect to security from our scope. The latter are certainly domain-dependent (in medicine, for example, security is highly important) and adequate authorization, encryption or even data anonymization approaches have to be used or developed. Such approaches might be highly complex and have to be used in addition to our developed methods.

1.6 Outline

We now outline the contents of the thesis, consisting of four parts – *Foundations*, *Learning*, *Automation* and *Conclusion*.
Proceeding with Part I - *Foundations*, we introduce the following chapters to set the baseline for the novel methods proposed in the thesis.

Chapter 2 - *Scenarios*
We describe the application scenarios used throughout the thesis for instantiating and evaluation our proposed approaches and highlight the respective domain challenges.

Chapter 3 - *Preliminaries*
The chapter defines and introduces the baseline concepts and technologies needed for the contributions of this thesis.

Chapter 4 - *Related Work*
The chapter deals with the related works for the contributions of this thesis with regards to the learning- and automation problem.

In Part II - *Learning*, we propose two frameworks for assessing experts in multi-step tasks, propose appropriate learning methods for them and evaluate the latter.

Chapter 5 - *Learning with Expert Processes*
The chapter comprises the formalization of EPs, which is a general purpose framework for multi-step decision-making with expert advice. We then propose a general way to derive feature representations for expert weight approximation and introduce two RL approaches for EPs.

Chapter 6 - *Learning with Multiagent Expert Processes*
We extend the EP formalism for MAS – resulting in MEPs - to enable the study of flexibly reducing expert correlation. Based on the MEPs definition, we develop and present methods for a *passive-* and a *active coordination protocol*, where the latter entail solving a novel decision process.

Chapter 7 - *Evaluation of Learning with EPs & MEPs*
In this chapter, we evaluate the proposed expert weight learning methods for EPs and MEPs for the task NERD. Here, we distinguish between the ability to estimate the performance of experts for individual steps and the ability to maximize the number of correct task solutions for multi-step tasks.

Part III - *Automation* follows by taking a Web perspective on EPs to work towards automatically using expert services on available on the Web to solve EPs.

Chapter 8 - *Automating Semantic Expert Processes*
We first extend the EP framework to SEPs, where state, action and expert spaces are lifted to semantic representations. We then propose a general Web architecture to solve SEPs, which we describe in detail.

Chapter 9 - *Applications & Evaluations of Semantic Expert Processes*
The chapter comprises applications of SEPs with instantiations of the Web architecture for NERD and two medical assistance scenarios. We evaluate the respective applications scenarios in terms of time-efficiency and automation capabilities for SEPs.

Finally, Part IV - *Conclusion* provides a retrospective overview of the content of the thesis and an outlook on future works.

Chapter 10 - *Summary*
We summarize the thesis with respect to the stipulated challenges, research questions, hypotheses and contributions.

Chapter 11 - *Outlook*
The final chapter gives an outlook on future research directions which result from our work on EPs, MEPs and SEPs for multi-step tasks with expert advice.

Throughout the thesis, we will mention the published research papers which correspond to the contributions of the respective *chapters*.

Chapter 2

Scenarios

In **this chapter**, we present the scenarios used to apply and evaluate our proposed methods for both the learning- and the automation problem of multi-step tasks with expert advice. We deal with two different scenarios within the field of medical assistance, namely *Surgical Phase Recognition* and *Tumor Progression Mapping* (TPM), and one scenario in NLP, namely NERD. We first describe the respective tasks and then discuss their challenges with regards to multi-step tasks with expert advice.

2.1 Named Entity Recognition & Disambiguation

2.1.1 Description

NERD deals with the problem of resolving ambiguity in unstructured text, as available on the Web. *Named entities* are real-world objects which can be textually represented in a number of ways. A distinct instantiation of a named entity in a piece of text is then referred to as *mention* (e.g. `#MichaelJackson` for named entity `Michael Jackson`). As a mention of an entity in text can be ambiguous, it often remains unclear which real-world entity is actually being referred to. It is thus important to find links between such mentions and unique entities available in KBs, such as Wikipedia, DBpedia, Yago or Wikidata.

More specifically, NERD consists of the steps NER and NED. In NER, one has to decide if textual tokens have to be *annotated* as distinct type of named entity (e.g. a person, a location or an organization) or as not being a named entity (i.e. tagged as *NIL*). However, other than in traditional NER, we only need a binary decision for solving NERD, i.e. NER can be reduced to mapping tokens (or token sequences) to $\{0,1\}$ with 1 referring to the presence of a named entity and 0 to the absence. In this context, NER is often referred to as *mention detection*.

NED, in turn, receives text annotated with named entities as input and has to link the latter to a modeled *resource* in a KB, which eventually disambiguates the mention in a text. NERD can be evaluated and tested on publicly available training sets and experts, which have been integrated by State of the Art NERD benchmark GERBIL [131].

2.1.2 Challenges

- We approach learning to assess the performance of experts with supervised learning and thus need an appropriate feature representation. Feature generation for text is challenging since available training sets are small in size. Deep learning approaches [84], for example, need large numbers of samples to converge and are not directly applicable. To this end, one could reuse learned *vector embeddings* for words (e.g. [92, 98]), which are feature representations learned on large textual corpora in an unsupervised manner. While a vector embedding might generalize well, its availability for a specific word is strongly dependent on used corpus and its dimensionality might be high.

- The NERD task consists of two steps, where the overall performance is strongly dependent on NER. If a named entity is wrongly excluded by NER experts, the result cannot be correctly processed by any NED expert. Also, if tokens are wrongly annotated as named entity, the respective NERD task can only be correctly solved if NED experts do not link the tokens to a resource. The latter requires sufficient *robustness* of NED experts. Lastly, NED experts might better work with inexact inputs, where available named entities are not perfectly annotated.

- For different text distributions, such as tweets or news articles, NED and NER experts often significantly vary in their performance. As a consequence, not a single NER or NED expert usually dominates the rest for each available task instance.

- It is essential to interpret larger amounts of text (e.g. sentences or paragraphs) at once, as the latter contain *contextual information* needed for NER and NED. Numerous decisions for each token or token sequence then influence each other, which makes learning expert weights for NERD more difficult, as different experts often generate correct hypotheses for different tokens in a single piece of text.

2.2 Surgical Phase Recognition

2.2.1 Description

Surgical Phase Recognition (short *Phase Recognition*) is focus of active research for *medical assistance systems* [76]. Surgeons today are faced with a variety of intraoperative information sources. Vital signs, scans generated by Computed Tomography (CT) or Magnetic Resonance Imaging (MRI), or various device states are available at any point during the surgery. The problem is that only a small fraction of the available information is actually relevant in a given situation. Showing all information at once is thus distracting and potentially harmful, as the data will surpass human ability to process it [60]. In order to make full use of *computerized surgery*, we need methods to infer the current phase of the surgery to only present relevant information to the surgeon.

Phase recognition is a *one-step* task which can be approached by only analyzing *situation features* (also referred to as *activities*) [79, 95]. Here, activities are represented as triples consisting of the used *instrument*, the performed *action* and the corresponding anatomical *structure*, e.g. <Scalpel, cut, Gallbladder>. Surgical phases are defined over sequences of activities and are labeled based on standardized medical terminology (e.g. *monitoring phase* where organs are inspected or *resection phase* where parts of organs are removed). Intraoperativey, these activities are recognized using *sensor analysis techniques* [120]. For evaluation and testing, we use manually annotated videos of surgeries, where clinical experts provide the training sets. The annotations were created using the *SWAN-Suite software* [96].

2.2.2 Challenges

- Surgical Phase Recognition can be approached by diverse methods. Available experts, for example, exploit predefined heuristic rules for sequences of activities or build on supervised learning approaches to generalize from the current activity by exploiting additional training sets. The resulting inputs for such experts differ, as rule-based experts do not need training sets. This is challenging for automating the execution of expert services, as their *interface complexity* is different.

- Available training sets for Phase Recognition are small, as only domain experts are able to confidently label recorded surgeries. In addition to labelling, surgeons significantly differ in the way they perform surgeries. They generally use different instruments and perform activities in different order. This makes using labeled surgeries from different surgeons challenging for learning expert weights.

2.3 Tumor Progression Mapping

2.3.1 Description

Radiologists have to monitor the development of *glioblastoma* (i.e. *brain tumors*) during the treatment of a patient, which requires several tedious and complex, usually manually performed steps. This is due to the fact that glioblastoma grow irregularly and monitoring is generally conducted by visually comparing headscans taken by different physicians. As these headscans are not spatially *registered* (i.e. *aligned*), monitoring becomes cumbersome.

TPM is an approach to visualize the timely progression of brain tumors for radiologists. A TPM consists of processed, spatially registered headscans (i.e. CTs or MRIs) of a single patient, which are ordered with respect to the time the scan was taken (the horizontal dimension) and the used *concentration* (the vertical dimension). Figure 2.1 and 2.2 present an exemplary headscan and a TPM. The images were displayed and processed with the *Medical Imaging Interaction Toolkit* (MITK) [142].

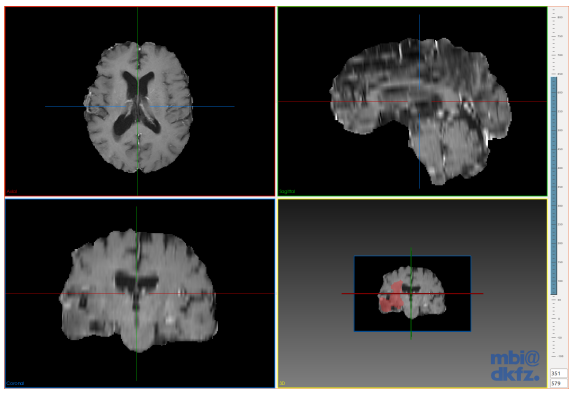

Figure 2.1: An example of a MRI headscan.

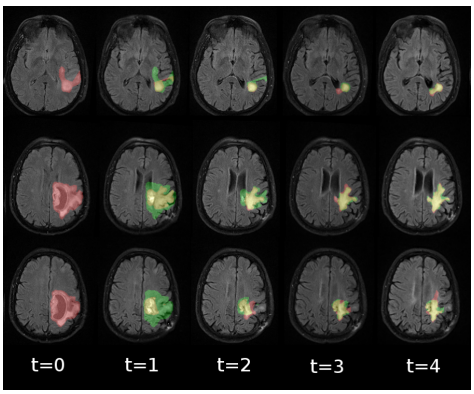

Figure 2.2: An example of a TPM.

The workflow for creating TPMs is illustrated in Figure 2.3. Available headscans are stored in a *Picture Archiving and Communication System* (PACS) and converted into a common format, either *Nearly Raw Raster Data* (NRRD) [1] or *MetaImage* (MHA) [2]. A *brain mask* for the brain region is created (step *Brain Mask Generation*), ensuring that subsequent steps are not negatively influenced by bones or other structures in the headscan. Step *Standard Normalization* adapts the intensities of MRI scans to ensure that similar tissue types have similar values. If brain masks with *additional details* have been manually created by radiologists, a more robust normalization can be achieved (step *Robust Normalization*). All normalized brain masks of a single patient now have to get spatially *registered* such that the same perspective is used for the eventual TPM (step *Registration*). Based on registered brain masks, the TPM can finally be created (step *Map Generation*). An optional step before generating the TPM is to *segment tumors* to further ease interpretation for radiologists (step *Tumor Segmentation*). For evaluating and testing TPM experts, we use MRI scans taken within different time steps of the treatment process of a patient.

2.3.2 Challenges

- Given that a large number of MRI or CT scans are available for a single patient, correctly composing and executing TPM experts becomes difficult. This is due to the fact that numerous possible mappings result from all available headscans to input parameters of TPM experts. The Map Generation step, for example, requires all processed brain masks (which are needed for the final TPM) as input. If not all headscans should be part of the *tpm* (based on a constraint given by a radiologist), one has to test all brain mask combinations of lengths $1, \ldots, |n|$ (with n the number of available headscans). Such *brute force* approaches are not tractable.

[1] `http://teem.sourceforge.net/nrrd/format.html` (*accessed on 05/01/2018*)
[2] `https://itk.org/Wiki/MetaIO/Documentation` (*accessed on 05/01/2018*)

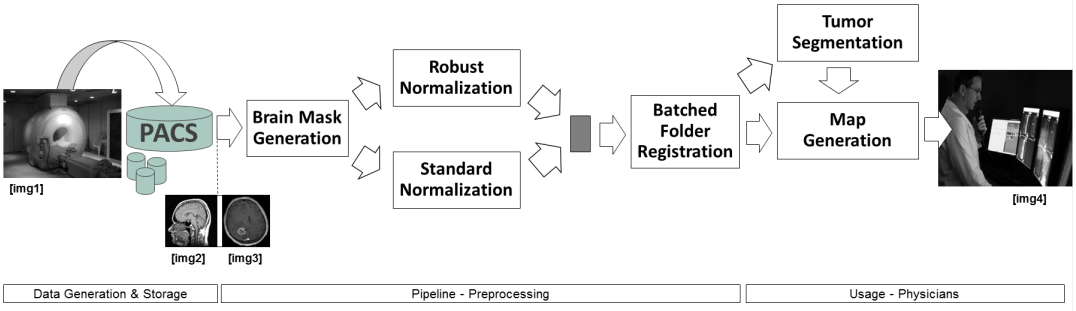

Figure 2.3: Steps for TPM experts.

- The TPM task consists of at least four steps (without optional segmentation step), which requires to efficiently find (i.e. discover) appropriate experts. *Brute force* approaches, again, are not tractable for tasks with large numbers of steps, as one has to repeatedly perform pairwise comparisons of experts to test for correct sequences.

Chapter 3

Preliminaries

This chapter defines basic concepts needed for the contributions of this thesis. We approach to solve multi-step tasks with available experts in terms of two specific goals, i.e. *learning* to choose experts for task instances and combine their hypotheses, and *automating* the process of finding and executing experts to solve multi-step tasks on the Web. To clarify the scope of our proposed methods, we start by defining our constrained notion of multi-step tasks in Section 3.1. While the definitions related to multi-step tasks are novel and not reused from other works, the residual sections focus on pointing out as well as explaining existing concepts and methods, which are required to present the contributions of this thesis. We first deal with the goal of *learning*. The methods we present throughout the thesis aim to generalize the expected correctness of an expert's output based on characteristics of task instances (e.g. the tokens of a sentence). A central means to enable generalization is *supervised learning*, which we selectively introduce in Section 3.2. However, to deal with budget constraints for experts and to account for having multiple sequential steps, we discuss selected aspects of *decision-making* theory in Section 3.3. To prepare the presentation of our proposed methods for *automation*, we turn to *Web technologies* in Section 3.4, especially ones enabling machine-readability within the Semantic Web. For that matter, we deal with central methods for creating and querying structured vocabularies as well as using the latter to enrich Web services.

3.1 Multi-Step Tasks

We first define our notion of *tasks* with respect to the problems and challenges approached in this thesis.

Definition 3.1 (Tasks)**.** *We define a task as* 3*-tuple* $Task = (X, B, Y)$*, i.e. in terms of a mapping of an arbitrary input space* X *and background knowledge space* B *to an arbitrary output (or hypothesis-) space* Y*. The solution to a task is a function* $Solution : X \times B \rightarrow Y$*. Hence, for a given input instance* $x \in X$ *and subset of the background knowledge* $\hat{B} \subset B$*,* $Solution(x, \hat{B})$ *returns the solution as instance* $y \in Y$*. We, thus, assume that solving a task exclusively entails to derive or generalize information from* X *and* B *to* Y*.*

If tasks are complex, it is sensible to divide them into smaller, more manageable *steps* which can be optimized more efficiently and transparently. With this intuition, we introduce

multi-step tasks (and from now perceive tasks and multi-step tasks as being equal). As the *step structure* might get complex, we distinguish between two general types of multi-step tasks, namely *layered-* and *non-layered* multi-step tasks.

A *layered multi-step task* assumes the simplest step structure in that – except for starting and ending steps – a step has exactly one predecessor and one successor. Although this property significantly reduces the possible *structural complexity* of tasks, it is often taken for optimization problems (e.g. [77]).

Definition 3.2 (Layered Multi-Step Tasks)**.** *A* layered *multi-step task is a task where sequential steps can be identified such that the task can be divided and a step has one exact predecessor and/or successor. Given input space X, we assume that $i = 1,\ldots,H$ with $H \in \mathbb{N}^+$ steps have to be performed to solve the task, where a step is considered to be a task itself, i.e. a step is a 3-tuple $Step^i = (X^i \times \hat{B}^i \to Y^i)$ with $X^1 = X$, $Y^H = Y$, and $X^i = Y^{i-1}$ for $i > 1$. The solution of a layered multi-step task for an input instance x is thus defined as $Solution(x,B) = Solution^H(Solution^{H-1}(\ldots(Solution^1(x,\hat{B}^1)\ldots),\hat{B}^{H-1}),\hat{B}^H)$, where $Solution^i : X^i \times \hat{B}^i \to Y^i$ solves the i-th step.*

Non-layered multi-step tasks can be defined in a similar fashion, where now multiple predecessors as well as successors are possible.

Definition 3.3 (Non-Layered Multi-Step Tasks)**.** *A non-layered multi-step task is a task where appropriate steps can be identified such that the task can be divided, but a step might have more than one predecessor and/or successor. Given input space X, we again assume $i = 1,\ldots,H$ steps to be performed to solve the task, where a step is defined as before with $X^1 = X$, $Y^H = Y$, and $X^i = \bigcup_{Step^j \in PRE(Step^j)} Y^j$ if $|PRE(Step^i)| > 0$ and $X^i = X$ otherwise, where $PRE(Step^i) := \{Step^j | Y^j \cap X^i \neq \emptyset\}$, i.e. PRE returns all directly preceding steps of a step. The solution of the task for an input instance x is thus defined as $Solution(x,B) = \bigcup_{Step^i \in LAST(Task)} Step^i(x^i)$ where $LAST(Task) := \{Step^i | \forall Step^i \in Task : Step^i(x^i) \in y^H\}$ and $x^i \in X^i$ is recursively estimated. LAST returns all final steps of the multi-step tasks, i.e. all steps which are part of the solution.*

A crucial property of multi-step tasks is the number of steps H. We distinguish between two *conditions* for multi-step tasks: (i) It can be solved in exactly H or within $[1,H^+]$ steps where $H^+ \in \mathbb{N}^+$ is a predefined *upper bound* [1] and (ii) neither H or H^+ can be set *in advance* with *full confidence*.

Given that H or upper bound H^+ can be confidently estimated, we assume that each step of a task solution is of different nature such that a unique, finite solution can be found. In other words, all possible step sequences to solve the task are countable and finish in at most N^+ steps. Hence, if a step has to be repeated on the resulting output space, one cannot ensure

[1] H^+ could, for example, be set according to the longest possible sequence of steps to solve the task.

that a correct solution is generated in a finite number of steps, where only constraining or probabilistic upper bounds can be set.

We first define *finite multi-step tasks*, where condition (i) holds – i.e. we at least know an upper bound N^+ for the number of steps of a task. We additionally require all steps to be mutually *different* with regards to the respective input instances, as otherwise no upper bounds can be set. As a consequence, a step could still be repeated for a different input instance.

Definition 3.4 (Finite Multi-Step Tasks)**.** *A finite multi-step task is a multi-step task where the number of steps H or the upper bound of steps H^+ is known and fixed, and all steps are different, i.e. $\forall i, j$ with $i \neq j$ $Step^i \neq Step^j \vee x^i \neq x^j$.*

If condition (ii) holds, we cannot set H and/or H^+ confidently and deal with *infinite multi-step tasks*.

Definition 3.5 (Infinite Multi-Step Tasks)**.** *An infinite multi-step task is a multi-step task where H and H^+ are unknown, as they cannot be confidently estimated, and steps might have to be repeated for input instances.*

To illustrate the differences in finite- and infinite as well as layered and non-layered multi-step tasks, we discuss example tasks for each case in Example 3.1.

Example 3.1 (Multi-step tasks with different properties)**.** *Given our definition of multi-step tasks, four cases are possible. Each case has different real-world applications.*

- ***Finite Layered Multi-Step Tasks:*** *In NLP, numerous challenges involve to solve tasks with multiple, conceptually different steps in a processing pipeline. One such example is NERD, as introduced in Section 2.1. Each of the NERD steps has to be solved once for a task instance and all steps are conceptually different, as they tackle different problems.*

- ***Infinite Layered Multi-Step Tasks:*** *In Image Processing, one often has to solve tasks where steps have to be solved multiple times. One prominent example of such a step is applying image filters, where it is unknown (in advance) when the image was sufficiently processed. Here, the respective input instances (i.e. the processed images) usually differ for all steps.*

- ***Finite Non-Layered Multi-Step Tasks:*** *A simulation of, for example, a human organ such as the heart, requires a large and complex set of definitions with respect to materials or physical behavior, and additional patient-specific inputs such as segmented images or demographic information. The segmented images, for example, can either be manually created by domain experts, derived by an expert algorithm or a series of expert algorithms in a finite sequence. As a consequence, the overall multi-step tasks is not layered, but requires a more complex structure of steps.*

- ***Infinite Non-Layered Multi-Step Tasks:*** *We can directly extend the finite non-layered multi-step task example for simulation to the infinite case. For the latter, we might define an infinite layered multi-step task for image segmentation, where a more elaborate sequence of steps enables to apply respective filters multiple times to also normalize the image. Since now a subset of the former finite multi-step task is infinite, the complete overall multi-step task becomes infinite.*

Note that task properties (i.e. finite, infinite, layered or non-layered) are often dependent on design choices for steps and/or their solution functions.

An important observation is the following: Our task definitions are recursive, as a task is dependent on its direct predecessors. In case of the non-layered property, there are thus no *exhaustive* assumptions on the *absolute order* of solving steps (e.g. they might occur in parallel). When trying to determine the absolute order (or plan) of solving the task, there is a critical distinction to make. In *workflow systems*, one would derive plans where *parallel executions* are specifically modeled or retrieved (e.g. via a graph). In a *decision-theoretic* perception, where actions need to be actively taken to reach a goal state (see MDPs in Section 3.3.2), a *single* action needs to be taken per time step. Depending on how the environment is modeled, one can represent multiple step combinations as single actions, but the underlying assumption is different. A *combined task action* (which would simulate parallel steps in a workflow) is only chosen if there is positive feedback from the environment, i.e. if there is sufficient added value (e.g. due to time savings). On a final note, the two perceptions are *complementary* (thus *not* conflicting), as one could derive optimized workflow plans *after* having optimized the decision-theoretic value of available actions.

To this end, central properties of a number of solutions for task instances are *optimality* and *near-optimality*. *Optimality* with respect to a task solution function and a set of input instances of the task entails that the respective solution function *must not make any mistakes*. *Near-optimality*, on the other hand, assumes a predefined threshold which defines the number of allowed mistakes for a solution function. Note that we define optimality and near-optimality for the solution of the *complete multi-step task*. The concepts could, in principle, be similarly defined for step solution functions, but would overly restrict task solutions, as only the overall hypothesis has to be correct.

Definition 3.6 (Optimality and Near-Optimality for Task Solutions)**.** *Given a task (X,B,Y) with set of input instances $x_1,\ldots,x_n \in X$, let $y_1^*,\ldots,y_n^* \in Y$ be the correct solutions with respect to the task. A solution function $Solution^*$ is optimal with respect to $x_1,\ldots,x_n$ if it outputs $y_1^*,\ldots,y_n^*$ as set of solutions, i.e. $\forall x_i$ with $i \in 1,\ldots,n : Solution^*(x_i,B) = y_i^*$. A solution function Solution is near-optimal with respect to $x_1,\ldots,x_n$ if it outputs the correct solution for $k \leq n$ input instances (with $k \in \mathbb{N}^+$) and $n-k < \varepsilon$ with $\varepsilon \in \mathbb{N}^{\geq 0}$ a predefined error threshold, i.e. $|\{x_j | x_j \in x_1 \ldots x_n \wedge Solution(x_j,B) = y_j^*\}| = k$.*

The property is often used for methods in supervised learning and decision theory, and thus similarly holds for subsequent concepts. We do not attempt to prove any values or boundaries

for ε (e.g. showing that a solution function always reaches a certain correctness). We rather assume that absolute performances of solution functions (or relative improvements of solution functions with respect to others) have to be shown and reasoned upon empirically, i.e. on available input instances.

The thesis deals with solving multi-step tasks assuming available *experts*. The latter are – as mentioned in Chapter 1 – algorithms which are able to solve steps for multi-step tasks. Expert hypotheses are, however, not always correct for each input instance. Given exchangeable experts for a given step, one thus needs to *learn* when to choose which proposed hypothesis or how to combine the available hypotheses in order to solve the complete tasks, by making as few mistakes as possible.

Learning about experts can be approached by using available *training sets*. *Supervised learning* is a central branch of methods for using training sets to generalize to novel, unseen input instances, which we deal with next. After having introduced basic concepts and approaches of supervised learning, we deal with selected *decision-theoretic* elements to deal with sequential decisions as well as missing information (e.g. due to expert budget constraints). We specifically define *experts* in Section 3.3.1 after having introduced the necessary concepts.

3.2 Supervised Machine Learning

In supervised learning, one assumes training sets (referred to as *labeled data*) for *predicting* diverse outcomes, such as the assignment of classes or numerical values to an input instance (referred to as *data point*). The general supervised learning paradigm can be defined as follows.

Definition 3.7 (General Supervised Learning Paradigm)**.** *Given data Ξ, which might be represented as graph, set or otherwise and given samples $(x_i, y_i) \in \Xi$, we wish to learn a function (referred to as model) $f : X \to Y$. Here, x_i is referred to as data point or input instance, and y_i is referred to as dependent variable or label and defines the correct solution for x_i for the current task. Given random variables (referred to as features) $x_i^{(1)}, \ldots, x_i^{(m)}$ with $x_i^{(j)} \in \mathbb{R}$, one has to learn model parameters $\phi^{(j)}$ for $j = 1, \ldots, m$ with $\phi^{(j)} \in \mathbb{R}$ where the prediction, $f(x_i)$, is dependent on products of random variables and model parameters, i.e. $x_i^{(j)} \phi^{(j)}$. The loss of a series of predictions of f for samples $x_1, \ldots, x_n$ quantifies the correctness with respect to y_i, where the loss is zero if $x_i = y_i$. A loss function $\mathbb{L} : \phi \times X \times Y \to \mathbb{R}$ maps predictions $f(x_i)$ to a real number.*

A *function* (or *model*) thus consists of *random variables* and respective *model parameters*, where we *observe* values of random variables and have to *learn* their model parameters to explain them (see Section 3.2.4 for learning regression models). The relationship among model parameters might range from simple (e.g. linear) to complex (e.g. non-linear).

To this end, models might also have *hyperparameters*, which are essentially different from model parameters, and might influence both how the model parameters are learned and what the model predicts for a data point.

Definition 3.8 (Hyperparameter)**.** *A **hyperparameter** defines design choices for a model, e.g. value thresholds for predictions or categories given multiple available learning or prediction methods, and can thus be a real number, String or otherwise. A hyperparameter, in general, has to be empirically assigned based on the current task and available input instances of* Ξ*.*

As supervised learning is a very broad research field, we only highlight and introduce selected concepts which are relevant to this thesis. We first discuss differences in assumptions with regards to available data and then deal with the learning process itself – namely how *learners* (i.e. the instance controlling the respective model) receive training data and what this implies for the resulting models.

3.2.1 Model-specific Assumptions

Given data Ξ and a model f, there are two opposing assumptions. The most common assumption is that the random variables of f based on Ξ are *independent and identically distributed* (i.i.d.).

Definition 3.9 (Independent and identically distributed random variables)**.** *The i.i.d. assumption stipulates that random variables used for a model and instantiated by data* Ξ *are (i) independent, i.e. do not have any causal effects on each other, and (ii) identically distributed, i.e. share the same distribution with the same model parameters (e.g. same mean value for a normal distribution).*

While the assumption is strong, it enables to (theoretically-) study the generalizability of learned models to unseen data in terms of *performance bounds* (i.e. general statements on near-optimality). However, the assumption might be inadequate if data distributions are changing over time. To this end, *adversarial learning* takes the opposite stance and does not take any assumptions on random variables.

Definition 3.10 (Adversarial Learning)**.** *The adversarial assumption stipulates that no assumption can be made about the generating distribution of random variables.*

Note that this has essential consequences on the ability to learn from data. We discuss a central adversarial learning approach in Section 3.3.1 and now deal with possible learning protocols, which are closely connected to model-specific assumptions and determine how many training samples we are allowed to perceive before *learning* or *updating* a model.

3.2.2 Learning Protocols

In general, one can distinguish between two learning protocols – *batch* and *online*. In the *batch learning* setting, we assume all relevant training data points to be available before learning a model, while in the *online learning* setting, we receive the labels gradually throughout time. These differences occur for several reasons: (i) data sets might consist of a large number of training samples which might eliminate the possibility to use them at once, (ii) training samples might be available in streaming settings, where updates should happen as fast as possible, and lastly (iii) training data might have to be generated by interaction (e.g. executing learned models to observe their hypotheses), where actions have to be taken in sequential order to reach a given goal.

To ease comparison between the settings, we assume that the set of data Ξ with data points (x_i, y_i) is ordered by the timely occurrence of the latter, i.e. data points with smaller indexes occur earlier. To highlight this property, we assume that data points occur in *episodes* $z \in 1, \ldots, Z$ with $Z \in N^+$. We use episodes as subscript for data points, i.e. (x_z, y_z) is the data point with label at episode z. Note that an episode is different from a step in a multi-step task – the latter deals with a distinct problem with respective input, background knowledge and output spaces and the former is merely the timely ordering of labeled data points.

In the *batch learning setting*, it is generally easier to learn models with high *predictive performance* (i.e. low loss for unseen data points), if the i.i.d. assumption holds, as one is allowed to observe the complete training set.

Definition 3.11 (Batch Learning)**.** *Given data points and labels observed until episode z, i.e.* $(x_1, y_1), \ldots, (x_z, y_z)$*, learn a model* f *using all labeled data points at once to minimize the loss of* $f(x)$*.*

For *online learning*, one gradually updates models for individual data points (i.e. for each sample one first makes a prediction and then receives the corresponding label). Here, both i.i.d. as well as adversarial approaches exist.

Definition 3.12 (Online Learning)**.** *Given episode z, only use* (x_z, y_z) *to learn or update a model* f *to minimize the loss of* $f(x_z)$*.*

Finally, an *intermediate scenario* exists, where a model is gradually updated by a *mini-batch*, i.e. a subset of all available samples is used at once. *Mini-batch learning* thus entails to either wait for a predefined number of samples to update the model or to gradually receive the mini-batches at once. Compared to online learning, exploiting samples in mini-batches can improve the predictive performance of a model. Again, both i.i.d. as well as adversarial approaches exist.

Definition 3.13 (Mini-batch Learning)**.** *Given episode z and batch size* $Z_{BATCH} \in \mathbb{N}^+$*, use samples* $(x_{z-Z_{BATCH}}, y_{z-Z_{BATCH}}, \ldots, x_z, y_z)$ *if* $z \mod Z_{BATCH} = 0$ *to learn or update a model* f *in order to minimize the loss of* $f(x)$*.*

Similar to batch and online learning protocols, there are different *prediction settings*, which we now discuss.

3.2.3 Prediction Settings

The standard prediction setting in supervised learning is that a *single input instance* x_z is given to the learner in every episode z (referred to as *single label prediction*). However, it might be beneficial to exploit the knowledge of seeing multiple input instances at once. This setting is referred to as *pool-based prediction*. The latter is especially useful when relations among data points are available (see Link Prediction in Section 3.2.4).

We first deal with the single label prediction case, where a classic example is linear regression, which we deal with in Section 3.2.4.

Definition 3.14 (Single label prediction)**.** *In the single label prediction case, the learner receives data point x_z at episode z and makes prediction $f(x_z)$.*

For pool-based prediction, we receive a predefined number of input instances at once to predict outcomes their outcomes.

Definition 3.15 (Pool-based prediction)**.** *Given pool interval number $Z_{POOL} \in \mathbb{N}^+$ and episode z, the learner receives data points $x_z, \ldots, x_{Z_{POOL}}$, makes predictions $f(x_z), \ldots, f(x_{Z_{POOL}})$ at once.*

An important application of the pool-based prediction setting is *collective inference* in *Graphical Models*, where links among graph entities are exploited to generate consistent hypotheses for multiple properties (see Definition 3.22).

We now revisit selected *instantiations* of mentioned learning scenarios, namely *Regression* and *Link Prediction*.

3.2.4 Selected Learning Scenarios

We first deal with *Regression* and assume data points $x \in X$ to be represented as vectors, which often consist of manually engineered features. We then turn to *Link Prediction* in graphs, where we also deal with *relational learning*.

Regression

We first define the general regression setting, where an input instance represented as vector has to be mapped to a numerical value.

Definition 3.16 (Regression, Squared loss)**.** *Regression entails to map an input instance x_i to a real valued number, i.e. to learn a function $f : X \to \mathbb{R}$. In the multivariate case, we assume each data point to be represented as feature vector of length N^{FEAT}, i.e. $x =< x^{(1)}, \ldots, x^{(N^{FEAT})} >$, where $x^{(i)}$ are random variables. A standard loss function for regression is the squared loss $\mathbb{L}_{SQUARED}(\phi, x, y) = \|\phi^T x - y\|_2$*

In the linear case, we search for a *linear combination* of model parameters based on available random variables.

Definition 3.17 (Linear Regression)**.** *Linear regression entails to find model parameter vector* $\phi^* = w^{(1)}, \ldots, w^{(N^{FEAT})}$ *such that*

$$f(x) = \phi^T x = w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \ldots + w^{(n)}x^{(N^{FEAT})} = y \tag{3.1}$$

We focus on standard linear regression and discuss *gradient solutions* for batch-, mini-batch and online learning, more specifically *gradient descent*.

Definition 3.18 (Gradient Descent, Stochastic Gradient Descent, Batch Gradient Descent)**.** *When given samples* (x_i, y_i)*, we seek to minimize the gradient of an arbitrary loss function* $\mathbb{L}(\phi, x_i, y_i)$.

- *Gradient descent (GD) refers to the online update, where the model is individually updated with every sample based on a learning rate hyperparameter* $\eta^{GD} \in \mathbb{R}^{\geq 0}$*, i.e.*

$$\phi = \phi - \eta^{GD}\nabla_\phi \mathbb{L}(\phi, x_i, y_i) \tag{3.2}$$

- *Stochastic gradient descent (SGD) extends the concept of GD by sampling a number of labeled data points* $N^{IT} \in \mathbb{N}^+$ *for individual updates, where one update is referred to as* iteration*. The advantage is that not all data points have to be used for learning.*

- *In batch gradient descent (BGD), a batch of samples* $(x_1, y_1), \ldots, (x_Z, y_Z)$ *collected until episode* Z *is used for a single update, i.e.*

$$\phi = \phi - \eta^{BGD}\nabla_\phi \frac{1}{Z}\sum_{i=1}^{Z} \mathbb{L}(\phi, x_i, y_i) \tag{3.3}$$

  For BGD to converge, one usually repeats batch updates in epochs*, where the number of epochs is a hyperparameter and defined as* $N^{EPOCH} \in \mathbb{N}^+$*. Note that for standard GD, single updates might also be repeated in epochs.*

To calculate the gradient updates, we assume available random variables to be i.i.d. and need to take partial derivatives for each variable in the model, i.e. $\forall \phi^j \in \phi$.

Complexity Analysis 3.1. *GD and SGD are not dependent on the size of training data sets, but on the dimensionality of the feature representation, and the number of epochs or iterations. The computational complexity in Big O notation for GD is thus within* $O(|N^{FEAT}||N^{EPOCH}|)$ *and for SGD within* $O(|N^{FEAT}||N^{IT}|)$*. Batch GD updates, on the other hand, are additionally dependent on the size of training data sets. The resulting complexity lies within* $O(|N^{EPOCH}||N^{FEAT}||\Xi|)$.

Link prediction for graphs is related to regression, as one wants to estimate the probability of an edge to be part of a graph. It can be approached by *Relational Learning*, for which we now describe a simple approach.

Link Prediction

Link prediction is an important problem setting, where one aims to predict if two nodes in a graph are connected via a specific relation. As graphs on the Web often comprise multiple relations, we first define *multi-relational graphs* (MRG) as basic representation for *Relational Learning*.

Definition 3.19 (Multi-Relational Graph)**.** *Let $G = (V, \mathbf{R})$ be a MRG with set of nodes V and family of typed edge sets $\mathbf{R}$ of length m, i.e. $\mathbf{R} = \{R_0, \ldots, R_m \subseteq (V \times V)\}$ where $r = 1, \ldots, m$ denote the unique relations in the graph.*

MRGs frequently occur on the Web (e.g. in *social networks* or the *Linked Open Data Cloud* [2]) and are often used for analysis.

We can now define the linked prediction problem for MRGs.

Definition 3.20. *Given MRG $G = (V, \mathbf{R})$, the goal is to predict a fixed relation $r \in R_i$ with $R_i \in \mathbf{R}$ for pairs of nodes $(v_i, v_j) \in V \times V$ with $i \neq j$, i.e. learn function $f^{LP} : V \times V \times R \to [0, 1]$.*

A straightforward *Relational Learning* approach to learn f^{LP} is using supervised learning approaches for vector-based representation (i.e. by reducing the graph structure).

Definition 3.21 (Relational Classifier for Link Prediction)**.** *For nodes (v_i, v_j), a transformation function $f^{TRANS} : V \times V \to X$ has to be developed, where X is a fixed-length feature space. Based on the derived vector-based representation X, any adequate approach can be used to learn regression functions for Linked Prediction, i.e. $f^{LP} : X \to [0, 1]$.*

Such approaches, however, *lower* the graph structure and thereby lose *expressivity*, as the relation among nodes in the graph are not fully exploited. The class of *Lifted Graphical Models* (LGMs) [68] approaches this shortcoming by extending *Probabilistic Graphical Models* (PGMs) [71] with a relational structure, yielding a *Statistical Relational Learning* (SRL) approach. LGMs thus exploit declarative representations of knowledge, where one manually models relations among nodes before any learning takes place. Pool-based predictions then enable to *jointly* predict new relations in the graph for a number of nodes, referred to as *collective inference*. We use a LGM for collective inference to solve the learning problem in Chapter 5.

Definition 3.22 (Collective Inference)**.** *Given a pool-based prediction setting and a MRG $G = (V, \mathbf{R})$, collective inference exploits the graph structure of G to generate joint predictions for a pool of nodes $v_1, \ldots, v_{Z_{POOL}} \in V$ of predefined size $Z_{POOL} \in \mathbb{N}^+$.*

[2] `http://lod-cloud.net/` (*accessed on 05/01/2018*)

Given that a model has been learned for learning scenario, it is important to *evaluate* it in order to quantify its predictive performance. We now introduce different *performance metrics* for evaluating models.

3.2.5 Evaluating Learned Models

Evaluating learned models can be approached *theoretically* or *empirically*. In this work, we focus on *empirical* evaluations and use training sets to quantify the *predictive performance* of models.

To generate *representative* results for empirical evaluations, one can either (i) *bootstrap* samples of a predefined size or (ii) perform *cross-validation*. We use cross-validation for our evaluations, as available training sets are sufficiently large in order to divide them into *folds* (i.e. disjoint subsets of samples).

Definition 3.23 (Cross-Validation)**.** *Given training set Ξ, cross-validation entails to split Ξ into K folds Ξ_k with $k \in 1,\ldots,K$ of size $\frac{|\Xi|}{K}$ to eventually test a given model f on each of the K folds by learning K sets of model parameters ϕ_k. One uses $K-1$ folds for learning model parameters ϕ_k for f to validate the predictions on the left out fold. The process is repeated K times to validate the performance of f on all available samples.*

For the multi-step tasks used in our evaluations in Chapter 9, established metrics for the learning problem are *precision*, *recall* and *F1-measure*.

Definition 3.24 (Precision, Recall, F1-measure)**.** *For the binary classification problem (with $0-1$ loss function $\mathbb{L}_{0\text{-}1}(\phi,x,y) = \mathbb{1}_{\{x \neq y\}}$) with possible outcomes $Y = \{True, False\}$, samples $(x,y) \in \Xi$ and learner prediction $f(x)$, we define the followings sets:*

- *True positives TPs:* $\{x | x \in \Xi \wedge f(x) = y \wedge y = True\}$
- *True negatives TNs:* $\{x | x \in \Xi \wedge f(x) = y \wedge y = False\}$
- *False positives FPs:* $\{x | x \in \Xi \wedge f(x) \neq y \wedge y = False\}$
- *False negatives FNs:* $\{x | x \in \Xi \wedge f(x) \neq y \wedge y = True\}$

The precision of f given Ξ is defined as $Precision = \frac{|TPs|}{|TPs+TNs}$
The recall of f given Ξ is defined as $Recall = \frac{|TPs|}{|TPs|+|FNs|}$
The F1-measure of f given Ξ is defined as $F1 = 2\frac{Precision\ Recall}{Precision+Recall}$

True positives as well as true negatives thus are correct predictions of a model, and false positives and false negatives are the sets of its wrong predictions. The mappings from task instances to positives or negatives is task dependent and will be pointed out in the evaluation chapters.

We finally discuss a branch of approaches to combine available learned models for the same step (i.e. same input and output spaces) to improve the overall performance.

3.2.6 Learning Combination Functions over Models

Given that a number of models for the same step exist, a combination of all models might yield better results than the best single model *in hindsight*. The field of *Ensemble Learning* studies how to learn combination functions over *exchangeable* models. Here, one usually perceives individual models as *black-box models* or *black-box functions*, as the respective learning approach is not taking into account. To learn combination functions, one usually assumes individual models to be *mutually independent*, such that resulting feature spaces are i.i.d. .

Definition 3.25 (Ensemble Learning)**.** *Given N black-box models $f_1, \ldots, f_N$ and data set Ξ for a step, where $f_i : X \to Y$ predicts a hypothesis for a step, the goal of ensemble learning is to learn a joint function $\hat{f} : f_1 \times \ldots \times f_N \times X \to Y$.*

Prominent approaches comprise *boosting* [17, 48] where additive joint functions are learned, *mixture of experts* [58] where a *softmax layer* (i.e. models are weighted exponentially to their estimated performance) switches among outputs of individual models or *stacking* [143] where the joint function is learned with generated data sets based on cross-validated black-box models.

With respect to the contributions of this thesis, we focus on stacking.

Definition 3.26 (Stacking / Stacked Generalization)**.** *Given N models $f_1, \ldots, f_N$ and data set Ξ, as in the Ensemble Learning setting, the high-level protocol of stacking follows:*

- *For $f_1, \ldots, f_N$, perform K-fold cross-validation on Ξ.*
- *Use the resulting model-dependent data set $\{((f_1(x), \ldots, f_E(x)), y) | \forall (x,y) \in \Xi\}$ to learn joint function $\hat{f}$.*

Based on prior introduced topics in supervised learning, we now turn towards frameworks for *decision-making*.

3.3 Decision-Making

Other than in the standard supervised learning setting, we now deal with learning settings where (i) feedback is limited and only available for actively chosen hypotheses and/or (ii) multiple decisions have to be taken in a sequence for a single task instance. Taking standard supervised learning as baseline, an *agent* (i.e. the decision-maker) has to take a single *action* (i.e. predict a hypothesis) for a *state* (i.e. a data point or input instance) and gets *full feedback* (i.e. receives the label). We, from now on, use the notion of states and actions to refer to data points and predictions. Using this terminology eases the integration of prominent works in supervised learning, single-step decision-making and multi-step decision-making.

Relevant frameworks can be categorized into *one-step* and *multi-step* decision-making as well as *full feedback* and *partial feedback*. We begin by discussing *single-step* settings which assume *expert advice* (i.e. several available models to generate hypotheses) and then turn to *multi-step* decision-making, where we deal with general decision-making aspects, as no prior work in multi-step expert advice exists. We finally deal with general *Multiagent multi-step* decision-making, where the control of the *decision process* is distributed among several independent, cooperating agents.

3.3.1 Prediction with Expert Advice

Prediction with expert advice [27] is a prominent problem setting, where multiple *experts* suggest actions (i.e. their hypotheses) to an agent, which has to choose among the latter. We first define experts based on concepts from supervised learning.

Definition 3.27 (Expert)**.** *An expert for a step in a multi-step task is the equivalent of a black-box function f_i (see Definition 3.25), which is not constrained to supervised learning techniques – it might also, for example, be a heuristic or simple rule. Given a state space S and an action space A, an expert is a function $e : S \to A$, which suggests an action $a \in A$ for a state $s \in S$.*

We now present the general setting of *prediction with expert advice*, where we take into account possible *budget constraints* with regards to feedback. Note that, while all subsequently presented settings within the general expert advice setting are extensively studied in the literature, a formalization of the general framework is novel with respect to this thesis.

Definition 3.28 (General Expert Advice Setting)**.** *The general expert advice setting is a* 6*-tuple* $(S, E, A, R, C_{EXP}, C_{FEEDBACK})$.

- *Let S be the set of states.*
- *Let $\Gamma \in \Delta(S)$ be the state distribution, which is strongly dependent on the underlying assumptions of the data distribution (i.e. i.i.d. vs. adversarial).*
- *Let A be the set of actions.*
- *Let E be the set of experts, where an expert is a function $e : S \to A$, proposing an action for a state.*
- *Let $w_e : S \to \mathbb{R}$ be the weight function for an expert $e \in E$ which returns an estimate of its predictive performance for a state.*
- *Let $C_{EXP} \in \mathbb{N}^+$ be the expert budget, which determines how many experts the agent is allowed to execute per state.*

- *Let $C_{FEEDBACK} : S \times A \to \{E\}$ be a function which return the set of experts the agent receives feedback for.*
- *Let $R : S \times A \to \mathbb{R}$ be the reward function for an action given a state, which quantifies its correctness (i.e. similar to the inverse of a loss function in supervised learning).*
- *Let $\pi : S \to A$ be the policy which depicts the eventual action the agent chooses.*

The protocol of the general expert advice setting proceeds in episodes $z \in 1, \ldots, Z$ with Z as defined before, where in each episode a single state s_z is revealed to the agent.

- The agent perceives s_z.
- The agent gets estimates w_e for all available experts.
- Given the expert budget C_{EXP}, the agent executes a number of experts $e_1(s_z), \ldots, e_{C_{\text{EXP}}}(s_z)$ and perceives their suggested actions.
- The agent chooses an action based on its expert weight estimates.
- The agent is revealed the reward of the chosen action, $R(s,a)$, and, based on the current feedback setting, of expert set $C_{\text{FEEDBACK}}(s_z, a_z)$.

The goal is to maximize the expected cumulative reward R^{CUM} with respect to the chosen actions.

Definition 3.29 (Cumulative Reward for Prediction with Expert Advice)**.**

$$R^{CUM} = \mathbb{E}[R(s,a)] \tag{3.4}$$

We now describe different expert advice settings, where budget C_{EXP} and feedback constraint function C_{FEEDBACK} are varied.

Expert Advice with Full Information

We begin by discussing the *full information* setting, where the agent is allowed to query all available experts and receives feedback about all available actions.

Definition 3.30 (Prediction with Expert Advice)**.** *Given the general expert advice setting, we set $C_{EXP} = |E|$ and $C_{FEEDBACK}(s,a) = |E|$.*

A central algorithm which solves the setting is *Exponential Weights/Hedge* (EWH), which is an adversarial learning approach. The algorithm sets the baseline for numerous extensions (e.g. [3, 6, 15]). We now introduce EWH in its simplest form.

Definition 3.31 (Exponential Weights/Hedge algorithm)**.** *To update the weight of an expert w_e, EWH assumes a scaled reward function $R^{SCALED} : S \times A \to [-1,1]$ where rewards in R are mapped to $[-1,1]$, a learning rate hyperparameter $\eta^{EWH} \in N^+$ and initial weight $w_e(s_1) = 1.0$. The update rule is then defined by:*

$$w_e(s_{z+1}) = w_e(s_z)\ exp\big(R^{SCALED}(s_z, e(s_z))\ \eta^{EWH}\big) \tag{3.5}$$

The algorithm thus updates weights of experts exponentially to the scaled reward observed for their actions. The decision for a step is then based on probability distribution $P : S \times A \to [0,1]$ for all available actions:

$$P(s_z, a_z) = \frac{\sum_{e \in E} 1_{\{e(s_z)=a_z\}} w_e(s_z)}{\sum_{e \in E} w_e(s_z)} \tag{3.6}$$

Based on the probability distribution, the greedy policy $\pi(s_z) = \max_{a_z} P(s_z, a_z)$ chooses the action with the highest probability.

Complexity Analysis 3.2. *As EWH is an algorithm for online learning, we measure its complexity in terms of experts and not episodes. As all experts are executed for each episode, the complexity in Big O notation is $O(|E|)$.*

Note that for decision-theoretic algorithms, one often deals with *sample complexity* and thus aims to provide lower bounds for the predictive performance, which we do not deal with in this thesis.

We now briefly discuss different constrained variations of prediction with expert advice.

Expert Advice with Constraints

A natural specialization of prediction with expert advice is constraining the number of experts the agent is allowed to execute for a state. In this setting, the agent only receives rewards for *executed experts*, as the residual expert hypotheses remain unknown. The problem can be approached by an extension of the EWH algorithm [3], where all expert combinations of size C_{EXP} are weighted.

Definition 3.32 (Prediction with Budgeted Expert Advice)**.** *Given the general expert advice setting, we set $C_{EXP} \leq |E|$ and $C_{FEEDBACK}(s,a) = \{e | \exists a = e(s)\}$ returns all experts which have been executed for the state.*

Another specialization for expert advice is related to *contextual bandits* [6], where an agent only receives a reward for the *action* it chooses. A prominent solution to contextual bandits with expert advice is the *EXP4.P* algorithm [15], which extends EWH with respect to the *exploration-exploitation* problem. The latter is central for *single-* and *multi-step partial label* problems (where partial labels result from constrained information). In the context of expert

advice, agents need to successfully balance *exploiting* experts with high weights and *exploring* experts with low or unknown weights.

Definition 3.33 (Prediction with Expert Advice for Partial Feedback)**.** *Given the general expert advice setting, we set* $C_{EXP} = |E|$ *and* $C_{FEEDBACK}(s,a) = \{e|e(s) = \pi(s)\}$.

We now deal with *multi-step* decision-making problems, where *sequential* decisions have to be taken in a single episode.

3.3.2 Markov Decision Processes

If an agent has to choose several *sequential* actions for a single episode, the problem can be formalized as MDP [109], where the goal is to find an optimal policy for *acting* in the world. We first define the general MDP setting.

Definition 3.34 (General Markov Decision Processes)**.** *A General MDP is a* 6*-tuple* (S,A,γ,Tr,R,H).

- *Let S be the set of world states and A the set of actions, as defined for expert advice.*
- *Let* $\Gamma \in \Delta(S)$ *be the start state distribution, as defined for expert advice.*
- *Let*$\gamma \in [0,1]$ *be the discount factor hyperparameter.*
- *Let* $Tr : S \times A \to S$ *be the transition function, which depicts the resulting state of the agent after taking an action.*
- *Let* $R : S \times A \to \mathbb{R}$ *be the reward function, which returns the local feedback for the taken action, as defined for expert advice.*
- *Let H be the horizon, which defines the number of actions to be taken in an episode and is defined differently for respective instantiations of the General MDP setting.*
- *Let* $\pi : S \to A$ *be the policy, which depicts the action the agent chooses, as in the expert advice case.*

The protocol of a general MDP proceeds in episodes $z \in 1,\ldots,Z$ with Z as defined before, where in each episode a single start state s_z is revealed to the agent.

- For each time step $h \in 1,\ldots,H$, the agent acts based on the state of the current episode and time step s_z^h, where $s_z^1 = s_z$.
  - The agent perceives the current world state s_z^h.
  - The agent chooses action a_z^h.
  - Based on the transition function Tr, the agent moves to the next state s_z^{h+1}.

– The agent receives reward $R(s_z^h, a_z^h)$ for its action.

Based on the definition of General MDPs, we now define two cases, namely *infinite-horizon MDPs* and *finite-horizon MDPs*.

Definition 3.35 (Infinite-Horizon MDP)**.** *An infinite-horizon MDP is a General MDP with $\gamma = [0,1]$ and $H = \infty$. In this case, an agent might be taking an infinite amount of actions.*

Definition 3.36 (Finite-Horizon MDP)**.** *A finite-horizon MDP is a General MDP with $\gamma = 1$ and $H = \mathbb{N}^+$. In this case, the number of decisions is constrained by H and the discount factor hyperparameter is not required.*

The essential properties of MDPs are as follows:

- **Markovian states:** Taking an action in a MDP only depends on the current state.
- **Rewards:** Rewards are numerical values which quantify the *local* value of an action in a state.
- **Stationary versus non-stationary policies:** If the policy is dependent on the time step, it is *non-stationary*. This might be the case for finite-horizon MDPs, as one knows about the maximal number of actions, and can define reward- and transition functions over time steps or directly learn time step-dependent policies.
- **Full observability:** We assume that we can observe all relevant *features* of a state to take optimal actions.

The *global quality* of states and *state-action pairs* (i.e. an action taken in a state) are estimated by *value functions*. Based on the latter, one can determine the policy based on an *action selection strategy*, which aims to balance exploration and exploitation.

Definition 3.37 (Value functions)**.** *The state value function $V : S \to \mathbb{R}$ quantifies the value of a state, while the state-action value function or Q-function $Q : S \times A \to \mathbb{R}$ depicts the quality of state-action pairs.*

The Bellman equations [11] define *optimal* properties of value functions and constitute the principles of *dynamic programming*.

Definition 3.38 (Bellman Equations)**.** *The value of a state is set according to the current reward of the policy and the values of future state values, based on the transition probabilities. Hence, given action a, where $\pi(s) = a$:*

$$V(s^h) = \max_{a^h \in A} R(s^h, a^h) + \gamma \sum_{s^{h+1} \in S} Tr(s^{h+1} | s^h, a^h) V(s^{h+1}) \tag{3.7}$$

Similarly, state-action value function (also referred to as the Q-function) expresses the current reward and the maximal future state-action value, based on the transition probabilities:

$$Q(s^h, a^h) = R(s^h, a^h) + \gamma \sum_{s^{h+1} \in S} Tr(s^{h+1}|s^h, a^h) \max_{a^{h+1} \in A} Q(s^{h+1}, a^{h+1}) \tag{3.8}$$

The goal of a MDP is to maximize the reward of the decision process. The latter is defined as expected cumulative reward *received* over all actions taken (see Definition 3.39).

Definition 3.39 (Cumulative Reward for MDPs)**.**

$$R^{CUM} = \mathbb{E}[\sum_{h=1}^{H} R^h(s^h, a^h)] \tag{3.9}$$

For an *inexperienced* agent to learn how to act, the environment needs to be *explored*. *Learning by interaction* methods are referred to as RL, where the agent has to learn how to deal with partial feedback for sequences of actions.

Reinforcement Learning

RL [126] studies how to *learn* to behave by *interacting* with the environment on a *trial-and-error* basis (see Definition 3.40).

Definition 3.40 (Reinforcement Learning)**.** *Given a MDP, RL is the problem of an agent having to learn a policy π by taking actions $a \in A$ in states $s \in S$ and receiving numerical rewards $R(s,a)$ in order to maximize the expected cumulative reward.*

To this end, the most common way to classify RL approaches is distinguishing between *Policy Search RL*, *Value-based RL* and *Model-based RL*. We now briefly discuss each type.

Policy Search RL In *Policy Search RL* (e.g. [141]; see [99] for an overview), one aims to directly learn policy π and thus omits to learn a value function. This is achieved by observing complete *trajectories* of a given policy (i.e. a predefined number of transitions $s^1, a^1, r^1, s^2, \ldots, s^H, a^H, r^H, s^{H+1}$) and then learning the model parameters of a policy. Similar to contextual bandit algorithms, action selection strategies are directly integrated into the learning process. We present a Policy Search RL algorithm in Sec. 6.6.

Value-based RL Approaches in the class of *Value-based RL* (e.g. [140]) directly learn the value function for states (V) or state-action pairs (Q). Other than in Policy Search RL, one needs an explicit action selection strategy (i.e. how to use the learned value functions with respect to the eventual policy) – e.g. the greedy policy, $\pi(s) = \max_{a \in A} Q(s,a)$, where the action with maximal state-action value is always selected. We present two Value-based RL algorithms in Section 5.4 & 5.5.

Note that there are hybrid methods using both Policy Search- and Value-based RL, referred to as *Actor-Critic RL* algorithms (e.g. [94, 115]), which we do not deal with in this thesis.

Model-based RL *Model-based RL* methods (e.g. [21, 64, 87]) directly learn the reward function R and transition function Tr from which one can derive V and Q with MDP Planning techniques. Here, *model* refers to a *MDP model*, consisting of the available states as well as approximations for the reward- and transition function, and is not to be confused with a model in supervised learning. Exploration is usually approached by measuring the uncertainty or estimated error in the current MDP model approximation. Model-based RL algorithms are not covered in this thesis.

To this end, an important property of RL algorithms is whether they deal with *discrete* or *continuous* MDP action spaces. In *discrete action spaces*, one can enumerate all actions and use the Bellman equations (see Definition 3.38) to calculate the state- or state-action values. For *continuous action spaces*, one could still discretize the available actions and follow the same schema, but direct calculation of state values is not possible. Policy Search as well as Actor-Critic RL algorithms can be directly adapted to continuous action spaces [133], while Value-based RL algorithms are usually constrained to the discrete case. For Model-based RL, works on continuous action spaces exist (e.g. [32]), which usually need more samples to converge.

Finally, the last dimension for RL algorithms relevant to our work is the *update frequency* – i.e. distinguishing between *Online-* and *Batch RL* approaches. The following definitions thus align with Section 3.2.2. We begin by defining the *Online RL* setting.

Definition 3.41 (Online RL)**.** *Given transition* (s^h, a^h, r^h, s^{h+1}) *at time step h, use* (s^h, a^h, r^h, s^{h+1}) *to update approximations* R, Tr *for Model-based RL,* Q, V *for Value-based RL or* π *for Policy Search RL.*

The Online RL setting thus requires agents to conduct immediate updates of the respective functions. In *Batch RL*, function updates are deferred for a predefined number of time steps, for which the agent follows the current policy.

Definition 3.42 (Batch RL)**.** *Given batch hyperparameter* $Z_{BATCH} \in N^+$ *and transitions* $(s^{h-Z_{BATCH}}, a^{h-Z_{BATCH}}, r^{h-Z_{BATCH}}, s^{h-Z_{BATCH}+1}, \ldots, s^h, a^h, r^h, s^{h+1})$ *at time step h, use* Z_{BATCH} *transitions to update approximations* R, Tr *for Model-based RL,* Q, V *for Value-based RL or* π *for Policy Search RL, given* $z \mod Z_{BATCH} = 0$.

Another central problem in MDPs is the one of *Planning*, where a learned MDP model exists but the policy has to be *calculated*.

Planning

Given that all possible states and actions have been explored or that we learned a MDP model of the environment, a policy can be found by *Planning* in the MDP.

Definition 3.43 (Planning in MDPs)**.** *Given a model of MDP, $M = \langle S, A, \gamma, Tr, R \rangle$, planning is the problem to find a policy π by only using the M, without interacting with the world.*

Value iteration [11] is a straightforward technique to derive an *exact solution* for MDPs with *tabular state representations*. The latter is inefficient for storing large state spaces (as states are enumerated), but describes the baseline for Planning in MDPs. The basic scheme is depicted in Algorithm 1, where the Bellman equation for state values is iteratively used for a predefined number of time steps.

Algorithm 1 Value Iteration

Require: $M = \langle S, A, \gamma, Tr, R \rangle$
  for all $s \in S$ **do**
    $V(s) \leftarrow 0$
  end for
  for all $h = 1, 2, \ldots, H$ **do**
    use Equation 3.38
  end for

- **Line 1-3:** Set the state values of all possible states to zero, which requires to know all possible world states.
- **Line 4-6:** Use Equation 3.38 to update the respective state value, i.e. update the latter based on the current reward and the maximal future state value.

Complexity Analysis 3.3. *A single iteration of value iteration (i.e. one time step h) ends within $O(|S|^2|A|)$, as one has to calculate the current state value based on possible future states. To this end, sparse transition functions considerably reduce the complexity, where relatively few paths among states are possible. In general, value iteration needs more iterations to converge the higher γ is set, as we have to account for future states occurring after numerous transitions.*

Important extensions of tabular representations are *vector-based MDPs*, where states are represented as fixed-length feature vectors, and *factored MDPs* [53] which use PGMs to abstract the state space.

Until now, we assumed that a single agent learns to choose experts or actions. We now discuss the *multiagent* setting for sequential decision-making, where independent, cooperating agents have to take actions.

3.3.3 Multiagent Decision Processes

In *Multiagent Systems* (MAS), multiple agents share control over the actions that are taken. This generates several advantages, namely robustness of the system due to multiple potentially

diverse agents, scalability due to the ability to distribute logic, or real-world suitability as constraints from various parties can be modeled.

Learning in MAS settings is challenging because agents must learn to *coordinate* their actions and learn to *communicate* with each other if communication is budgeted. Based on prior work in multiagent decision-making for multi-step problems [20, 46], we define a *Multiagent Markov Decision Process* (MMDP), which studies both coordination and communication in MAS. We restrict our definition to the *finite-horizon* case, but the infinite-horizon extension is analogous to single-agent MDPs.

Definition 3.44 (Finite-Horizon Multiagent MDPs)**.** *Based on a finite-horizon MDP, a finite-horizon Multiagent Markov Decision Process (MMDP) is defined as* 8*-tuple* $(\{S_\lambda\}_{\lambda\in\Lambda}, \Lambda, \{A_\lambda\}_{\lambda\in\Lambda}, \{A_\lambda^{COMM}\}_{\lambda\in\Lambda}, R, Tr, H, C_{COMM})$.

- *Let* Λ *be the agent space.*
- *Let* S_λ *be the state space of agent* λ *which determines what part of the global state space* λ *can observe. It might be dependent on taken communication actions.*
- *Let* A_λ *be the environment action space of agent* λ*, where* $a_\lambda \in A_\lambda$ *is similar to an action in a MDP, where the aggregate of all environment actions yields the global action. The choice aggregation function is dependent on the respective MMDP – an example would be the majority vote.*
- *Let* A_λ^{COMM} *be the communication action space of agent* λ*, where* $a_\lambda \in A_\lambda^{COMM}$ *is the equivalent of a query for another agent's last environment action.*
- *Let reward function R, transition function Tr and horizon H be defined as for finite-horizon MDPs.*
- *Let* $C_{COMM} \in N^{\geq 0}$ *be the communication budget for each agent.*

The protocol of a MMDP proceeds in episodes $z \in 1, \ldots, Z$ with Z as defined for MDPs, where in each episode a single start state s_z is revealed.

- For each time step $h \in 1, \ldots, H$ with $H \in \mathbb{N}^{>0}$, the agents act based on the current state $s^h_{\lambda,z}$, where $s^1_{\lambda,z} = s_{\lambda,z}$.
  - All agents perceive their parts of the states $s^h_{\lambda,z}$
  - All agents choose their environment action $a^h_{\lambda,z}$.
  - All agents have the chance to choose communication actions $a^{\text{COMM},h}_{\lambda,z}$ given budget C_{COMM}.
  - The joint action a^* is chosen based on an arbitrary strategy (e.g. majority voting after several communication actions).

- Based on the joint action, the agents moves to the next states $Tr(s^{h+1}_{\lambda,z}|s^{h}_{\lambda,z},a^*)$
- All agents receive the reward of the joint action, i.e. $R(s^{h}_{\lambda,z},a^*)$ for agent λ [3].

As all agents have to *cooperatively* find a single *joint action*, the global value functions for a MMDP (i.e. based on the global state s_z and joint action a^*) are defined equally to MDPs (see Definition 3.37 & 3.38). Each agent still has individual state and action spaces, where individual *local* MDPs could be instantiated to find a policy for the global MDP.

The MMDP is defined in general terms, where the communication protocol as well as the reward conditions are left variable. To learn, for example, in MMDPs with *perfect communication* (i.e. all agents have global state information), RL extensions for MAS exist (e.g. Q-learning for MAS [20]) which can be implemented for each agent, respectively. The scenario is referred to as *joint-action learner setting*.

Definition 3.45 (Joint-action learner setting for MMDPs)**.** *The joint-action learner setting studies MAS coordination with perfect communication. Given a MMDP, we set $C_{COMM} = \infty$. Note that $C_{COMM} = |\Lambda - 1|$ might suffice to query all environment actions of all agents, given that environment actions have to be fixed beforehand.*

Definition 3.46 (Independent learner setting for MMDPs)**.** *The independent learner setting studies MAS coordination with no communication. Given a MMDP, we set $C_{COMM} = 0$.*

For *budgeted* communication with resulting imperfect state information, one has to attempt to learn a communication protocol (e.g. [46]).

We, finally, turn towards methods developed for the Semantic Web to set the baseline for situating prior decision-theoretic concepts into Web scenarios.

3.4 Semantic Web

The *Semantic Web* [14] describes the gradual goal of *lifting* the contents of the *World Wide Web* (WWW) with *semantic annotations* to make it *machine-readable*, where each modeled element is an uniquely identified *resource*. It consists of a set of standards and recommendations on how to build a feasible and adequate machine-readable Web architecture. The Semantic Web therefore reduces *content ambiguity* and enables rich queries to be issued against the Web. Central driving factors for the Semantic Web are a *shared model* (or syntax) to structure data as well as *vocabularies* for generating annotations. We first deal with how data is represented in the Semantic Web.

[3] Note that one could implement a different protocol where, for example, all agents receive the reward of their environment action

3.4.1 Resource Description Framework

The *Resource Description Framework* (RDF) [4] defines the basic elements of a graph representation to derive <subject, predicate, object> *triples* and is used as shared syntax to annotate data, where all elements of the graph are represented as *Uniform Resource Identifier* (URI).

Definition 3.47 (Resource Description Framework, triples)**.** *Let U be the set of URIs, where $u \in U$ might be either a subject (or object) or a predicate. Let B be the set of blank nodes, where $b \in B$ can be a subject or an object. Let L be the set of literals where $l \in L$ can only be an object. A RDF triple is thus defined as $< s, p, o > \in U \cup B \times U \times U \cup B \cup L$. A RDF graph then is a finite set of RDF triples $G \subseteq U \cup B \times U \times U \cup B \cup L$.*

To publish, persist and share the graph representation across the Web, different RDF serializations exist. In this work, we use the *Terse RDF Triple Language* (TURTLE) [5] which is a natural and easily understandable language to model RDF graphs. Example 3.2 illustrates TURTLE for a small RDF graph.

Example 3.2 (TURTLE syntax)**.** *We model two simple triples describing that a resource is of class person and has an assigned name.*

```
1 @prefix :       <http://example.org/>.
2 @prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
3 @prefix foaf:   <http://xmlns.com/foaf/0.1/>.
4
5 :Patrick_Philipp    rdf:type    foaf:Person;
6                      foaf:name   "Patrick Philipp".
```

Listing 3.1: TURTLE example.

Here, FOAF is the Friend of a Friend vocabulary.

Next to RDF and FOAF, we now discuss two vocabularies for modeling triples with higher expressivity, namely RDFS and OWL.

3.4.2 RDF Vocabularies

RDF Schema (RDFS) [6] is a RDF vocabulary to construct simple *ontologies*. RDFS enables to model general statements about predicates and classes, and their mutual relationships.

More expressive than RDFS is the *Web Ontology Language* (OWL) [7]. OWL enables to extensively model classes and relationships, and assuring that individuals adhere to *constraints*. In addition to modeling, OWL enables to *reason* about classes and relationships to derive new knowledge.

[4] `https://www.w3.org/RDF/` (*accessed on 05/01/2018*)
[5] `https://www.w3.org/TR/turtle/` (*accessed on 05/01/2018*)
[6] `https://www.w3.org/TR/rdf-schema/` (*accessed on 05/01/2018*)
[7] `https://www.w3.org/TR/owl-features/` (*accessed on 05/01/2018*)

We do not exploit any reasoning mechanisms in this thesis and solely perceive OWL and RDFS as means to model RDF triples. We thus do not specifically require nor exclude their usage, as models need to fit the respective domains and needs. Based on RDF and its TURTLE serialization, we now shortly discuss the related *Notation3* language.

3.4.3 Notation3

TURTLE is a subset of *Notation3* (N3) [8] which extends RDF with *logical constructs*. The `log:implies` predicate [9] is one such example, enabling to model implications of RDF graphs. Example 3.3 illustrates N3 with respect to implications.

Example 3.3 (Implications in N3 syntax)**.** *We model a simple implication for the* `foaf:Person` *example, where a FOAF predicate is mapped to its inverse predicate in the example namespace.*

```
1  @prefix :       <http://example.org/>.
2  @prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
3  @prefix foaf:   <http://xmlns.com/foaf/0.1/>.
4  @prefix log:    <http://www.w3.org/2000/10/swap/log#>.
5
6  {
7  :Patrick_Philipp    rdf:type    foaf:Person;
8                      foaf:made   :Figure1.
9  :Figure1            rdf:type    foaf:Image.
10 }
11 log:implies
12 {
13 :Figure1           :madeBy     :Patrick_Philipp.
14 }
```

Listing 3.2: Example implication in N3 syntax.

Note that => can be used as synonym for `log:implies`.

We limit our discussion to implications, as we do not use other N3 constructs in this thesis. We next discuss how to query RDF triples.

3.4.4 SPARQL

The *SPARQL Protocol and RDF Query Language* (SPARQL) [10] enables to query RDF triples. Its syntax is similar to the *Structured Query Language* (SQL) to query relational data bases. The basic concept to query RDF triples is a *basic graph pattern* (BGP), which extends RDF with *variables*.

[8] `https://www.w3.org/TeamSubmission/n3/` (*accessed on 05/01/2018*)
[9] `http://www.w3.org/2000/10/swap/log` (*accessed on 05/01/2018*)
[10] `https://www.w3.org/TR/rdf-sparql-query/`(*accessed on 05/01/2018*)

Definition 3.48 (Basic Graph Patterns)**.** *Let V be the set of all variables. A triple pattern is, similar to a RDF triple, defined as $t_i := < s,p,o > \in V \cup U \cup B \times V \cup U \times V \cup U \cup B \cup L$. A BGP then is a set of triple patterns, i.e. $BGP := \{t_1 \ldots, t_n\}$.*

Example 3.4 (TURTLE syntax)**.** *Based on prior modeled RDF triples, we now illustrate a simple SPARQL query to find the resource for a person named 'Patrick Philipp'.*

```
1  frame=single]
2  PREFIX     :  <http://example.org/>
3  PREFIX rdf :  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4  PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
5
6  SELECT ?s
7  FORM <http://aifb-ls3-vm2.aifb.kit.edu:8080/test.ttl>
8  WHERE
9  { ?s    rdf:type    foaf:Person ;
10         foaf:name   "Patrick Philipp" .
11 }
```

Listing 3.3: SPARQL example.

While the `SELECT` clause defines the variables which are to be returned, the respective BGPs are used within the `WHERE` clause of the SPARQL query. The `FROM` clause finally defines the RDF graph to which the query can be issued. The latter might be a simple file containing RDF triples or a *triple store*, i.e. a graph-based data store. Throughout the thesis, we generally refer to triple stores or RDF files as structured knowledge base (KB), where the latter might consist of several independent RDF files or triple stores which have to be integrated at run time.

We next discuss *Linked Data*, which describes a special class of structured data of the Semantic Web.

3.4.5 Linked Data

As a subset of the Semantic Web, Linked Data deals with *publishing* and *interlinking* structured data. The *Linked Data Principles* [18] are as follows:

- Use URIs as names for things.
- Use URIs accessible via the *Hypertext Transfer Protocol* (HTTP) to enable people to look up names.
- Provide useful information for URI lookups using standards, such as RDF and SPARQL.
- Include links to other URIs to discover more things.

The principles essentially describe how to structure Web data, such that the latter becomes useful for users as well as *Web agents* (e.g. automated Web services). This is achieved by using URIs to describe individual resources, which can be easily accessed, return structured data describing the resource and link to related resources.

Based on the introduced concepts for the Semantic Web, we now turn towards *Web services* and how they can be lifted to structured representations.

3.4.6 Services in the Semantic Web

A *Web Service* encapsulates *task-specific* behavior, where an input of arbitrary complexity is processed to generate an output. The service is then deployed on the Web such that it can be universally accessed. We restrict our perception of Web Services to *information services* [121] and subsequently define the latter (see Definition 3.49). Throughout this thesis, we perceive an information service as the equivalent of a solution function for a step (i.e. an expert) in a multi-step task, exposed as Web service.

Definition 3.49 (Information service)**.** *An information service is a black-box with set of input parameters, which generates a hypothesis based on a given set of input parameter bindings. Input parameters refer to the needed inputs for the information service which are defined in an arbitrary service description. Information services thus return dynamically derived hypotheses which are calculated during the service call. Executing (referred to as querying) an information service must not change any state of any entity, i.e. no task instances or data points are being modified. Information services thus do not have any side effects.*

With the rise of the Semantic Web, the idea to enrich available description languages and execution frameworks with semantic concepts came up naturally. Numerous approaches to Semantic Service architectures and -description languages exist (see Section 4.3.1 for an overview and a discussion). We focus on an instance of Semantic Web Services [121] called *Linked APIs* (LAPIs) [121, 122], where API refers to *Application Programming Interface*, corresponding to the access to a Web Service.

Definition 3.50 (Linked API)**.** *A LAPI is an information service for which the following conditions must hold:*

- *SPARQL graph patterns (i.e. BGPs) are used to describe inputs and outputs of the service.*
- *Use RDF as communication means by RESTful content negotiation.*
- *The output should explicitly provide links to the inputs.*

LAPIs enable to encapsulate complex functionalities and make the latter easily accessible by exposing *preconditions* and *postconditions* as Linked Data. Here, conditions are modeled as BGPs, where the same variables should be used for preconditions and postconditions to

express the relations between inputs and generated outputs. RDF communication is enabled via *Representational State Transfer* (REST) [42], promoting the use of HTTP methods for services on the Web. HTTP `GET` requests can then be directly used to retrieve structured content provided by the respective LAPI or its description. *Content negotiation*, here, enables to request and link to specific RDF serializations. If LAPIs require to send structured content in order to correctly call them, HTTP `POST` or `PUT` requests are used, where RDF graphs can be sent in a specified syntax.

Chapter 4

Related Work

In **this chapter**, we discuss the related work for the learning- as well as the automation problem for multi-step tasks with expert advice. We start by discussing related work for the learning problem and begin with multi-step tasks in *Single-Agent Systems* (SAS) in Section 4.1, where we further distinguish between learning and service-related works. We then deal with works for multi-step tasks in MAS (Section 4.2) and distinguish between MAS approaches for SAS tasks and MAS approaches for MAS tasks. We finally discuss related works for the automation problem (Section 4.3), where we align our work with regards to available service descriptions, decision-theoretic frameworks and workflow systems.

4.1 Multi-Step Tasks in Single-Agent Systems

In Part II of this thesis, we deal with the learning problem and start by proposing the EP framework as well as two RL algorithms to learn weights for experts (Chapter 5 & 7). We thus first situate EPs into well-established *decision-theoretic* frameworks of SAS and compare our approaches to related SAS techniques. We then focus on service selection and ranking techniques which enable to learn expert service weights.

4.1.1 Learning- & Decision-Theoretic Approaches

Decision-Making with Multi-Step Expert Advice maps to a SAS, where one agent controls all actions (i.e. experts), and is related to several decision-theoretic frameworks. (i) *Prediction with expert advice* [27] studies one-step expert advice, where we are able to observe the rewards of all queried experts. However, since there is a budget for querying experts we need to take decisions inbetween steps. There are overlaps with (ii) *budgeted expert advice* [3], where not all experts can be queried and only a subset of expert hypotheses and rewards are observed, and (iii) contextual bandits [6], where all experts are executed, but only the chosen action is rewarded. In addition, as we deal with decision processes over multiple steps, (iv) *Contextual Decision Processes* (CDPs) [77] are strongly related to EPs, as they approach the multi-step contextual bandit problem. While a single action needs to be chosen in both EPs and CDPs, one receives rewards for all available actions in EPs. EPs still deal with a partial label problem, as not all experts can be executed for each step.

Knowledge-based trust [35] extends knowledge fusion approaches [34], which combine the outputs of multiple *triple extractors* and is thus an instance of (i), where a combination function over experts is learned based on heterogeneous data. While we do not deal with problem settings where input data might be flawed, our approach could be integrated into knowledge-based trust by providing accurate confidence measures for available triple extractors.

To determine the accuracy of experts in an i.i.d. setting, [105, 106] use pairwise error independence measures between binary classifiers *without* requiring labeled data (i.e. they use unsupervised learning) and also fall under (i). We do not solely rely on unlabeled data, but leverage collective inference to integrate labeled and unlabeled data. In addition, experts are allowed to perform worse than average in EPs, which is not possible for the unsupervised case.

The *algorithm selection* problem has widely been studied for *combinatorial search problems* [75] or within *automatic machine learning* (AutoML) [41, 128], and was originally stated by [111]. It deals with weighting a set of available algorithms (similar to experts) based on available training sets, where to goal is not to execute all available algorithm configurations while learning. The algorithm selection problem thus is a special case of (ii) and (iii), as high-dimensional state spaces are dealt with and a budget is imposed during the learning phase. AutoML entails automating *feature selection*, *algorithm selection* and *hyperparameter optimization* and thus falls under (iv), as (in addition to algorithm selection) one has to deal with multiple steps. While, for example, used *Bayesian optimization techniques* share numerous overlaps with strategies to balance exploitation and exploration needed for EPs, we focus on guiding the decision process for single states, where a single state often entails to take a number of decisions (referred to as *decision candidates*) – in NLP, for example, a single state or data point is a piece of text, where individual decisions might have to taken for each token. More specifically, the set of available hyperparameters in AutoML is assumed to be known in advance, which is not the case for decision candidates in EPs.

While *ensemble learning* deals with learning functions over individual models, *meta-learning* [22] aims to optimize the application of experts to new data sets based on performance histories and *meta-features* describing the data sets. Both types of approaches can be seen as batch versions of (i). Our work, however, focuses on selecting experts for specific data points, which requires finer-grained expert weight functions and novel feature representations.

We finally distinguish our work with respect to *state representations* used for modeling the decision process. *Tabular representations* are the most basic, quickly yielding large state spaces. To this end, *Relational Markov Decision Processes* (RMDPs) [38, 65] enable to *factorize* state and action spaces by allowing relational structures. The resulting representations are compact and can be used with relational extensions of prominent RL algorithms – broadly referred to as *Relational Reinforcement Learning* (RRL) (e.g. [37, 38, 65]) – or *Planning* algorithms, such as a relational lifting of *value iteration* [66]. In EPs, we also exploit relational state- and action spaces in terms of *meta dependencies*, which are pairwise expert features for single decision candidates, but our goals for the learning problem significantly differ from RRL. We exploit relational measures between experts and data points, where the goal state

is not explicitly known a priori, i.e. it is not clear which expert or combination of experts suggests the correct action for a given state (and its decision candidates).

4.1.2 Service Approaches

To *select* and *rank* appropriate services, different approaches exist which all fall under (i) of decision-theoretic frameworks. A straightforward way to use *Quality-of-Service* (QoS) parameters for service ranking is by *Simple Additive Weighting* [5, 145], where the preference-weighted combination of QoS parameters is used. To alleviate *sparse representations* of users, link prediction based on user similarity matrices is an effective way to select appropriate services [129]. In a similar vein, pairwise service measures can be used to retrieve rankings of services [147, 148], which is also studied for limited information settings [57]. To rank services without pairwise measures, *clustering* techniques such as *k-Means* can be applied to QoS parameters [88]. The latter also enable the use of decision trees, where historic logs are modeled as rules [26]. To select and combine analytics services based on available data sets, a simple approach is to simplify multi-step problems to *service bundles* and weight the latter by a static accuracy estimate calculated via cross-validation [113].

All prior methods share similarities to our approach, where services are individually assessed by – among others – pairwise service measures. The latter are, in our case, dependent on neighborhoods of specific data points, as we only optimize for QoS parameter *accuracy*. All prior methods, however, do not learn weights for individual decision candidates and do not deal with multi-step learning problems.

4.2 Multi-Step Tasks in Multiagent Systems

In Chapter 6 & 7, we argue that the single-agent assumption does not specifically deal with resolving expert correlation and might not be adequate for numerous real-world systems. We now additionally deal with advances in MAS which are related to multi-step expert advice. We first compare our approach to MAS for single-agent tasks and subsequently turn to MAS for Multiagent tasks .

4.2.1 Multiagent Systems for Single-Agent Tasks

For multi-step problems, *centralized* multiagent learning has recently been proposed [82, 134] within the *Separations of Concerns* (SoC) framework. By having specialized agents learn different parts of a complex reward function, the goal is to speed up the learning process. The predictions of the individual specialized agents depend on the RL scenario (i.e. model-based, value-based or policy search) and are eventually integrated by a combination function of a centralized *meta agent*. The framework generalizes – among others – Hierarchical Reinforcement Learning (HRL) [9, 33], where agents are organized in hierarchical structures such that

agents higher in the hierarchy select agents beneath them. In MEPs, we do not assume a *centralized* meta agent to coordinate all agents, which requires agents to *actively coordinate* their actions. We additionally approach multiagent coordination in a designated decision process, where available agents can adapt their actions several times until the joint action is fixed.

4.2.2 Multiagent Systems for Multiagent Tasks

A central aspect for cooperative MAS is the degree of information available to individual agents. In the *independent learner setting* (see Definition 3.46), agents attempt to coordinate without the knowledge of other actions and are not allowed to communicate [61, 62, 90]. In the *joint-action learner setting* ((see Definition 3.45)), actions have perfect knowledge about all taken actions [25, 29]. Here, the joint action is rewarded for all agents, but penalization might occur if no consensus was reached. For the joint-action learner setting, managing agent communication is a central problem, where communication can be modeled as separate action space (e.g. [46]). In this work, we assume the joint-action learner setting and assume communication is either assumed to be perfect (e.g. due to a centralized controller who only distributes information) or has already taken place in an optimized manner to yield perfect information.

Approaches to *decentralized* Multiagent multi-step coordination problems [20, 83] partially extend MDPs by partitioning the action space according to available agents, resulting in MMDPs (see Definition 3.44). Distributed variants of Q-Learning are often used to learn both independent as well as joint-action policies [25, 83]. In general, MAS involving autonomous coordination among cooperating agents have been studied for different domains and tasks, such as work load allocation [74] or goal searching robots [146]. In a similar vein, *Multiagent Relational Reinforcement Learning* [30, 107] extends RRL to the multiagent case and thus enables speeding up the learning process by compact relational state and action spaces. An essential difference in MEPs is the possibility of *active coordination*, which is an additional decision process to enable agents to react to the current weighted majority vote. In order not to lose expressiveness, active coordination assumes *continuous action spaces* which enable agents to adapt the weights for their experts. Here, policy-based RL techniques often work better than value- or model-based ones, as convergence is usually faster.

Swarm intelligence approaches for *decentralized* Multiagent coordination [146] focus on aligning single-agent strategies with the current weighted majority vote. The goal is to exploit high local agent weights to influence *non-informed* cooperating agents (i.e. agents with low local weights). *Active coordination*, as pursued in MEPs, is directly inspired by swarm intelligence, as agents either persist on their experts' strategies or follow the weighted majority vote, thus aligning their actions. We, however, extend techniques proposed in [146] to take into account the current context (i.e. available state information and pairwise agent behavior) as well as to learn policies in a RL setting.

From a Web service perspective, MAS involving autonomous coordination among cooperating services [1, 54, 116] work towards self-organization, where Web architectures, agent

types and communication protocols are developed. The focus, however, is put on system- and composition aspects. The central idea of our work for the learning problem is to develop supervised methods for expert weight estimation, which enable agents to self-organize.

4.3 Multi-Step Tasks on the Web

Part III of this thesis deals with the Web automation problem for multi-step expert advice. We divide related works into (i) *description languages* for Web services, (ii) *frameworks* for decision-making and (iii) *workflow systems*.

4.3.1 Semantic Web Service Description Languages

While there is a large body of works covering *non-semantic Web service descriptions* (e.g. the Web Service Description Language (WSDL) [28]; see [110] for an overview), we restrict the comparison to semantic descriptions, as the latter depict an important step towards machine-readability and are the closest to our work (see [135] for a complete overview).

Semantic Annotations for Web Service Description Language (SAWSDL) [72] is a semantic extension to WSDL (based on WSDL-S [1], an early semantic extension for WSDL), which however neglects to describe functional relationships between inputs and outputs. To this end, the *Web Service Modeling Ontology* (WSMO) [112] and OWL-S [89] are based on OWL, enabling to model more expressive descriptions. However, both require complex models even for simpler services and – similar to WSMO-Lite [137] which extends SAWSDL with conditions and effects – suffer from inexact input/output mappings as pointed out by [123]. We thus rely on LAPIs, where preconditions and postconditions are modeled as BGPs, and on the *Minimal Service Model* (MSM) description language [73] [2], a lightweight ontology with base elements for structured service descriptions.

Closely connected to service languages is the *Open Provenance Model for Workflows* (OPMW) [50] ontology, which is based on the *Open Provenance Model* (OPM) [3] and provides essential elements to describe workflow-related provenance metadata. As OPMW might be used to document the outcomes of LAPI executions, it does not compete with our work but potentially extends our generated provenance metadata.

4.3.2 Decision-Making Frameworks & Applications

The *Stanford Research Institute Problem Solver* (STRIPS) [43] comprises all functionalities for *problem solving*, i.e. a formal language to describe actions and a planner to choose actions given states. STRIPS requires actions to consist of an add list, a delete list and a precondition.

[1] `https://www.w3.org/Submission/WSDL-S/` (*accessed on 05/01/2018*)

[2] `http://iserve.kmi.open.ac.uk/ns/msm/msm-2014-09-03.html` (*accessed on 05/01/2018*)

[3] `http://open-biomed.sourceforge.net/opmv/ns.html` (*accessed on 05/01/2018*)

By adding or deleting state conditions, one can express impacts of actions. STRIPS is the baseline for multiple decision-making methods, such as *Problem-Solving Methods* (PSMs) or MDPs, which we now integrate into our work.

PSMs [12, 108] are closely connected to STRIPS as well as to our work. PSMs describe highly parameterized algorithms which, in conjunction with expert knowledge, are able to solve real-world tasks. Similar to our work, PSMs are described in terms of functional specifications, requirements and operational specifications (i.e. the preconditions and postconditions). To this end, the *Unified Problem-solving Method Language* (UPML) [40] enables to graphically model and semantically describe PSMs, which supports both reusing and adapting available PSMs. While several PSMs can be easily composed and combined to solve multi-step tasks, the resulting learning problems are not dealt with.

Usually, *Planning* (see [55] for an overview) involves finding a policy which suggests the optimal action for each possible world state. For tasks involving uncertainties with respect to action outcomes, stochastic STRIPS enables to assign probabilities to actions. To this end, MDPs deal with learning the parameters of stochastic STRIPS tasks, which we use as decision-theoretic baseline for our work.

Planning techniques have been applied to Semantic Web services before (see [70] for an extensive overview). Approaches to enable dynamic orchestration of Semantic Web services comprise – among others – Hierarchical Task Networks (HTN) [69, 118] or OWL reasoning [117]. MDPs have been applied to compose *non-semantic* Web services [36], but the reward function was solely dependent on static *Service Level Agreements* (SLAs). There are, to the best of our knowledge, no works for Semantic Web services and MDPs to deal with dynamic rewards based on task instances, as is the case for multi-step expert advice. We thus introduce SEPs which extend EPs with semantic annotations and allow to further distinguish among abstract and grounded tasks.

To this end, it is important to point out that SEPs share several similarities to an extension of RMDPs to multi-step expert advice on the Semantic Web. Here, Logical Markov Decision Programs (LOMDPs) [65] assume manually modeled state- and action spaces, which might consist of numerous abstract or grounded relations. Logical rules are then used to define (stochastic-) transitions and assigned rewards. In SEPs, semantic experts can be interpreted as to providing similar STRIPS-like rules, consisting of RDF triples. We show that, equal to LOMDPs, SEPs can be reduced to discrete MDPs but do not exploit relations to speed up Learning or Planning. Using relational MDPs, such as LOMDPs, could thus improve planning in SEPs if one can guarantee a direct mapping to the former. We discuss this challenge in detail in terms of future works of the thesis (see Chapter 11).

4.3.3 Workflow Systems

There is ongoing research in *workflow systems* which enable to describe and execute experts (and expert services) of different kinds. *Pegasus* [31] is a workflow management system able to map abstract pipelines for simulation data analysis to distributed computing environments.

In a similar vein, *FireWorks* [59] focuses on enabling workflows to be executed on supercomputers. *AKSALON* [39] enables to define grid workflow applications via a graphical user interface or directly via XML, thereby easing the use of grids. *dispel4py* [44] is a Python-based framework to enable workflows for data streams, especially for distributed computing. *Makeflow* [2] enables to define workflows based on *Unix Make*, which are suitable for data-intensive distributed computing applications. Finally, Web service workflows are approached by the *LanguageGrid* project [4], where diverse NLP experts are integrated from various institutions. All mentioned workflow systems are sophisticated approaches to compose services, but do not focus on decision-making for single data points (i.e. decision candidates) or exploit semantic annotations.

Taverna [97], on the other hand, is a scientific workflow system supporting process prototyping by creating generic service interfaces and thus easing the integration of new components. Lightweight semantic descriptions modeled in RDF are used to better capture the view of the scientists. Taverna is also able to integrate data from distributed sources and automate the workflow creation process for users. Taverna was successfully used in the *PANACEA* project [5] for NLP tasks. Our work is strongly related to Taverna in terms of using RDF to describe (NLP-) services. Service selection then equals constructing SPARQL queries, where our architecture diverges from Taverna with regards to using supervised learning to optimize the selection for single data points. Similarly, *Galaxy* [19] enables to conduct sustainable and reproducible workflow experiments based on well-defined services and been successfully used for the *LAPPS Grid project* [6], where numerous NLP services have been implemented, semantically described using JSON-LD [7] and published on the Web. Our work shares many overlaps with both Taverna and Galaxy, but differs in the use of *semantic meta components* to combine and plan under uncertainty. Our architecture could greatly benefit from reusing semantically annotated NLP services of prior projects.

The work centered around *semantic workflows* [52] aims to enable the automatic composition of components in large-scale distributed environments. Generic semantic descriptions support combining experts by Ensemble Learning, but require to specifiy conditions and constraints. The framework also automatically matches components and data sources based on user requests and uses prior mentioned OPMW ontology for provenance generation. We also employ Planning techniques but build our work on MDPs, which are based on environmental feedback and enable to approach our learning and automation problems. To this end, we also deal with optimization for multi step tasks, which is not covered by semantic workflows. We rely on training sets to execute and test experts, which might be annotated by OPMW.

In [144], abstract workflows are created as domain models, formalized using OWL, to enable dynamic instantiation of real processes. The components can be reused in another context or process, and their abstract representations can be shared across the Web through

[4] `http://langrid.org/` (*accessed on 05/01/2018*)
[5] `http://www.panacea-lr.eu/` (*accessed on 05/01/2018*)
[6] `http://www.lappsgrid.org/` (*accessed on 05/01/2018*)
[7] `http://json-ld.org/` (*accessed on 05/01/2018*)

OWL classes. The authors rely on triple patterns to select appropriate annotated components for steps, but only exploit `owl:subClassOf` relationships to enable generalization. The approach thus does not deal with uncertainties. With SEPs, we are able to use lightweight semantic annotations (in contrast to OWL), while also generalizing the performance of experts using RL, assuming available training sets.

Finally, automatic composition of *analytical workflows* is studied in [16]. The system essentially uses a planner, a leaner and a large KB to solve tasks. A large amount of potential workflows are taken into account to answer a user specified query with the optimal choice. The decision process comprises complex Learning and Planning approaches, and entails exploring large feature spaces. Lastly, atomic actions are lifted with semantic annotations to better adapt to user queries. The proposed system, similar to our work, assesses experts in terms of labeled data and uses a planning engine to compose experts given a user query. The system, however, does not enable flexibility with respect to available learners and planners. In addition, weights for experts are not adapted for individual decision candidates, as the system relies on global ensemble functions. Finally, central properties for optimizing multi-step tasks are ignored, such as the impact of current hypotheses on future steps.

Part II

Learning

This part of the thesis deals with the *learning problem*, where we deal with research questions about choosing and combining experts for multi-step tasks in order to maximize the number of correct solutions. We begin by defining the central framework of this thesis – EPs – and subsequently present two RL approaches for learning expert weights (Chapter 5). We then present an extension of EPs to the Multiagent setting – i.e. MEPs – where we present methods to coordinate expert weights in order to reduce expert correlation (Chapter 6). We evaluate our approaches for both EPs and MEPs in Chapter 7 based on the NERD scenario.

Chapter 5

Learning with Expert Processes

This chapter proposes the EP framework as solution to Research Question 1, dealing with an appropriate decision-theoretic model to capture challenges for multi-step tasks with expert advice. We respectively state in Hypothesis 1 that we need to enable learning global expert weights for individual data points such that combined and weighted expert hypotheses maximize the overall performance and that the exploration-exploitation problem is central. To this end, we also approach Research Question 2 by introducing two methods for learning expert weights in EPs. In Hypothesis 2, we target relational feature spaces for expert weights, which can be used for RL algorithms. To confirm Hypothesis 1 & 2, we empirically evaluate the learning methods (and thereby the EP framework) in Chapter 7.
Our reference for this chapter is [100].

5.1 Introduction

An increasing amount of experts become available via service provider platforms such as Algorithmia, enabling to reuse and compose them for complex real-world applications, such as multi-step tasks. The availability of exchangeable experts for a single step of a multi-step task highlights its difficulty, as individual experts often generate wrong hypotheses for at least a small subset of all available data points. Here, the average number of mistakes often significantly differs for different data distributions. To this end, one needs to weight exchangeable experts for individual data points in order to (i) choose a subset of available experts to deal with expert budgets and to (ii) find the correct hypothesis of the queried experts which supports to solve the multi-step task.

In addition to dealing with exchangeable experts, we also have to take into account how experts perform in a *sequence*, as slight changes in inputs often have significant influence on the correctness of an expert's output. Even more so, experts in each but the last step are allowed to make small errors as long as the eventual solution is correct, which needs to be taken into account when learning their weights.

Well-established *decision-theoretic* frameworks partially deal with these problems, including prediction with expert advice, budgeted expert advice, contextual bandits and CDPs (see Section 4.1.1 for a detailed discussion). Still, none of the frameworks cover the problem of *Decision-Making with Multi-Step Expert Advice* at once.

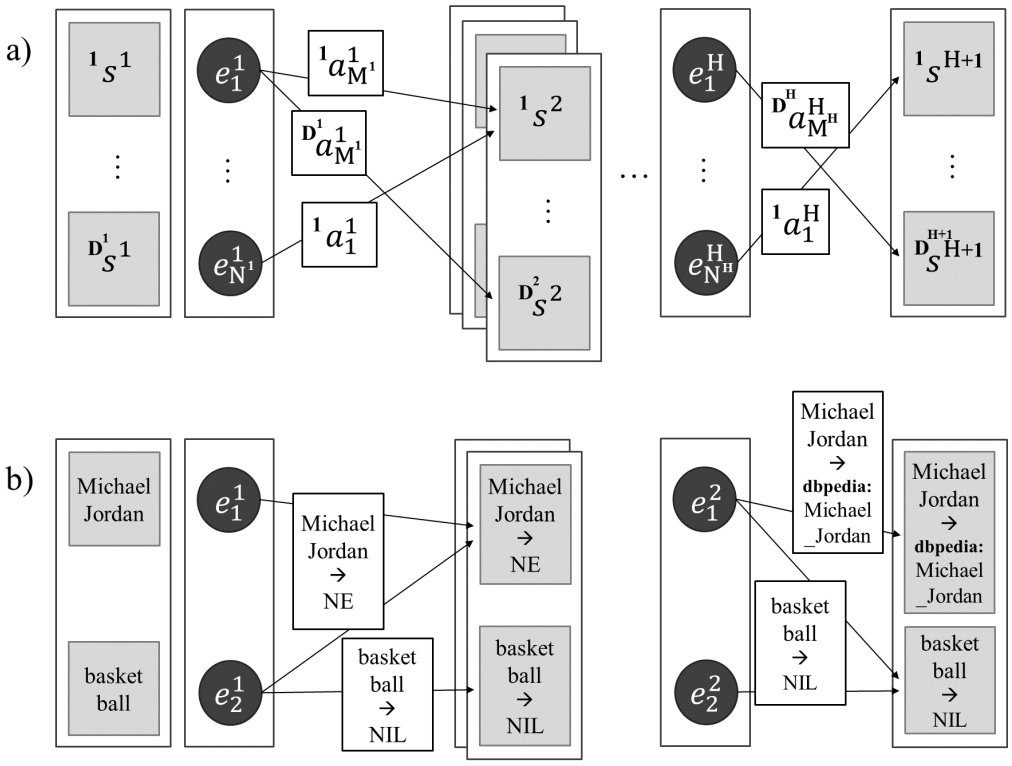

Figure 5.1: Multi-Step Expert Advice in an Expert Process with a) the formal process and b) a NLP example for NERD.

In this chapter, we define the EP to bridge this gap. EPs enable to learn fine-grained expert weights by considering the impact of current decisions on future experts, while not being overly restrictive in rewarding outcomes. Figure 5.1 a) illustrates EPs in terms of their schema. EPs extend MDPs (see Definition 3.34), which model the general sequential decision-making setting. In an EP, world states (variable s^h) belong to a single step and are partitioned into *decision candidates* (variable ${}^d s^h$), which are distinct parts of the world state the agent has to interpret. Actions (variable ${}^d a^h$) for decision candidates can only be generated by available experts. The agent has to query a subset of experts (variable e_i^h) and then choose among the hypotheses of experts, after which it receives a reward and perceives the resulting state (variable s^{h+1}).

Based on the schema of an EP, we now discuss the illustrated NLP application to step NER (see Figure 5.1 b)).

Example 5.1 (EP example for step NER)**.** *Step 1 for multi-step task NERD is NER, where an agent perceives states comprising text and has to find available named entities – an exemplary state might be* $s_1 =$ `"Michael Jordan plays basketball."`*. To find possible action to choose from, an agent needs to query available NER experts for (parts of) the state.*

Let us assume two generic NER experts e_1, e_2 *which are both queried (ignoring a possibly predefined budget which limits the query to a single expert) and produce* $e_1(s_1) =$ `"<e>Michael Jordan</e> plays basketball."`*, i.e.* e_1 *suggests action* $a_1 =$ *{(*`"Michael Jordan", <e>Michael Jordan</e>`*)}, and* $e_2(s_1) =$ `"<e>Michael Jordan</e> plays <e>basketball</e>."`*, i.e.* e_2 *suggests action* $a_2 =$ *{(*`"Michael Jordan", <e>Michael Jordan</e>`*),(*`"basketball", <e>basketball</e>`*)}.*

As NER is not the last step of NERD and the agent still needs to query NED experts to find possible mappings from named entities to resources in a given KB, the agent has to keep a probability distribution over all actions and resulting states, and *needs to choose a single action for which it gets a reward. Given the budget for experts, the agent then queries NED experts based on all possible (weighted) states, as the agent has to learn how to best query experts.*

To this end, we construct relational feature spaces, referred to as *meta dependencies*, which quantify the predictive performance of experts for decision candidates. Relational measures are promising to overcome generalization problems for heterogeneous data distributions, for example available for text or images. These problems partially stem from assuming models for data to be i.i.d., limiting generalization for individual data points. We use meta dependencies as state representation for an Online- and a Batch RL approach, where we extend the EWH algorithm for efficient expert weight updates as well as use *Probabilistic Soft Logic* (PSL) [24, 67] for collective inference. Combining relational features and supervised learning is related to *stacked generalization* (see Definition 3.26), where the central difference is using relations among learners other than using their outputs directly. This is important, as the set of possible decision candidates is intractable to use as feature representation for certain steps, e.g. NED.

5.1.1 Challenges

The central challenges approached in this chapter are summarized subsequently and refer to the challenges defined in Sec 1.2.

- As querying experts often entails costs, one usually does not want to execute a large number of exchangeable experts. However, choosing only a single expert might result in wrong or unusable hypotheses for the current as well as later steps. Even more so, when learning to assess experts with supervised learning, querying a single expert (or a small subset of available experts) might result in missing feedback (*Challenge 1*).

- Local optimization for individual steps often fails when experts perform better with inexact inputs (*Challenge 2*).
- Generating training samples for learning expert weights requires experts to be executed on training sets. For multi-step tasks, random execution of experts is potentially harmful, as experts of later steps are executed in an uncontrolled manner (*Challenge 3*).
- When labeled data sets are small, it is difficult to engineer or learn well-working feature representations for unstructured data (such as text or images). (*Challenge 4*).

5.1.2 Contributions

Based on these challenges, we now summarize our contributions.

- We provide a formalization for EPs, aligning them with prominent decision-theoretic frameworks which allow to deal with multi-step problems, where feedback is only partially available. EPs enable to solve multi-step tasks with expert advice with RL algorithms (*Contribution 1*).
- We define and use meta dependencies as feature representation for learning, which enable to integrate other, potentially high-dimensional feature representation as well as exploiting pairwise expert behaviors (*Contribution 2.1*).
- We present two RL approaches, namely Online RL based on the EWH algorithm and Batch RL using PSL. The EWH algorithm scales well, even when a large number of experts are available. PSL, on the other hand, is able to exploit meta dependencies with respect to pool-based prediction as well as collective inference (*Contribution 2.2*).
- We instantiate our approaches for task NERD and empirically evaluate it against monolithic real-world systems as well as available expert combination approaches in Chapter 7.

The chapter proceeds as follows: We first formalize EPs (Section 5.2) and introduce meta dependencies for experts and data points (Section 5.3). We then present an Online RL and a Batch RL approach to solve EPs in Section 5.4 & 5.5. We finally summarize this chapter in Section 5.7.

5.2 Problem Formalization

Given a finite layered multi-step task (see Definition 3.2 & 3.4), we now define a decision process given the introduced challenges for multi-step expert advice. More specifically, we now introduce EPs which extend MDPs with the availability of decision candidates, experts as well as less restrictive budgets.

Definition 5.1 (Expert Processes)**.** *An EP extends a MDP and is a* 8*-tuple* $(S, E, A, \gamma, Tr, R, H, C_{EXP})$.

- *Let S be the set of all states, where* $S = S^1 \cup S^2 \cup \ldots \cup S^{H+1}$ *consists of* $H+1$ *disjoint, heterogeneous state spaces. A state* s^h *is thus uniquely assigned to a step and can be represented in any form, e.g. as a set, sequence, vector, graph or otherwise. Each state* s^h *has a set of* D^h *decision candidates* ${}^1s^h, \ldots, {}^{\mathrm{D}^h}s^h$*, i.e.* D^h *decisions need to be taken for* s^h.

- *Let* Γ *is the tart state distribution, i.e.* $\Gamma \in \Delta(S^1)$.

- *Let A be the set of all possible actions.*

- *Let E be the set of all experts, where an expert* $e \in E^h$ *is a function* $e^h : S^h \to AR$ *with* $AR \subseteq S^h \times S^{h+1}$ *an action relation for mapping decision candidates between two state spaces. Possible actions* ${}^d a^h_j \in {}^d A^h$ *for* ${}^d s^h$ *are defined as follows*

$$ {}^d A^h := \{ {}^d s^{h+1} | \exists e^h \in E : (s^h, s^{h+1}) \in e^h(s^h) \} \tag{5.1} $$

  With slight abuse of notation, we use the expert function to return the suggested action of expert e^h *for state* s^h*, i.e.* $e^h : S^h \to A$*, as it eases demonstration of our proposed methods as well as alignment with works in other fields, such as prediction with expert advice, contextual bandits or MDPs.*

- *Let* $w_e : S \to \mathbb{R}$ *the expert weight function, which quantifies the expected quality of the expert's action for a state.*

- *Let* $\gamma \in [0,1]$ *be the discount factor hyperparameter.*

- *Let* $Tr : S^h \times A^h \to S^{h+1}$ *be the deterministic transition function. As EPs are acyclic (based on the definition of a layered finite multi-step task),* $Tr(s^h, a^h)$ *will always result in* s^{h+1}.

- *Let* $R : S^h \times A^h \to [0,1]^{\mathrm{D}^h}$ *be the reward function. The reward in an episode,* r^h_z*, is a vector of local feedback received after choosing action* a^h *in state* s^h*, i.e. the vector contains feedback for each decision candidate.*

- *Let* $H \in \mathbb{N}_+$ *be the number of steps.*

- *Let* $C^h_{EXP} \in \mathbb{N}_+$ *be the budget per step. It confines the number of state-expert pairs* (e^h, s^h) *one is allowed to query in step h.*

- *Let* $\pi : S \to A$ *be the policy, which maps states to actions.*

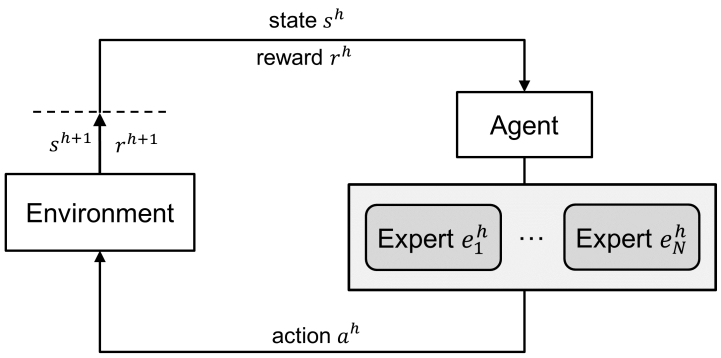

Figure 5.2: Schema of EPs in the MDP framework.

Figure 5.2 illustrates EPs in the RL framework, extending Figure 5.1 with a feedback loop. It illustrates a single step of an EP, where the agent observes the current state s^h, has to choose among experts $e_1^h, \ldots, e_N^h$ in order to find action a^h, and finally observes the new state s^{h+1} and receives reward r^{h+1}. Note that respective states consist of decision candidates ${}^d s^h$, which are omitted in the figure.

The *EP protocol* of an EP proceeds in episodes $z \in Z$, where for each episode z the process goes through all steps. In each step h for $h = 1, \ldots, \mathrm{H}$:

- We are given $\hat{S}^h$ (dependent on episode z).
- Build state-expert pair distribution $P_{\text{EXPERT}}(s^h, e^h)$.
- Choose and query C^h_{EXP} state-expert pairs (s^h, e^h) and retrieve $a^h_1, \ldots, a^h_{\text{M}}$.
- Build state-action distribution $P_{\text{ACTION}}(s^h, a^h)$ based on $\hat{S}^h$ and $\hat{A}^h$, with $\hat{A}^h = \{a^h | {}^d a^h \in {}^d A^h\}$, containing all possible actions based on $a^h_1, \ldots, a^h_{\text{M}}$.
- Choose (s^h, a^h) and *receive* $R(s^h, a^h)$, but *observe* all residual rewards, i.e $\forall(\hat{s}^h, \hat{a}^h)$: $R(\hat{s}^h, \hat{a}^h)$ [1].
- If $h < \mathrm{H}$ then set $\hat{S}^{h+1} = \{s^{h+1} | s^{h+1} \in \hat{A}^h\}$.

We now define the *value functions* for EPs, which express optimal characteristics of states and actions. The state-action value function for a MDP (see Definition 3.38) expresses the expected value of a state and action by considering possible future states and actions. We express this condition via expert weights, such that the state-value function is defined as weighted majority vote of experts.

[1] Feedback might be available in retrospect at $h = \mathrm{H} + 1$.

Definition 5.2 (State-action value function in EPs)**.** *The state-value-function, $Q : S \times A \rightarrow \mathbb{R}$, for taking an action a^h in s^h is defined as aggregation of expert weights, which suggest action a^h.*

$$Q(s^h, a^h) = \sum_{{}^d s \in s^h} \sum_{e \in E^h} w_e({}^d s) * \mathbb{1}_{\{e({}^d s) = {}^d a^h\}} \tag{5.2}$$

The *state value function* is analogously defined. Here, we only aggregate the respective expert weights for the respective state. The intuition is that a state has high value if numerous experts are estimated to perform well.

Definition 5.3 (State value function-, State-value distribution in EPs)**.** *The state value function $V : S \rightarrow \mathbb{R}$ is defined as follows:*

$$V(s^h) = \sum_{{}^d s^h \in s^h} \sum_{e^h \in E} w_{d_{e^h}}({}^d s^h) \tag{5.3}$$

As an expert weight should ideally express the current reward current state *and* for value for future states and actions. This is essential for a multi-step task, as we need to find global optima with respect to task solutions.

Definition 5.4 (Expert weights in EPs)**.** *An expert weight is defined via the local reward of the suggested action and the sum of maximal Q-values of actions for the resulting states.*

$$w_{e^h}({}^d s^h) = R({}^d s^h, e^h(({}^d s^h)) + \gamma \sum_{a^h \in A^h_{{}^d s^h}} Tr(s^{h+1} | s^h, a^h) \max_{a^{h+1} \in A^{h+1}} Q(s^{h+1}, a^{h+1}) \tag{5.4}$$

where $A^h_{{}^d s^h} = \{a^h | a^h \in A^h : e^h({}^d s^h) \in a^h\}$ is the set of all possible actions and, as we deal with finite multi-step tasks for the learning problem, the discount factor is fixed, i.e. $\gamma = 1$.

Given the state-action value function, we define the *state-action distribution* $P_{\text{ACTION}}(s^h, a^h)$, which expresses the probability for an action a to be chosen in state s, based on the state-action value. The *state value distribution* is analogously defined.

Definition 5.5 (Probability distributions in EPs)**.** *The state-action distribution $P_{ACTION} : S \times A \rightarrow [0,1]$ is defined as the normalized state-action value function estimate.*

$$P_{ACTION}(s^h, a^h) = \frac{Q(s^h, a^h)}{\sum_{s' \in \hat{S}^h} \sum_{a' \in A^h} Q(s', a')} \tag{5.5}$$

where $\hat{S}^h$ is the set of possible states.

The state value function $P_{STATE} : S \rightarrow [0,1]$ is defined as the normalized state value function estimate.

$$P_{STATE}(s^h) = \frac{V(s^h)}{\sum_{s' \in \hat{S}^h} V(s')} \tag{5.6}$$

Finally, the distribution over (state,expert) pairs $P_{\text{EXPERT}}(s^h, e^h)$ is constructed based on normalized expert weights for a single decision candidate.

Definition 5.6 (State-expert distribution in EPs)**.** *The state value function $P_{EXPERT} : S \times E \rightarrow [0,1]$ is defined as the normalized expert weight for a state.*

$$P_{EXPERT}(s^h, e^h) = \frac{\sum_{^d s \in s^h} w_{e^h}({}^d s)}{\sum_{s' \in \hat{S}^h} \sum_{^d s' \in s'} \sum_{e' \in E^h} w_{e'}({}^d s')} \tag{5.7}$$

The reward of the decision process R^{CUM} expresses the expected cumulative reward, similar to finite-horizon MDPs (see Sec 3.39) extended with decision candidates.

Definition 5.7 (Cumulative reward for EPs)**.** *The overall reward of an EP $R^{CUM} : S \times A \rightarrow \mathbb{R}$ is defined as:*

$$R^{CUM} = \mathbb{E}[\sum_{h=1}^{\text{H}} \sum_{d=1}^{\text{D}^h} R^h({}^d s^h, {}^d a^h)] \tag{5.8}$$

The goal in an EP is to learn how to act in the given decision process, i.e. finding the optimal policy π^* which maximizes R^{CUM} and, therefore, the number of correct solutions for the respective multi-step task. Learning to balance exploration and exploitation, appears in EPs in terms of learning which experts to query for which states, which can be approached with RL.

It is important to note that we do not assume an EP to be *realizable*. As a consequence, it is possible that none of the available experts returns the correct action for any state configuration. This makes learning more difficult, as one searches for unavailable positive rewards for the available actions.

In this chapter, we are interested in learning weights w_{e^h} for experts, constrained by a budget C^h_{EXP}. This entails continuously or periodically improving policy π while collecting samples (s^h, e^h), as the sample collection process directly influences the quality of the learned weights and vice versa. For learning w_{e^h}, we exploit available state spaces by calculating meta dependencies for experts and decision candidates. We now introduce the concept of meta dependencies, which is our feature representation for learning expert weights.

5.3 Meta Dependencies

To generalize reward signals for expert hypotheses across states, we need to find an appropriate feature representation for the latter. Before introducing meta dependencies, we discuss straightforward feature representations for NLP tasks.

Example 5.2 (Exemplary feature spaces for NLP)**.** *For tasks in NLP, such as NER, a possible representation for paragraphs or sentences would be bag-of-words (BOW), n-grams or k-shingles. The resulting features are represented as high-dimensional vectors, where all expected words, grams or shingles need to be enumerated. While BOW entails to use single words, n-grams retrieve all sequential words on length n and k-shingles follow the same approach on a word character level. Considering that most available training sets are rather small, most words, grams or shingles do not appear often. To this end, it might be suitable to exploit learned vector embeddings for texts, where generalization might be significantly better and vector sizes significantly smaller. However, not all possible tokens or token sequences might be within the dictionary of the respective trained embedding vectors, which yields no available features for some decisions.*

As training sets are usually limited in size and domain experts are costly to hire, large feature spaces are problematic, as the learning process might not converge. We thus introduce meta dependencies, which are relational features based on relational classification (see Sec 3.21). With meta dependencies, we can model low-dimensional feature spaces based on diverse available feature representations (such as vector embeddings), while also modeling pairwise behaviors of experts for single decision candidates.

A *meta dependency* averages a number of expert-dependent performance metrics in a kernel-induced neighborhood. *Kernels* are similarity measures, which are based on available feature representations. Using a variety of such kernels is robust against unavailability of individual feature values (e.g. due to a missing dictionary entry for word embeddings) and reduces the dimensionality.

With meta dependencies, we can define relations among candidate ${}^{d}s^{h}$ and either (i) two experts e_i^h, e_j^h of the same step, (ii) two experts e_i^h, e_j^{h+1} of subsequent steps or (iii) a single expert e^h. While (iii) ignores the presence of other experts, (i) and (ii) are pairwise measures to quantify which experts cooperate well.

More specifically, a meta dependency $MD^{\kappa,l} : E \times S \rightarrow [0,1]$ averages how single- or pairs of experts behave for a *decision candidate* ${}^{d}s_i^h$ by visiting similar decision candidates ${}^{c}s_j^h$ in its sorted neighborhood, which is defined as follows.

$$N^{\kappa,\delta}({}^{d}s_i^h) = \{{}^{c}s_j^h | {}^{c}s_j^h \in S^h : \kappa({}^{d}s_i^h, {}^{c}s_j^h) \geq \delta_\kappa \wedge i \neq j\} \tag{5.9}$$

with minimal similarity $\delta_\kappa \in [0,1]$. The sorted neighborhood is construed based on a domain dependent kernel $\kappa \in K$ with $\kappa : S^h \times S^h \rightarrow [0,1]$.

The expert behavior is defined by *behavioral measures* for single experts $\beta^l_{\text{SINGLE}} : S^h \times E^h \rightarrow [0,1]$ or pairs of experts $\beta^l_{\text{PAIR}} : S^h \times E^h \times E \rightarrow [0,1]$. The resulting meta dependencies are transformed into probabilities by conditioning the decision candidates in the sorted neighborhood for a given behavioral measures. We thus define *conditions* for single experts $\psi_{\text{SINGLE}} : S^h \times E^h \rightarrow [0,1]$ and pairs of experts $\psi_{\text{PAIR}} : S^h \times E^h \times E \rightarrow [0,1]$.

Before further defining meta dependencies for cases (i)-(iii), we describe properties of neighborhood $N^{\kappa,\delta}({}^{d}s_i^h)$. The latter is constructed for individual decision candidates and con-

sists of sufficiently similar decision candidates for which the agent already received rewards. Such neighborhoods vary in size with respect to decision candidates and kernels. To not overestimate the resulting probabilities, we additionally quantify the density of a neighborhood, which we now define in detail.

Definition 5.8 (Relative density, absolute density)**.** *The relative density of the neighborhood is defined as:*

$$\theta^{\kappa,\delta}_{REL}({}^{d}s^{h}_{i}) = \frac{\sum_{{}^{c}s^{h}_{j} \in N^{\kappa,\delta}({}^{d}s^{h}_{i})} \kappa({}^{d}s^{h}_{i}, {}^{c}s^{h}_{j})}{|N^{\kappa,\delta}({}^{d}s^{h}_{i})|} \tag{5.10}$$

The absolute density is defined as:

$$\theta^{\kappa,\delta}({}^{d}s^{h}_{i}) = \max\{\frac{N^{\kappa,\theta}({}^{d}s^{h}_{i})}{\mathrm{M}_{\kappa}}, 1\}\ \theta^{\kappa,\delta}_{REL}({}^{d}s^{h}_{i}) \tag{5.11}$$

where threshold $M_{\kappa} \in \mathbb{N}_{+}$ defines the expected number of decision candidates in $N^{\kappa,\theta}({}^{d}s^{h}_{i})$, i.e. $|N^{\kappa,\theta}({}^{d}s^{h}_{i})| \geq \mathrm{M}_{\kappa}$.

The relative density weights a neighborhood in terms of kernel values and normalizes its estimate by the size of the neighborhood. It, however, overestimates small neighborhoods with highly similar decisions. As a consequence, we need to define a minimal threshold M_{κ} for the neighborhood size such that small neighborhoods are penalized.

We now present meta dependencies for single- as well as pairwise experts and will assume that received rewards are in interval $[0,1]$.

5.3.1 Single Experts

Single expert meta dependencies for $(e^{h}, {}^{d}s^{h})$ quantify the behavior of e^{h} for ${}^{d}s^{h}$ based on similar decision candidates ${}^{c}s^{h}$. Given neighborhood parameters κ, δ, M, we define $MD^{l}_{\text{SINGLE}}(e^{h}_{i}, {}^{d}s^{h})$ with behavior $\beta^{l}_{\text{SINGLE}}$ and condition c^{l}_{SINGLE} as:

$$MD^{\kappa,l}_{\text{SINGLE}}(e^{h}_{i}, {}^{d}s^{h}) = \frac{\sum_{{}^{c}s^{h}_{j} \in N^{\kappa,\delta}({}^{d}s^{h})} \beta^{l}_{\text{SINGLE}}({}^{d}s^{h}, e^{h}_{i}) \psi^{l}_{\text{SINGLE}}({}^{d}s^{h}, e^{h}_{i}) \kappa({}^{d}s^{h}, {}^{c}s^{h}_{j})}{\sum_{{}^{c}s^{h}_{j} \in N^{\kappa,\delta}({}^{d}s^{h})} \psi^{l}_{\text{SINGLE}}({}^{d}s^{h}, e^{h}_{i}) \kappa({}^{d}s^{h}, {}^{c}s^{h}_{j})} \tag{5.12}$$

We define one instantiation of single expert meta dependencies, which is summarized in Table 5.1. The *precision* of a single expert ($l = 1$) is the averaged received reward over all decision candidates available in the neighborhood. The respective condition states that only decision candidates are used for calculation for which the expert was queried.

Due to the available expert budget, we need to further constrain the neighborhoods based on the number of existing executions of an expert. We therefore define the *condition-specific absolute density* of a single expert meta dependency as

l	Name	Behavior β^l	Condition ψ^l
1	Precision	$\beta^1_{\text{SINGLE}}({}^d s, e_i) = R({}^d s, e_i({}^d s))$	$\mathbb{1}_{\{\exists e_i({}^d s)\}}$

Table 5.1: Single-step expert meta dependencies.

$$\theta_l^{\kappa,\delta}({}^d s^h) = \theta^{\kappa,\delta}({}^d s^h) \frac{\sum_{{}^c s^h \in N^{\kappa,\delta}({}^d s^h)} \psi^l_{\text{SINGLE}}({}^d s^h, e_i^h)}{|N^{\kappa,\delta}({}^d s^h)|} \tag{5.13}$$

5.3.2 Pairwise Intra-Step Experts

Pairwise intra-step meta dependencies for $(e_i^h, e_j^h, {}^d s^h)$ with $i \neq j$ quantify the relative behavior between two experts of the same step given a decision candidate, exploiting its neighborhood. Given neighborhood parameters κ, δ, M, we define $MD_{\text{INTRA}}(e_i^h, e_j^h, {}^d s^h)$ with behavior β^l_{PAIR} and condition c^l_{PAIR} as:

$$MD^{\kappa,l}_{\text{INTRA}}(e_i^h, e_j^h, {}^d s^h) = \frac{\sum_{{}^c s_j^h \in N^{\kappa,\delta}({}^d s^h)} \beta^l_{\text{PAIR}}({}^d s^h, e_i^h, e_j^h) \psi^l_{\text{PAIR}}({}^d s^h, e_i^h, e_j^h) \kappa({}^d s^h, {}^c s_j^h)}{\sum_{{}^c s_j^h \in N^{\kappa,\delta}({}^d s^h)} \psi^l_{\text{PAIR}}({}^d s^h, e_i^h, e_j^h) \kappa({}^d s^h, {}^c s_j^h)} \tag{5.14}$$

We define four instantiations of pairwise intra-step expert meta dependencies, which are summarized in Table 5.2. The first meta dependency depicts the *independent error* of two experts ($l = 2$), which measures the difference in received reward conditioned on at least one expert being wrong. We then define the *joint performance* meta dependency ($l = 3$) which averages the received rewards of two experts on the same decision candidate, assuming both experts have been queried for the latter and agree on their action prediction. Meta dependencies for *dominant performance* ($l = 4$) as well as *additive performance* ($l = 5$) both deal with the case that respective experts disagree (and both have been queried) for a decision candidate, where either the reward of the current expert or the one in comparison is assessed.

Similar to the single expert case, we need to constrain the neighborhoods based on missing expert executions. We define the *condition-specific absolute density* of an intra-step experts meta dependency as:

$$\theta_l^{\kappa,\delta}(e_i^h, {}^d s^h) = \theta^{\kappa,\delta}({}^d s^h) \frac{\sum_{{}^c s^h \in N^{\kappa,\delta}({}^d s^h)} \psi^l_{\text{PAIR}}({}^d s^h, e_i^h, e_j^h)}{|N^{\kappa,\delta}({}^d s^h)|} \tag{5.15}$$

5.3.3 Pairwise Inter-Step Experts

Pairwise inter-step meta dependencies for $(e_i^h, e_j^{h+1}, {}^d s^h)$ with $i \neq j$ quantify the impact of e_i^h on the subsequent step. Given neighborhood parameters κ, δ, M, we define $MD^l_{\text{INTER}}(e_i^h, e_j^{h+1}, {}^d s^h)$, with behavior β^l_{PAIR} and condition c^l_{PAIR} as:

l	Name	Behavior β^l	Condition ψ^l
2	Error	$\beta^2_{\text{PAIR}}({}^d s, e_i, e_j) = \lvert R({}^d s, e_i({}^d s)) - R({}^d s, e_j({}^d s))\rvert$	$\psi^2_{\text{PAIR}}({}^d s, e_i, e_j) = \mathbb{1}_{\{R({}^d s, e_i({}^d s)) + R({}^d s, e_j({}^d s)) \leq 1\}}$
3	Joint	$\beta^3_{\text{PAIR}}({}^d s, e_i, e_j) = \frac{R({}^d s, e_i({}^d s)) + R({}^d s, e_j({}^d s))}{2}$	$\psi^3_{\text{PAIR}}({}^d s, e_i, e_j) = \mathbb{1}_{\{e_i({}^d s) = e_j({}^d s)\}}$
4	Dominance	$\beta^4_{\text{PAIR}}({}^d s, e_i, e_j) = R({}^d s, e_i({}^d s))$	$\psi^4_{\text{PAIR}}({}^d s, e_i, e_j) = \mathbb{1}_{\{e_i({}^d s) \neq e_j({}^d s)\}}$
5	Additive	$\beta^5_{\text{PAIR}}({}^d s, e_i, e_j) = R({}^d s, e_j({}^d s))$	$\psi^5_{\text{PAIR}}({}^d s, e_i, e_j) = \mathbb{1}_{\{e_i({}^d s) \neq e_j({}^d s)\}}$

Table 5.2: Intra-step expert meta dependencies.

l	Name	Behavior β^l	Condition ψ^l
6	Future	$\beta^6_{\text{PAIR}}({}^d s^h, e^h_i, e^{h+1}_j) = R({}^d s^h, e^{h+1}_j({}^d s^{h+1}))$	$\psi^6_{\text{PAIR}}({}^d s^h, e^h_i, e^{h+1}_j) = \mathbb{1}_{\{e^h_i({}^d s^h) = {}^d s^{h+1}\}}$
7	Past	$\beta^7_{\text{PAIR}}({}^d s^h, e^h_i, e^{h-1}_j) = R({}^d s^h, e^h_i({}^d s^h))$	$\psi^7_{\text{PAIR}}({}^d s^h, e^h_i, e^{h-1}_j) = \mathbb{1}_{\{e^{h-1}_j({}^d s^{h-1}) = {}^d s^h\}}$

Table 5.3: Inter-step expert meta dependencies.

$$MD^{\kappa,l}_{\text{INTER}}(e^h_i, e_j, {}^d s^h) = \frac{\sum_{{}^c s^h_j \in N^{\kappa,\delta}({}^d s^h)} \beta^l_{\text{PAIR}}({}^d s^h, e^h_i, e^h_j)\, \psi^l_{\text{PAIR}}({}^d s^h, e^h_i, e_j)\, \kappa({}^d s^h, {}^c s^h_j)}{\sum_{{}^c s^h_j \in N^{\kappa,\delta}({}^d s^h)} \psi^l_{\text{PAIR}}({}^d s^h, e^h_i, e_j)\, \kappa({}^d s^h, {}^c s^h_j)} \tag{5.16}$$

Note that we removed the horizon assignments from the expert in comparison, as both $h+1$ and $h-1$ would be possible. We define two instantiations of pairwise inter-step expert meta dependencies which are summarized in Table 5.3. The *future impact* meta dependency ($l = 6$) measures how experts of the subsequent horizon are rewarded if the decision candidate of the current expert is chosen. Similarly, the *past impact* meta dependency ($l = 7$) quantifies how well the current expert is rewarded if the current decision candidate was suggested by the expert in comparison.

Finally, we again account for missing expert executions by defining the *condition-specific absolute density* of an inter-step experts meta dependency as:

$$\theta^{\kappa,\delta}_l(e^h_i, {}^d s^h) = \theta^{\kappa,\delta}({}^d s^h) \frac{\sum_{{}^c s^h \in N^{\kappa,\delta}({}^d s^h)} \psi^l_{\text{PAIR}}({}^d s^h, e^h_i, e_j)}{\lvert N^{\kappa,\delta}({}^d s^h)\rvert} \tag{5.17}$$

Given meta dependencies for classes (i) - (iii), we define the complete feature set for an expert and a decision candidate consisting of all meta dependencies for all experts and pairwise

expert combinations. For expert e^h, the set of meta dependencies of expert e^h for decision candidate ${}^d s^h$ is construed as union over all kernels, i.e.:

$$\mathbf{MD}(e^h, {}^d s^h) = \cup_{\forall \kappa \in K} MD^{\kappa,l}(e^h, {}^d s^h) \tag{5.18}$$

The kernel-dependent set of meta dependency $\mathbf{MD}^{\kappa}(e^h, {}^d s^h)$ then consists of all single- and pairwise meta dependencies for κ and the respective behavioral measures β^l as well as conditions ψ^l, instantiated for all residual experts.

$$\begin{aligned} MD^{\kappa}(e_i^h, {}^d s^h) = \langle & \mathrm{MD}^{\kappa,1}_{\mathrm{SINGLE}}(e_i^h, {}^d s^h), \\ & \cup_{\forall e_j^h \in E^h \setminus e_i^h \forall l \in \{2,3,4,5\}} \mathrm{MD}^{\kappa,l}_{\mathrm{INTRA}}(e_i^h, e_j^h, {}^d s^h), \\ & \cup_{\forall e_j^{h+1} \in E^{h+1}} \mathrm{MD}^{\kappa,6}_{\mathrm{INTER}}(e_i^h, e_j^{h+1}, {}^d s^h), \\ & \cup_{\forall e_j^{h+1} \in E^{h+1}} \mathrm{MD}^{\kappa,7}_{\mathrm{INTER}}(e_i^h, e_j^{h-1}, {}^d s^h) \rangle \end{aligned} \tag{5.19}$$

To learn expert weight functions w_{e^h}, single expert meta dependencies measure the likelihood of an expert's success, while pairwise intra-step expert meta dependencies use correlations between two experts. As pairwise inter-step expert meta dependencies take into account future rewards based on experts of the subsequent step, they support learning state-action values in EPs.

Complexity Analysis 5.1. *Calculating a meta dependency is dependent on (i) deriving the neighborhood of the decision candidate, (ii) the number of available kernels and (iii) its type (i.e. single, intra pairwise or inter pairwise). Building the neighborhood for a kernel is a Nearest Neighbors (NN) problem – more specifically fixed-radius NN – where approximate methods have been sufficiently studied. Still, exact fixed-radius NN potentially results in quadratic performance – i.e. $O(|S|^2)$ – because of extensive pairwise comparisons of decision candidates. To this end, the specific computational complexity of approximate methods is dependent on the similarity metric of the respective kernel. If the Euclidean similarity is used, one might, for example, exploit k-d trees [13] to yield quasilinear complexity – i.e. $O(|S|log|S|)$ for neighborhood construction and sublinear complexity for lookups ($O(log|S|)$). For the Cosine- or Jaccard similarity – where vectors are compared – techniques coined Locality Sensitive Hashing (LSH) can be used to achieve linear lookups and quasilinear neighborhood construction times. As a result, the computational complexity for single expert meta dependencies for a single expert and decision candidate is $O(|K||S|log|S|)$ as we have to conduct calculations for each expert of a step and for each kernel. Similarly, complexities for pairwise meta dependencies can be bound by $O(|K||E^h||S|log|S|)$ for the intra-step case and $O(|K||E^{h+1}||S|log|S|)$ (or $O(|K||E^{h-1}||S|log|S|)$) for the inter-step case, as the behavior of an expert has to be compared to others. The time needed for lookups can be neglected as it merely linearly influences the complexity.*

We now present two *Online model-free* RL approaches to learn expert weights with meta dependencies as feature representation.

5.4 Online Reinforcement Learning

In the Online RL setting (see Definition 3.41), one aims to continuously improve predictions with every observed reward. We take a model-free RL approach, thus not learning R and Tr (where Tr is known in our special case) but directly approximating expert weights. Note that generalization across high-dimensional states in RL is a hard problem [77] – numerous approaches deal with function approximation, where linear function approximation suffices [86, 124, 139]). For EPs, where feedback is usually inferred from relatively small training sets, the problem becomes even more difficult.

To this end, we use meta dependencies as direct estimators for expert performances. Over multiple episodes, we keep and update *meta-weights* for meta dependencies to gradually get more accurate predictions. We first apply the EWH algorithm to EPs and meta dependencies. Based on the latter, different measures related to budgeted expert advice, contextual bandits and MDPs are integrated to account for imperfect information due to the expert budget as well as the influence of the current decision on future experts.

5.4.1 EWH with Meta Dependencies

Based on the prior defined EWH update rule (see Definition 3.31), the mapping to EPs is straightforward in that we need to extend it with decision candidates. For an EP, the essential EWH update rule for $w_{e^h}(s_z)$ can be defined as:

$$w_{e^h}(s_{z+1}) = w_{e^h}(s_z)\ exp\left(\frac{\sum_{{}^d s^h \in s_z} R^{\text{SCALED}}({}^d s^h, e^h(({}^d s^h))}{\sum_{{}^d s^h \in s_z} \mathbb{1}_{\{\text{q}(e^h, s^h)=1\}}}\ \eta^{\text{EWH}}\right) \tag{5.20}$$

where $\text{q} : E \times S \rightarrow \{0,1\}$ denotes if expert e was queried for s, η^{EWH} is the EWH influence parameter of the update and R^{SCALED} returns the rewards rescaled to $[-1,1]$.

The update rule iterates through all decision candidates of a state and keeps a global weight for each expert, as it does not incorporate any available *contextual information*. To be able to exploit contextual information, we apply EWH directly to meta dependencies and refer to them as *meta-experts*. The set of meta dependencies ME is defined as follows:

$$\forall \text{MD}^i(e_i^h, {}^d s^h) \in \mathbf{MD}(e^h, {}^d s^h) \exists me_i : me_i({}^d s^h) = \text{MD}^i({}^d s^h)\ \theta_l^{\kappa,\delta}({}^d s^h) \tag{5.21}$$

A meta-expert thus predicts the value of meta dependency $\text{MD}^i({}^d s^h)$ [2] for candidate ${}^d s^h$ weighted by $\theta^{\kappa,\delta}({}^d s^h)$.

[2] For simplicity $\text{MD}^i({}^d s^h)$ refers to either $\text{MD}^{\kappa,l}(e_j^h, e_k^h, {}^d s^h)$, $\text{MD}^{\kappa,l}(e_j^h, e_k^{h+1}, {}^d s^h)$ or $\text{MD}^{\kappa,l}(e^h, {}^d s^h)$

To this end, let $\chi : ME \rightarrow E$ be the assignment function from meta-experts to experts, and let the inferred reward function for meta-experts be defined as follows:

$$R^{meta}({}^{d}s^{h}, me) = me({}^{d}s^{h})\ R^{\text{SCALED}}({}^{d}s^{h}, \chi(me)({}^{d}s^{h})) \tag{5.22}$$

The prediction of the meta expert thus weights the received reward of the assigned expert. Similar to experts, each meta-expert gets assigned a weight function $w_{me} : S \rightarrow \mathbb{R}$. The resulting update rule for meta-weights is:

$$w_{me}(s_{z+1}) = w_{me}(s_z)\ exp\left(R^{\text{AVG}}(s_z, me)\ \eta^{\text{EWH}}\right) \tag{5.23}$$

where $R^{\text{AVG}}(s_z, me) = \frac{\sum_{d_{s \in s_z}} R^{meta}({}^{d}s^{h}, me)}{\sum_{d_{s \in s_z}} \mathbb{1}_{\{\text{q}(\chi(me), s_z)=1\}}}$ is the average reward for all decision candidates, which we define to ease illustration. Note that the meta-expert weight update rule is defined over complete states and not single decision candidates. A meta-weight stays constant for all decision candidates of a single state in an episode and a horizon.

An expert weight is then is defined as:

$$w_{e^h}({}^{d}s^{h}) = \frac{\sum_{me} \mathbb{1}_{\{\chi(me)=e^h\}}\ me({}^{d}s^{h})\ w_{me}(s_z)}{\sum_{me} w_{me}(s_z)} \tag{5.24}$$

However, EWH does not deal with incomplete information resulting from budget C^h_{EXP}. We thus reuse techniques from budgeted expert advice and adversarial contextual bandits to account for state-expert pairs we did not query.

5.4.2 EWH with Incomplete Information

As an EP allows to query C^h_{EXP} different state-expert pairs, we extend the weight updates with *importance weighting*, which was proposed for budgeted expert advice. This is achieved by normalizing the reward by the probability of all queried experts in an episode and horizon, where the importance weight function is defined as:

$$iw(z,h) = \sum_{s \in \hat{S}^h_z} \sum_{e \in E^h} \mathbb{1}_{\{\text{q}(e,s)=1\}} P_{\text{EXPERT}}(s,e) \tag{5.25}$$

We thus iterate through all possible states $\hat{S}^h_z$ and use the expert probability distribution to get an estimate for the respective state-expert pair. The resulting importance weight function is then used as *regularizer* for average rewards, as it either reinforces the update if the query probability was relatively low or weakens it otherwise (see Definition 5.26).

$$w_{me}(s_{z+1}) = w_{me}(s_z)\ exp\left(\frac{R^{\text{AVG}}(\hat{S}^h, me)}{iw(z,h)}\ \eta^{\text{EWH}}\right) \tag{5.26}$$

To this end, it is essential to deal with the exploration-exploitation problem with respect to non-queried experts. The *EXP family* of *adversarial contextual bandits* (e.g. [6]) are based on EWH, but extend the approach to update expert weights for non-taken *actions*. While in contextual bandits all experts are queried to suggest an action, in EPs all meta-experts are queried to suggest a weight for their assigned expert. We therefore directly reuse methods from the *EXP4.P* [15] algorithm, an adversarial bandit approach with high expected reward. Here, the probability of choosing an action is equivalent with an expert weight in EPs, which we adapt with respect to the *minimal probability* to choose any expert, namely $p^h_{min} = [0, \frac{1}{\text{N}^h}]$.

$$w_{e^h}({}^d s^h) = (1 - \text{N}^h\ p^h_{min}) \frac{\sum_{me} \mathbb{1}_{\{\chi(me)=e^h\}}\ me({}^d s^h)\ w_{me}(s_z)}{\sum_{me} w_{me}(s_z)} + p^h_{min} \tag{5.27}$$

To update the weights of meta-experts, we build on the expert update of *EXP4.P*, which uses minimal probability p^h_{min} to align the weight update and exploits prior defined importance weight function *iw* with respect to *confidence bounds* to express the *variance* of the reward. It can be directly used for EPs:

$$w_{me}(s_{z+1}) = w_{me}(s_z)\ exp\left(\frac{p^h_{min}}{2}\left(\mathbb{1}_{\{\chi(me)=e\}}\frac{R^{\text{AVG}}(\hat{S}^h, me)}{iw(z,h)} + \frac{1}{iw(z,h)}\ \eta^{\text{EWH}}\right)\right) \tag{5.28}$$

We finally align meta-expert weights with respect to the actions taken in the EP. We thus *reinforce* the meta reward if the meta-expert was correct but its expert' action was not chosen or if the meta-expert was wrong but its expert's action was chosen, i.e.:

$$\begin{aligned} \text{wrong}(me, s_z) = & \mathbb{1}_{\{\text{q}(\chi(me), s_z)=1\}} \\ & \left(\mathbb{1}_{\{(e(s_z))=a^*) \wedge (R^{meta}(s_z, e(s))>0))\}}\right. \\ & \left. + \mathbb{1}_{\{(e(s_z))\neq a^*) \wedge (R^{meta}(s_z, e(s))<0)\}}\right) \end{aligned} \tag{5.29}$$

The resulting *boosting* factor then consists of hyperparameter $\beta \in \mathbb{R}^+$ which is either active or inactive, i.e. $\text{boost}(me, s_z) = \beta^{\ \text{wrong}(me, s_z)}$. The final meta-expert weight update rule is:

$$w_{me}(s_{z+1}) = w_{me}(s_z)\ exp\left(\frac{p^h_{min}}{2}\ \text{boost}(me, s_z)\left(\mathbb{1}_{\{\chi(me)=e\}}\frac{R^{\text{AVG}}(\hat{S}^h, me)}{iw(z,h)} + \frac{1}{iw(z,h)}\ \eta^{\text{EWH}}\right)\right) \tag{5.30}$$

Both meta-expert weight update rule and expert probability do not specifically incorporate Value-based or Policy Search RL techniques, i.e. future rewards are not *explicitly* optimized (see Equation 5.4 for explicit optimization). We incorporate future rewards via meta dependencies, where meta-experts for pairwise inter-step experts take into account how an expert's suggestion influences future results.

Complexity Analysis 5.2. *As the proposed Online RL approach is based on the EWH algorithm with computational complexity $O(|E^h|)$, one only has to additionally take into account the complexity of calculating all meta dependencies – i.e. $O((|E^h|^2 + |E^h||E^{h+1}| + |E^h||E^{h-1}|)(|K||S|log|S|))$.*

While Online RL directly uses the received feedback, Batch RL [80] entails to conduct less frequent updates, which might enable more efficient use of rewards. We now present a Batch RL approach for learning expert weights in EPs, where we use SRL for function approximation.

5.5 Batch Reinforcement Learning

Other than in Online RL, we now conduct updates after a fixed number of episodes for Batch RL. The central idea is to use a scalable SRL approach – i.e. PSL – for meta dependencies to learn a collective model (see Definition 3.22) for experts. SRL is especially useful due to the relational nature of meta dependencies as well as the availability of kernel functions, which enable to create links (i.e. relations) among decision candidates. We defer the learning process for expert weights to exploit PSL for a number of states, expert weights and rewards, as it is a batch learning approach (see Definition 3.11). By modeling relations between decision candidates, we also align *predictions* for expert weights in a *pool-based* prediction setting (see Definition 3.15).

We define an adapted version of an EP, where we enable Batch RL and pool-based prediction. The resulting *Batch EP* is defined to deal with Z_{POOL} samples within a single step h before continuing to $h+1$. With respect to batch updates, one waits Z_{BATCH} episodes. The assumption is realistic for problems where decisions can be deferred for a certain amount of time (which is dependent on how fast new task instances are available).

Definition 5.9 (Finite-horizon Batch Expert Process)**.** *A finite-horizon Batch EP is a* 9*-tuple $(S, E, A, R, T, \text{H}, C_{EXP}, Z_{BATCH}, Z_{POOL})$ which extends a finite-horizon EP (see Definition 5.1) with batch size hyperparameter $Z_{BATCH} \in \mathbb{N}^+$ and pool size hyperparameter $Z_{POOL} \in \mathbb{N}^+$. While Z_{BATCH} corresponds to the number of episodes the agent defers updating expert weights,*

Z_{POOL} defines the number of episodes the agent is allowed to defer acting. Given that the agent went through the process for episode $z-1$, it can observe states $s_z,\ldots,s_{z+Z_{POOL}}$ before acting (see Definition 3.15). Similarly, the agent collects trajectories for episodes $s_z,\ldots,s_{z+Z_{POOL}}$ before updating (Definition 3.42).

To use all available information for the Batch EP, we train two models for each step – an a priori model before querying any expert and an a posteriori model after having queried all state-expert pairs [3]. To this end, PSL enables directly integrating kernels into the model, thereby transferring inferred weights to other decision candidates by collective inference.

The section proceeds as follows: we first describe PSL, a template language for generating Hinge-Loss Markov Random Fields (HLMRFs). We, then, elaborate on how to model weights w_{e^h} to approximate Q.

5.5.1 Probabilistic Soft Logic and Hinge-Loss Markov Random Fields

A HLMRF [8] is a conditional probabilistic model over continuous random variables. HLMRFs are defined as probability densities:

$$\mathrm{P}(Y|X) \propto \frac{1}{Z^{norm}} exp[-\sum_{j=1}^{M} \lambda_j \phi_j(Y,X)]$$

with Z^{norm} the respective normalization constant, weights λ_j, $\phi_j(Y,X) = [max\{l_j(Y,X),0\}]^{p_j}$ the hinge-loss potential functions, linear function l_j and $p_j \in \{1,2\}$.

PSL [24, 67] is a modeling language for HLMRFs. A model in PSL comprises weighted, first-order logic rules (templates for ϕ_j) with features defining a Markov network. Here, PSL relaxes conjunctions, disjunctions and negations as:

$$A \wedge B = \max\{A+B-1,0\}$$

$$A \vee B = \min\{A+B,1\}$$

$$\neg A = 1-A$$

with $A \rightarrow [0,1]$ and $B \rightarrow [0,1]$.

The use of hinge-loss feature functions makes inference tractable, as it is a convex optimization problem and thus potentially scales to large datasets. We now present our PSL model based on meta dependencies for EPs.

[3] Note that we gradually increase our training set with each batch, where densities provide a natural way to passively forget older samples. An alternative would be to actively forget all old batches but reuse old weights as priors.

5.5.2 Meta Dependencies with PSL

We learn expert weights w_{e^h} by using PSL to construct different potential functions $\phi_j(X,Y)$ and learn their weights λ_j. We therefore construct a single PSL model per step and, other than in our online approach, learn one expert weight function for all experts E^h, as our intuition is that meta dependencies enable to generalize across different experts of the same step. Labels for expert weights are manually constructed based on Equation 5.4 and normalized *after* all samples are gathered for a batch episode.

All n-ary relations we define map their variables to a real number between zero and one, e.g. Relation : $\text{Var}_i \times \text{Var}_j \rightarrow [0,1]$ for $n = 2$. An expert weight w_{e^h} is represented as relation $\text{W}(e^h, {}^d s^h)$. Meta dependency relations for pairwise intra-step experts $\text{MD}^{\kappa,l}_{\text{INTRA}}(e^h_i, e^h_j, {}^d s^h)$, pairwise inter-step experts $\text{MD}^{\kappa,l}_{\text{INTER}}(e^h_i, e^{h+1}_j, {}^d s^h)$ and single experts $\text{MD}^{\kappa,l}_{\text{SINGLE}}(e^h, {}^d s^h)$ are pre-computed meta dependencies with respective densities $\text{MD}_{\text{INTRA}}\text{–D}^{\kappa,l}(e^h_i, e^h_j, {}^d s^h)$, $\text{MD}_{\text{INTER}}\text{–D}^{\kappa,l}(e^h_i, e^{h+1}_j, {}^d s^h)$ and $\text{MD}_{\text{SINGLE}}\text{–D}^{\kappa,l}(e^h, {}^d s^h)$. Densities are calculated based on $\theta^{\kappa,\delta}_l({}^d s^h)$, as defined for single- as well as pairwise meta dependencies. $\text{Tr}(e^h, {}^d s^h, {}^c s^{h+1})$, the transition relation, denotes that an expert suggests decision candidate ${}^c s^{h+1}$ for ${}^d s^h$.

We first model rules for single expert meta dependencies, where we directly use the latter as positive influence on expert weight relation W_{e^h}. The negation of the rule holds as well and has to be explicitly modeled. A priori and a posteriori models consist of the same rules, as querying the respective expert does not give us more information in the single expert case.

$$\begin{aligned}
\text{MD}^{\kappa,l}_{\text{SINGLE}}(e^h, {}^d s^h) \wedge \text{MD}_{\text{SINGLE}}\text{–D}^{\kappa,l}(e^h, {}^d s^h) &\implies \text{W}(e^h, {}^d s^h) \\
\neg\text{MD}^{\kappa,l}_{\text{SINGLE}}(e^h, {}^d s^h) \wedge \text{MD}_{\text{SINGLE}}\text{–D}^{\kappa,l}(e^h, {}^d s^h) &\implies \neg\text{W}(e^h, {}^d s^h)
\end{aligned}$$

Pairwise intra-step expert rules express that the weight of an expert increases if respective meta dependencies are sufficiently high. To account for the current expert predictions, we additionally model agreement and disagreement relations and extend the rules with them. We first, model rules for the a priori setting where experts have not been queried yet. Here, we define relation $\text{Agree}(e^h_i, e^h_j, {}^d s^h)^l$, which is evaluated for the complete neighborhood in order to return a probability of agreement or disagreement based on the modeled conditions ψ^l.

$$\begin{aligned}
\text{MD}^{\kappa,l}_{\text{INTRA}}(e^h_i, e^h_j, {}^d s^h) \wedge \text{MD}_{\text{INTRA}}\text{–D}^{\kappa,l}(e^h_i, e^h_j, {}^d s^h) & \\
\wedge \text{Agree}(e^h_i, e^h_j, {}^d s^h)^l &\implies \text{W}(e^h_i, {}^d s^h) \\
\neg\text{MD}^{\kappa,l}_{\text{INTRA}}(e^h_i, e^h_j, {}^d s^h) \wedge \text{MD}_{\text{INTRA}}\text{–D}^{\kappa,l}(e^h_i, e^h_j, {}^d s^h) & \\
\wedge \text{Agree}(e^h_i, e^h_j, {}^d s^h)^l &\implies \neg\text{W}(e^h_i, {}^d s^h)
\end{aligned}$$

After having queried C^h_{EXP} state-expert pairs, we can exploit new information by only increasing expert weights if there is agreement.

$$\begin{aligned}
&\mathrm{MD}^{\kappa,l}_{\mathrm{INTRA}}(e^h_i, e^h_j, {}^d s^h) \wedge \mathrm{MD}_{\mathrm{INTRA}} - \mathrm{D}^{\kappa,l}(e^h_i, e^h_j, {}^d s^h) \\
&\wedge \mathrm{Tr}(e^h_i, {}^d s^h, {}^c s^{h+1}) \wedge \mathrm{Tr}(e^h_j, {}^d s^h, {}^c s^{h+1}) \implies \mathrm{W}(e^h_i, {}^d s^h) \\
&\neg\mathrm{MD}^{\kappa,l}_{\mathrm{INTRA}}(e^h_i, e^h_j, {}^d s^h) \wedge \mathrm{MD}_{\mathrm{INTRA}} - \mathrm{D}^{\kappa,l}(e^h_i, e^h_j, {}^d s^h) \\
&\wedge \mathrm{Tr}(e^h_i, {}^d s^h, {}^c s^{h+1}) \wedge \mathrm{Tr}(e^h_j, {}^d s^h, {}^c s^{h+1}) \implies \neg\mathrm{W}(e^h_i, {}^d s^h)
\end{aligned}$$

Rules for pairwise inter-step expert meta dependencies express that current expert weights have to be high if respective experts are either estimated to perform well or if they generated good states. Here, we use the same rules for both the a priori and the a posteriori setting.

$$\begin{aligned}
&\mathrm{MD}^{\kappa,l}_{\mathrm{INTER}}(e^h_i, e^{h+1}_j, {}^d s^h) \wedge \mathrm{MD}_{\mathrm{INTER}} - \mathrm{D}^{\kappa,l}(e^h_i, e^{h+1}_j, {}^d s^h) \\
&\qquad\qquad \implies \mathrm{W}(e^h_i, {}^d s^h) \\
&\neg\mathrm{MD}^{\kappa,l}_{\mathrm{INTER}}(e^h_i, e^{h+1}_j, {}^d s^h) \wedge \mathrm{MD}_{\mathrm{INTER}} - \mathrm{D}^{\kappa,l}(e^h_i, e^{h+1}_j, {}^d s^h) \\
&\qquad\qquad \implies \neg\mathrm{W}(e^h_i, {}^d s^h)
\end{aligned}$$

We propagate inferred weights by relation $\mathrm{Similar}_\kappa({}^d s^h_i, {}^c s^h_j)$, thus using kernels to exploit similarities among decision candidates. It becomes clear that pool-based predictions might potentially increase the performance of the approach by clustering similar decision candidates through kernels.

$$\begin{aligned}
\mathrm{Similar}_\kappa({}^d s^h_i, {}^c s^h_j) \wedge \mathrm{W}(e^h, {}^d s^h_i) &\implies \mathrm{W}(e^h, {}^c s^h_j) \\
\mathrm{Similar}_\kappa({}^d s^h_i, {}^c s^h_j) \wedge \neg\mathrm{W}(e^h, {}^d s^h_i) &\implies \neg\mathrm{W}(e^h, {}^c s^h_j)
\end{aligned}$$

We finally align experts with similar outputs by driving their weights to be the similar. For the a priori setting, we define rules based on the agreement probability.

$$\begin{aligned}
\mathrm{W}(e^h_i, {}^d s^h) \wedge \mathrm{Agree}(e^h_i, e^h_j, {}^d s^h) &\implies \mathrm{W}(e^h_j, {}^d s^h) \\
\neg\mathrm{W}(e^h_i, {}^d s^h) \wedge \mathrm{Agree}(e^h_i, e^h_j, {}^d s^h) &\implies \neg\mathrm{W}(e^h_j, {}^d s^h)
\end{aligned}$$

Based on the observed expert action candidates, rules for the a posteriori case are modeled as:

$$\begin{aligned}
&\mathrm{W}(e^h_i, {}^d s^h) \wedge \mathrm{Tr}(e^h_i, {}^d s^h, {}^c s^{h+1}) \wedge \mathrm{Tr}(e^h_j, {}^d s^h, {}^c s^{h+1}) \\
&\qquad\qquad \implies \mathrm{W}(e^h_j, {}^d s^h) \\
&\neg\mathrm{W}(e^h_i, {}^d s^h) \wedge \mathrm{Tr}(e^h_i, {}^d s^h, {}^c s^{h+1}) \wedge \mathrm{Tr}(e^h_j, {}^d s^h, {}^c s^{h+1}) \\
&\qquad\qquad \implies \neg\mathrm{W}(e^h_j, {}^d s^h)
\end{aligned}$$

For learning weights of the PSL model and inferring w_{e^h}, we use an approximate *maximum likelihood* weight learning algorithm and *most probable explanation* (MPE) for probabilistic inference [7]. More specifically, we reuse a *voted perceptron*, where an iteration of weight

learning entails to run inference and condition on the available training data (in our case the expert weights we calculated) to generate target values, and then to update the PSL model weights in terms of maximizing the agreement with prior set targets without using training data. The inference method (which is also used for training), MPE, exploits the fact that soft truth values (other than binary truth values as used, for example, in Markov Logic Networks (MLNs)) are used and that inference can be cast to a convex optimization problem (other than combinatorial problems as available in MLNs).

Complexity Analysis 5.3. *Quantifying the complexity of learning weights of an instantiated HLMRF can be approached by dealing with the time needed for individual steps, respectively comprising grounding, inference and learning. The general time needed for grounding is dependent on (i) the number of available potential functions (i.e. PSL rules), (ii) the complexity of individual potential functions and (iii) the length of the batch episodes. The complexity of potential functions (point (ii)) is not absolutely quantifiable, as it depends on the arity of relations, number of relations, involved variables and the availability of latent variables. The complexity of grounding thus exceeds the complete Online RL approach, as we additionally incorporate PSL rules (i.e. potential functions) for kernels (resulting in groundings of all pairwise decision candidates of the batch) and end up with* $O((|E^h|^2 + |E^h||E^{h+1}| + |E^h||E^{h-1}|)(|K||S|log|S||Z_{BATCH}|^2))$. *The time needed for inference and adapting the inferred weights based on training data is dependent on the chosen learner.*

5.6 Discussion

The EP framework builds on MDPs and (budgeted-) expert advice, and enables the use of RL techniques to deal with (i) partial information resulting from budget constraints, (ii) future impacts of expert outputs and (iii) unknown expert behavior for states. While we showed how to redefine value functions in terms of expert weights to enable optimization, our approaches additionally exploit meta dependencies to learn pairwise expert behavior.

While our Online RL approach cannot exploit batch learning and pool-based predictions, it is still favorable when computational resources are scarce. The EWH updates are efficient, where calculating meta dependencies is costing the most. There are, however, efficient approximations for the corresponding Nearest Neighbor problem.

Although RL algorithms have to be adapted to be used for EPs (especially to deal with a large amount of possible state-expert pairs based on uncertainty in the state spaces), their basic functionality can be directly reused.

5.7 Summary

In this chapter, we formalized EPs for multi-step expert advice by specializing MDPs with available experts as well as execution budgets. EPs are suitable to maximize the number of

correct task solutions, as they define essential properties of expert weight functions. The latter express the expected performance of an expert for an individual decision candidate, but also take into account the impact on experts of future steps. Our work on EPs addressed Research Question 1, where the corresponding Hypothesis 1 can be partially confirmed and will be revisited after empirically evaluating EPs in Chapter 9.

For learning fine-grained expert weights for decision candidates of states, we introduced meta dependencies as features which calculate performance-related relations among single- or pairs of experts. We showed that meta dependencies can be best exploited by Online RL based on the EWH algorithm and Batch RL with PSL as function approximator, both enabling to learn expert weights for EPs. We therefore addressed Research Question 2 with our RL algorithms for EPs and, similar to Hypothesis 1, can partially confirm Hypothesis 2, but have to revisit the latter after the empirical evaluation in Chapter 9.

Chapter 6

Learning with Multiagent Expert Processes

This chapter proposes the MEP framework as solution to Research Question 3 which deals with the expert correlation problem. Hypothesis 3 proposed a Multiagent perception of the problem, where coordination approaches for cooperating agents have to be developed. To fully confirm Hypothesis 3, we empirically evaluate three Multiagent coordination approaches for MEPs in Chapter 7.
Our reference for this chapter is [104].

6.1 Introduction

The prevalent SAS perception in RL entails that a single decision-maker learns a global estimate of the model, value-function or policy, depending on the learning scenario. With respect to EPs (see Definition 5.1), experts are merely queried and do not learn any adaptive behavior themselves, which might cause learned policies to misperform when experts correlate in their joint errors. Expert correlation might occur *globally* (i.e. for almost all data points) which can be easily detected, or *locally* (i.e. only for data with specific characteristics) which is hard to disclose and manage. It is thus sensible to investigate expert correlation for each individual data point and to only combine the prediction of two experts if their errors are sufficiently independent for data points with similar characteristics.

To this end, recent advances in *Separation of Concerns* (SoC) [82, 134] promote distributing complex single-agent tasks, such as learning to play games based on observable frames, to multiple agents. The authors show that decomposing such tasks into conceptually different steps with different reward functions enables to train separate agents in a MAS in order to minimize convergence time or improve predictive performance. As a consequence, distributing the control of experts within multi-step expert advice to multiple agents might result in similar improvements.

In addition to dealing with local expert correlation, a MAS setup has several advantages, namely (i) enabling to model novel, real-world problem settings with multiple stakeholders where experts are not publicly available but cooperation in a MAS provides mutual benefits, (ii) adding redundancy to centralized architectures, where experts can be shared among agents to coordinate missed updates and (iii) speeding up learning by first training multiple agents independently and eventually have them coordinate their learned policies [94].

In this chapter, we investigate these findings for the problem of multi-step expert advice, where we frame the setup as cooperative MAS, more specifically as MEP, where each expert is weighted and controlled by an individual agent. The resulting problem then is to learn how to coordinate expert weights and actions to optimize the decentralized weighted majority voting among agents. The central intuition is that the coordination process might enable to reduce the available bias when experts strongly correlate.

Figure 6.1 denotes the general schema of multi-step expert advice in a MAS. In the bottom layer, the conceptual steps of the respective multi-step task are shown, e.g. NER and NED for NERD. For all available steps, agents need to take joint decisions (i.e. choose joint actions) which can, for example, be the result of the weighted majority voting of all agents of the step. In the top layer, all available agents are shown, where an agent controls one or more experts and has to primarily coordinate its available actions (i.e. the action it receives via its experts) with residual agents of the same step. To account for future impacts of its decisions, an agent can also coordinate with agents of the subsequent step. The opposite communication direction is also possible, as agents might want to learn which agents of prior steps produce good or bad actions.

For better illustration, consider the following example for conceptual steps NER and NED of NERD.

Example 6.1 (The NERD task in a MAS)**.** *Agents for step NER need to sequentially decide for pieces of texts, such as* `Michael Jackson plays basketball.`*, which token sequences constitute named entities. Given black-box experts solving NER, agents coordinate prevalent actions, e.g. if* `Michael Jordan` *is a named entity or not. After a number of coordination rounds, the weighted majority estimate – i.e. the joint action – is passed to agents of step NED. To this end, NER agents also need to communicate with NED agents to find out who exploits their individually proposed actions.*

By distributing the control to multiple agents, we extend EPs to the MAS setting by specializing MMDPs with the availability of experts and budgets. We then present an adequate feature representation based on meta dependencies for individual agents, which expresses various relationships among experts and agents with respect to the coordination problem.

We first approach *passive coordination* in MEPs (i.e. the independent learner setting, see Definition 3.46), where agents only know about their own experts' predictions and the result of the weighted majority voting, but still have to align each other. Our passive coordination approach builds on the EWH algorithm to adapt agent weights with respect to the weighted majority vote of all agents.

For agents to *actively* coordinate their experts, we define a *decision protocol* inspired by *swarm intelligence*. Here, agents receive information about all residual actions to reconstruct the global state (i.e. joint-action learner setting (see Definition 3.45)). Other than in one-step coordination settings as available in SoC or most hierarchical RL approaches, we model a designated decision process for coordination. The latter – which we refer to as expert coordination process (ECP) – yields local finite-horizon MDPs with continuous action spaces

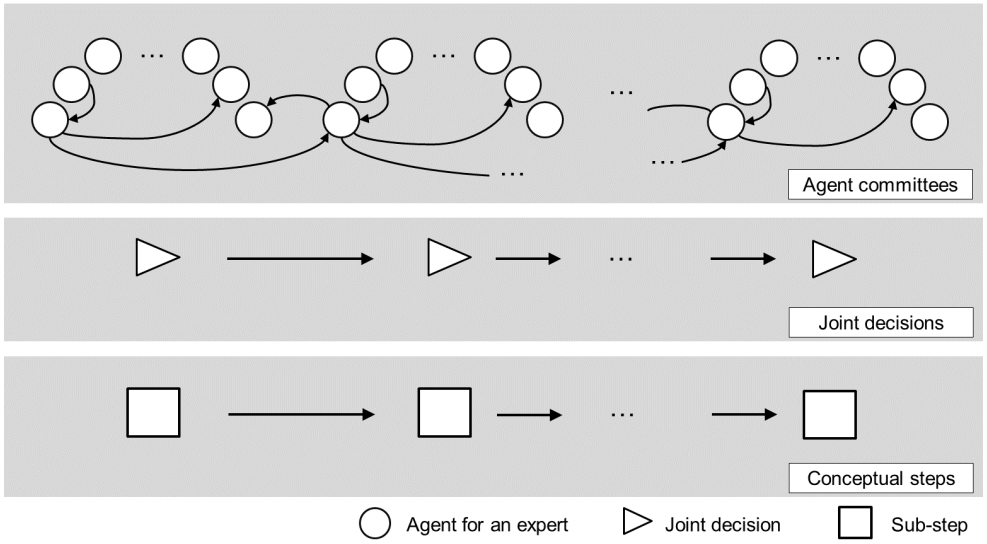

Figure 6.1: Schema of Multi-Step Expert Advice in a MAS.

for each participating agent, where non-stationary coordination policies have to be learned. Within the ECP, an agent learns a relative weight function which quantifies its confidence in its experts' actions with respect to residual agents. We first develop a heuristic based on meta dependencies which adapts agent weights for their experts on a rule basis. We then propose a Policy Search RL approach to learn robust non-stationary agent policies, where each learned model reflects a single decision in the fixed-length coordination process.

6.1.1 Challenges

In addition to challenges 1-4, this chapter deals with the following challenge, as initially defined in Section 1.2:

- When learning models based on prediction with expert advice or ensemble learning, one often suffers from expert correlation, which results in overestimating potentially wrong expert hypotheses (*Challenge 5*).

6.1.2 Contributions

This chapter contributes in the following ways:

- We extend our EP formalization to MAS and define MEPs with a passive- and an active coordination protocol. Passive coordination enables to develop lightweight solutions to decorrelate expert weights. For active coordination, we define a novel decision process

(referred to as ECP), which extends one-step coordination in MAS to a multi-step decision process. The ECP enables to develop complex coordination approaches, as more information are available about other agents (*Contribution 3.1*).

- We extend meta dependencies to relational agent features, which are used as feature representation for active coordination (*Contribution 3.2*).
- We propose a lightweight coordination approach for MEPs with passive coordination, where agents cannot communicate their actions (*Contribution 3.3*).
- We develop two coordination approaches for MEPs with active coordination, where agents are allowed to communicate to learn better coordination strategies (*Contribution 3.4*).
- We empirically evaluate our approach for the multi-step NERD task with respect to maximizing the overall expected reward in Chapter 7.

The chapter proceeds as follows: We first formalize our novel problem setting and introduce MEPs in Section 6.2. We then present our approach to passive coordination (Section 6.4, where respective agents receive limited information about their environment. In our active coordination approaches (Section 6.5 and 6.6), we assume full information for all agents, which enables more efficient coordination. We finally conclude with a summary of this chapter in Section 6.8.

6.2 Problem Formalization

We now define MEPs, where cooperative agents share the control over available experts and have to choose *joint actions* which maximize the number of correct task solutions. Figure 6.2 illustrates a MEP, where the joint action is chosen based on an arbitrary function (e.g. weighted majority voting).

Definition 6.1 (Multiagent Expert Processes)**.** *A finite-horizon MEP extends a finite-horizon MMDP (see Definition 3.44) with the availability of experts and is a 11-tuple* $(\{S_\lambda\}_{\lambda\in\Lambda}, \Lambda, \{E_\lambda\}_{\lambda\in\Lambda}, \{A_\lambda\}_{\lambda\in\Lambda}, \{A_\lambda^{COMM}\}_{\lambda\in\Lambda}, R, Tr, T, H, C_{EXP}, C_{COMM})$.

- *Let* $\{S_\lambda\}_{\lambda\in\Lambda}$ *be the set of states,* $A_{\lambda\in\Lambda}$ *be the set of environment actions and* $A_{\lambda\in\Lambda}^{COMM}$ *be the set of communication actions, as defined for MMDPs.*
- *Let* C_{COMM} *the communication budgets as defined for MMDPs. It confines the interaction among agents* $(\lambda_i^h, \lambda_j^h)$, $(\lambda_i^h, \lambda_j^{h+1})$ *and* $(\lambda_i^{h+1}, \lambda_j^h)$ *through communication actions in* $A_{\lambda\in\Lambda}^{COMM}$*, where each agent gets to query respective agents* C_{COMM} *times* [1].

[1] Without loss of generality, we assume unidirectional communication, where an agent has to actively use its budget.

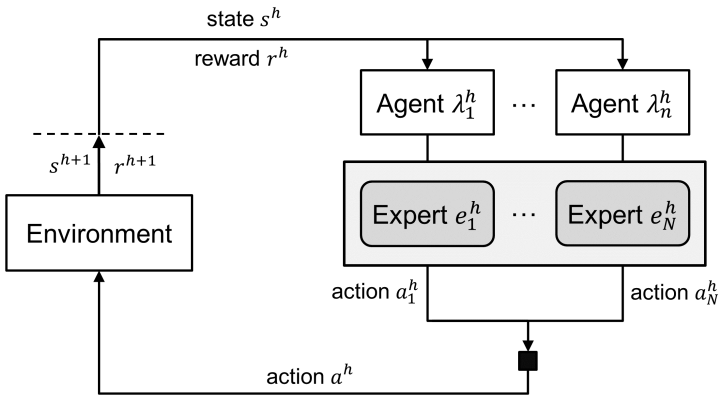

Figure 6.2: Schema of Multi-Step Expert Advice in the MDP framework with an arbitrary mechanism to choose joint actions.

- *Let Tr be the transition function, R the reward function, H the horizon and C_{EXP} the state-expert budget, as defined for EPs.*
- *Let Λ be the set of agents, where an agent $\lambda \in \Lambda$ is a function over decision candidates and coordination steps to actions, i.e. $\lambda_i^h : T \times S^h \to A^h$*
- *Let E_λ be the set of experts for agent $\lambda \in \Lambda$, where an expert $e \in E_\lambda$ is a function from states to actions, i.e. $e : S \to A$, as defined for EPs.*
- *Let $T \in \mathbb{N}^{\geq 0}$ be the number of coordination steps.*

The **protocol** of a MEP proceeds in episodes $z = 1, \ldots, Z$:

- For steps $h = 1, \ldots, H$:
  - Each λ_i receives state s_z^h of episode z.
  - Each agent chooses and queries C^{EXP} state-expert pairs (s_z^h, e^h) and retrieves actions $a_1^h, \ldots, a_{C_{\text{EXP}}}^h$.
  - Each agent chooses an action and sets its initial weights, $p_i(1, {}^d s)$, based on its highest weighted expert.
  - A **coordination protocol** finds joint action a^*
  - The agent committee predicts action a_t^* and each agent *observes* rewards $R(s_z^h, a^h) \forall a^h \in A^h$.

The MEP protocol is defined analogously to EPs, where all agents have to choose and query experts, and calculate weights. The challenge for agents is to find a *joint action* (i.e. a_t^*)

for every step h. The MEP protocol was kept variable with respect to the used *coordination protocol*, as we distinguish among *passive coordination* and *active coordination*. Coordination steps $t \in 1,\ldots,T$ are only used for the active coordination setting, and will be kept static (i.e. $t = 1$ throughout the MEP) for passive coordination.

Before introducing properties of value functions, we define the relationship between agents and their experts, as well as available weight functions.

Definition 6.2 (Agent-expert relationship, agent- and expert weights, agent prediction)**.** *An agent λ_i^h manages expert set $e_{i1},\ldots,e_{iM} \in E_i^h$ with $M \in \mathbb{N}^+$. Let $w_i : E_i^h \times S^h \to \mathbb{R}^{\geq 0}$ be the expert weight function of λ_i to score its experts. Let now e_i^h denote the expert with the highest estimated weight for a decision candidate:*

$$e_i^h : w_i(e_i^h, {}^d s^h) = \max_{e^h \in E_i^h} w_i(e^h, {}^d s^h) \tag{6.1}$$

The agent weight $p_i^h : T \times S^h \to \mathbb{R}^{\geq 0}$ expresses the confidence in the action of expert e_i^h, the expert with the highest estimated weight. To this end, an agent predicts the action of expert e_i^h for $t = 1$, i.e. $\lambda_i^h(1, {}^d s^h) = e_i^h({}^d s^h)$

The MEP extensions for individual **agent weights** p_i and **value function** Q^h with regards to EPs are straightforward. Individual agents are now in in control of their weights and predictions, and the same *global* properties hold as for EPs. An agent weight for a decision candidate ${}^d s^h$ has to express its local reward and the maximal Q-value of the next step, reached via any action which includes choosing ${}^d s^h$:

Definition 6.3 (Agent weights in MEPs)**.** *An agent weight in a MEP is defined analogously to expert weights in EPs:*

$$p_i(t, {}^d s^h) = R({}^d s^h, \lambda_i(t, {}^d s^h)) + \sum_{a^h \in A^h_{{}^d s^h}} Tr(s^{h+1} | s^h, a^h) \max_{a^{h+1} \in A^{h+1}} Q^h(s^{h+1}, a^{h+1}) \tag{6.2}$$

with $A^h_{{}^d s^h} = \{a^h | a^h \in A^h : \Delta_{e^h}({}^d s^h) \in a^h\}$ as defined for EPs (i.e. the set of all actions, where ${}^d s^h$ is chosen [2]).

The corresponding state-action value estimate for state s^h and action a^h is based on all involved agents, their weights for all decision candidates and their proposed actions.

Definition 6.4 (State-action value function in MEPs)**.**

$$Q^h(s^h, a^h) = \frac{\sum_{\lambda_i \in \Lambda^h} \sum_{{}^d s^h \in s^h} p_i(t, {}^d s^h) \mathbb{1}_{\{\lambda_i(t, s^h) = a^h\}}}{\sum_{\lambda_i \in \Lambda^h} \sum_{{}^d s^h \in s^h} p_i(t, ds^h)}. \tag{6.3}$$

[2] Note that this set might be incomplete if perfect communication is not possible

After all agents chose their actions and weights, the joint action of all agents has to be determined. We define the joint action to be the *maximal action* with respect to state-action values.

Definition 6.5 (Joint action of a MEP)**.** *The joint action a_t^* of all agents of the current step $\lambda \in \Lambda^h$ at coordination step t is defined as:*

$$a_t^* : Q^h(s^h, a_t^*) = \max_{a \in A^h} Q^h(s^h, a) \tag{6.4}$$

The overall goal of a MEP is to maximize the expected cumulative reward R^{CUM}, as defined for EPs:

Definition 6.6 (Cumulative reward for MEPs)**.** *The overall reward of a MEP, $R^{CUM} : S \times A \rightarrow \mathbb{R}$, is defined as:*

$$R^{CUM} = \mathbb{E}[\sum_{h=1}^{\text{H}} \sum_{d=1}^{\text{D}^h} R^h({}^d s^h, {}^d a^h)] \tag{6.5}$$

Until now, the prediction of agent λ_i^h with static coordination step $t = 1$ is equal to a *local policy* $\pi_i : S^h \rightarrow A^h$ in a single-agent EP. Here, each agent tries to find a *non-stationary* policy, dependent on the respective step h in order to maximize R^{CUM}. We refer to this setting as *passive coordination*. **Passive coordination** is equal to the general MEP schema in Figure 6.2.

Definition 6.7 (Passive Coordination in MEPs)**.** *For a single coordination step, i.e. $T = 1$, agent λ_i^h chooses the action proposed by its expert, i.e. $\lambda_i(1, {}^d s) = e_i({}^d s)$, and only adapts its weight $p_i(1, {}^d s)$ based on the reward of the weighted majority voting of the committee. Hence, we set $C_{COMM} = 0$ and forbid communication. The setting refers to the independent learner setting (see Definition 3.46).*

We now define a novel type of non-stationary MEP policies, which *depend* on coordination step $= 1, \ldots T$ and thus require to act in the ECP. We refer to this setting as *active coordination*.

Active coordination enables agents to negotiate their actions in a designated decision process. Here, agents can communicate with other agents to observe their actions and use this knowledge to better adapt their own weights and actions. Active coordination is illustrated in Figure 6.3, where the novel decision process corresponds to the ECP.

Definition 6.8 (Active Coordination in MEPs)**.** *For $T > 1$ coordination steps, agent λ_i can change its weight $p_i(t, {}^d s)$ as well as its chosen action $\lambda_i(t, {}^d s)$ Here, we set $C_{COMM} = |\Lambda|$ and hence do not constrain communication, i.e. each agent observes all residual action predictions for an episode. We thus follow the joint-action learner setting (e.g. [29]).*

We first explain how predictions of agents are adapted for the MEP and then define the ECP. We extend the agent prediction function in terms of ideas from *swarm intelligence* [146], by enabling an agent to abandon its experts' predictions to follow the action with highest weight.

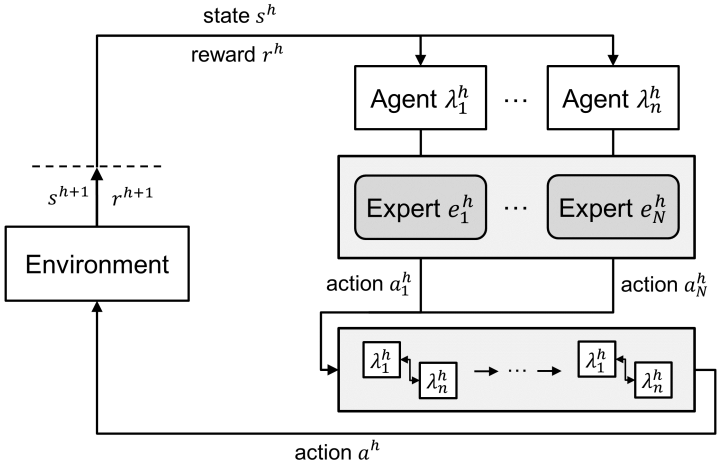

Figure 6.3: Schema of Multi-Step Expert Advice in the MDP framework with active coordination protocol.

Within a coordination step, an agent either chooses the action proposed by its highest weighted expert or follows the committee average a_t^*, resulting in the following agent prediction function at coordination step t.

Definition 6.9 (Agent prediction function for a coordination step)**.** *An agent λ_i predicts the action of its expert e_i for $t = 1$ (as defined before) and adapts the weight based on the following update rule:*

$$\lambda_i(t+1, {}^d s^h) = \max\left(e_i({}^d s^h), {}^d a_t^*)\right) \tag{6.6}$$

where $e_i({}^d s^h)$ is scored by $p_i(t+1, {}^d s^h)$ and ${}^d a_t^*$ by $Q^h({}^d s^h, {}^d a_t^*)$

Definition 6.10. *An ECP extends a MMDP and is a 8-tuple $(S^{ECP}, \Lambda, A^{ECP}, \{E^\lambda\}_{\lambda\in\Lambda}, R^{ECP}, Tr^{ECP}, H, T)$.*

- *Let S^{ECP} be the coordination step dependent state space. A state $s_t^{ECP} \in S^{ECP}$ consists of state information of the respective MEP as well as current action predictions and weight estimates of all agents of the coordination step t, i.e.*

$$s_t^{ECP} = s^h \cup_{\forall \lambda_i^h \in \Lambda^h \forall {}^d s^h \in s^h} \{(\lambda_i^h, {}^d a^h, w) | {}^d a^h = \lambda^h(t, {}^d s^h) \wedge w = p_i^h(t, {}^d s^h)\} \tag{6.7}$$

- *Let Λ be the set of agents, as defined for MEPs.*
- *Let A^{ECP} be the continuous action space.*
- *Let $\{E^\lambda\}_{\lambda\in\Lambda}$ be the agent-dependent expert space, as defined for MEPs.*

- *Let $R^{ECP} : T \times S_t^{ECP} \times A_t^{ECP} \rightarrow \mathbb{R}^{\geq 0}$ be the coordination step dependent reward function.*
- *Let $Tr^{ECP} : S_t^{ECP} \times A_t^{ECP} \rightarrow S_{t+1}^{ECP}$ the transition function.*
- *Let H the step of the current MEP.*
- *Let T be the coordination step, as defined for MEPs.*

The **ECP protocol** proceeds for coordination steps $t = 1, \ldots, T$:

- Each agent takes local action $a_t^{i,\text{ECP}}$ by adapting its MEP weight function $p_i(t+1, {}^d s^h)$.
- Each agent decides on MEP action $\lambda_i(t+1, s^h)$.
- The joint action of the MEP a_t^* is determined as weighted majority vote.
- Each agent receives reward r_t^{ECP}, calculated based on the reward of a_t^* in the MEP (i.e. $R({}^d s^h, a_t^*)$) and action $a_t^{i,\text{ECP}}$.

Actions in the ECP are continuous values, which have effect on the MEP weight functions of an agent, $p_i(t+1, {}^d s^h)$.

Definition 6.11 (Agent actions in the ECP)**.** *By taking action $a_t^{i,ECP} \in A^{ECP}$ in coordination step t, agent λ_i also adapts its weight function of the corresponding MEP:*

$$p_i(t+1, {}^d s^h) \leftarrow p_i(t, {}^d s^h) + a_t^{i,ECP} \tag{6.8}$$

Figure 6.4 summarizes the ECP.

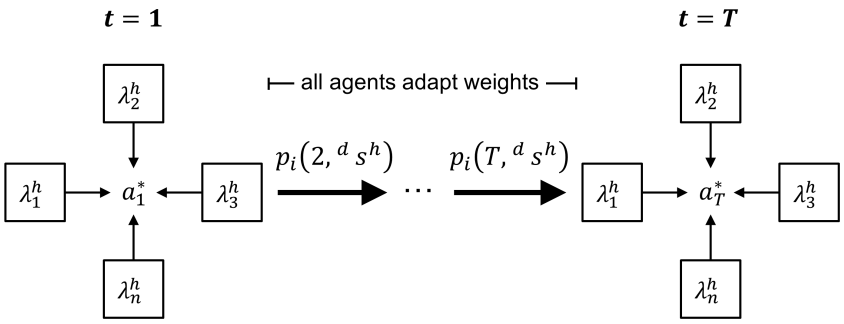

Figure 6.4: The ECP within a MEP.

The **state-action value function** of an ECP (i.e. Q^t) is defined with respect to the value function of the corresponding MEP (i.e. Q^h) in that local rewards of the ECP are added to future state-action values of the MEP. The only difference to the definition of agent weights in

MEPs is the coordination step dependent reward estimate. It is dependent on the joint action a_t^* of the MEP, i.e. the adapted agent weight $p_i(t, {}^d s^h)$ is evaluated with regards to the positive or negative impact agent λ_i has for joint action a_t^*.

Definition 6.12 (State-action value function in ECPs)**.**

$$Q_t(s_t^{ECP}, a_t^{i,ECP}) = R^{ECP}(t, s_t^{ECP}, a_t^{i,ECP}) + \sum_{s^{h+1} \in S^{h+1}} Tr^{ECP}(s^{h+1}|s^h, a^h) \max_{a^{h+1} \in A^{h+1}} Q^h(s^{h+1}, a^{h+1}) \tag{6.9}$$

Learning to act in a MEP with ECP entails learning a non-stationary agent-dependent policy $\pi_i : T \times S^{\text{ECP}} \to A^{\text{ECP}}$ for the ECP. The policy is non-stationary since it depends on the current coordination step. As an agent is only rewarded if the correct final joint action a_T^* has been chosen, the reward function is dependent on the coordination step as well.

In this work, budget C_i^{EXP} is agent-dependent and set to $C_i^{\text{EXP}} = 1 \; \forall \lambda_i \in \Lambda$, i.e. an agent is allowed to query its expert exactly once [3].

We start by introducing a novel approach to learning local expert weights for decentralized agents by exploiting the assumption that all experts have to be executed for the same decision candidate.

6.3 Expert Weight Learning for MEPs

In MEPs with $C_i^{\text{EXP}} = 1$ *and* perfect communication, we can exploit fixed-length feature spaces, as each expert is queried for the same decision candidates. As a consequence, any online supervised learning technique is applicable (such as GD, see Definition 3.18) to learn weights for our feature representation.

To this end, we use the set of all meta dependencies (see Section 5.3) as feature representation and assume a *linear relationship* among them. We thus deal with linear regression (see Definition 3.17) and standard squared loss $\mathbb{L}_{\text{SQUARED}}$, using GD to perform weight updates. For the MEP, we need to learn the agent-dependent model parameter vector ϕ_i for agent λ_i:

$$\text{p}_i^{\text{GD}}(1, {}^d s) = MD(e_i, {}^d s^h)^T \phi_i \tag{6.10}$$

To this end, multiple options for using *labels* exist. A straightforward way is using rewards $R({}^d s, e_i({}^d s))$ for a decision candidate and the expert of the agent, where future impacts of the action are only taken into account via inter-step meta dependencies. An alternative would be to use an estimate of the state-action value, which is approached by the well-known $Q-$Learning algorithm (extend with function approximation) [140].

With expert rewards as labels, we derive tuples $(MD(e_i, {}^d s^h), R({}^d s, e_i({}^d s)))$ to learn the model parameters, where deriving the gradients for $\mathbb{L}_{\text{SQUARED}}$ is straightforward. The result-

[3] Otherwise, agents are enabled to manipulate states to learn about their expert, which opens different research questions.

ing expert weight updates are situated between GD and SGD (see Definition 3.18) by using *every available* sample individually as done in GD, by not aiming to retrieve all possible samples (which would entail to execute all possible combinations of decision candidates for each expert) as approached by SGD. We thus do not incorporate iteration or epoch parameters, as available for GD and SGD, since we use each sample exactly once and the choice wit respect to state-expert pairs is controlled by the MEP.

Complexity Analysis 6.1. *Similar to our Online RL approach based on EWH (see Section 5.4), the complete set of defined meta dependencies is used as feature space for GD and the resulting computational complexity for a model update of a single agent is $O(|MD|)$, where $O(|MD|) = (|E^h|^2 + |E^h||E^{h+1}| + |E^h||E^{h-1}|)(|K||S|log|S|)$ as defined for the complexity of meta dependencies (see Section 5.3). Note that we assume perfect communication, where retrieving the predictions of all residual experts is not taken into account. If the latter is not optimized, an agent has to communicate with all agents of the prior, the current and the subsequent step, yielding a significant increase in complexity (querying the needed agents is in $O(|\Lambda| + |\Lambda^h| + |\Lambda^{h-1}|)$ for a single agent). The same holds for reusing expert weight learners proposed in Chapter 5 for MEPs.*

While this method presumes all experts to be executed for the same decision candidates, we will discuss a possible extension for regression models to the budgeted case in Section 6.8. Note that our SAS methods proposed in Chapter 5 already deal with this case, but using standard supervised learning techniques for multi-step expert advice is not yet fully covered.

We now present approaches to *coordinate* conflicting expert hypotheses, which also work without having learned expert weights as initial values. We start with passive coordination, where communication among agents is not allowed.

6.4 Passive Coordination with MEPs

Passive coordination requires agents to adapt their weights based on the taken joint action as well as the received reward. We therefore ignore coordination steps (i.e. $T = 1$), which improves scalability, but makes coordination difficult.

We aim to adapt agent weights for passive coordination, instantiated as $p_i^{PASSIVE} : T \times S \rightarrow \mathbb{R}$, by using the EWH algorithm (see Definition 3.31) in order to align current agent weights $_i$. More specifically, we define an additional *passive coordination weight* $w_i^{\text{PASSIVE}} : S \rightarrow \mathbb{R}$ for each agent, which is updated similar to EWH. The resulting agent weight is defined as:

$$p_i^{PASSIVE}(1, {}^d s_z^h) = p_i(1, {}^d s_z^h)\ w_i^{\text{PASSIVE}}(s_z) \tag{6.11}$$

For passive coordination in a MEP, each agent λ_i can only predict $\lambda_i({}^d s) = e_i({}^d s)$, i.e. the action proposed by its expert. We use the standard EWH update rule for w^{PASSIVE}, but only use the reward of the joint action as reward, i.e.:

$$w_i^{\text{PASSIVE}}(s_z+1) = w_i^{\text{PASSIVE}}(s_z)\ exp\big(R^{\text{SCALED}}(\lambda_i({}^d s_z))\ \eta^{\text{EWH}}\big) \tag{6.12}$$

To have agents passively converge to a consensus, we only update the weights if an agent's hypothesis was incorrectly chosen or incorrectly eliminated by the weighted majority voting [4], i.e.:

$$w_i^{\text{PASSIVE}}(s_{z+1}^h) = \begin{cases} \text{Equation 6.12}, & \text{if } R({}^d s, e_i({}^d s)) = 1 \wedge a^* \neq e_i({}^d s) \\ \text{Equation 6.12}, & \text{if } R({}^d s, e_i({}^d s)) = 0 \wedge a^* = e_i({}^d s) \\ w_i^{\text{PASSIVE}}(s_z^h) & \text{otherwise} \end{cases} \tag{6.13}$$

Complexity Analysis 6.2. *As passive coordination does not entail to learn models with respect to individual actions of agents, significant costs might result from retrieving the weighted majority voting (to update the passive coordination weights). Besides, we only keep and update an additional set of weights for C_{EXP} experts (in our special case only a single), resulting in a total complexity of $O(C_{EXP})$ for passive coordination only. The total cost of the resulting MEP is then also dependent on the calculation of agent weights p_i, which are not necessarily required.*

As adversarial learning approach, EWH enables robustness with respect to changing data distributions when paired with agent weight learning techniques (e.g. gradient descent on meta dependencies). Still, by taking into account the individual actions of other agents as well as allowing agents to respond to the latter, the expected cumulative reward of the MEP can be further improved.

6.5 Active Coordination with Heuristic Update Rules

For the *active coordination* protocol we allow multiple coordination steps (i.e. $T > 1$) and learn agent weight function $p_i^{ACTIVE} : T \times S \to \mathbb{R}$, defined analogously to p_i. If there is any disagreement for a decision candidate, all agents can alter their weights to have the committee converge to a consensus. Agents can thus either abandon their prediction or reinforce it.

Our first active coordination approach is a heuristic (which we name ACTIVEH), which exploits novel properties for meta dependencies and actions of agents dependent on coordination steps. The central intuition for the heuristic is that a *coordination weight update* for an agent λ_i should be be based on the decisions of other agents of the same step which are *confident* (i.e. dominant) for the current decision candidate. Here, a confident agent is taking the action its expert suggests (i.e. $\lambda_i({}^d s) = e_i({}^d s)$ holds), as the agent's weight is sufficiently high (i.e. $p_i^{\text{ACTIVE}}(t, {}^d s^h) > Q_t({}^d s^h, a_t^*)$ holds) or other confident agents' experts are predicting the action, such that the agent involuntarily predicts its expert's action. If not a single agent

[4] We assume binary rewards, but the strategy can easily be extended to continuous rewards settings by introducing a threshold.

is confident, the weighted majority based on the initial agent weights is chosen. The heuristic thus implements the basic idea of swarm intelligence, where agents both try to fight for their expert's hypothesis or propagate actions of other confident agents.

We use all meta dependency instantiations as introduced in Section 5.3 & Table 5.1,5.2,5.3. For ease of demonstration, we well use intuitive labels for their identifier l. Algorithm 2 defines our weight update strategy, where we omit expert-state parameters for meta dependencies as well as densities to increase readability, as the latter are constantly $(e_i, e_j, {}^d s^h)$. The algorithm takes an additional parameter $\theta^{\text{IMPACT}} \in \mathbb{R}^{\geq 0}$ which balances the impact of the weight updates.

Algorithm 2 Active Coordination with Heuristics

Require: λ_i, d, a_t^*, θ^{IMPACT}
Ensure: $p_i^{\text{ACTIVEH}}(t+1, {}^d s)$
1: $\Delta \leftarrow 0$
2: **for all** $\kappa \in K$ **do**
3: **for all** $\lambda_j \in \Lambda \setminus \lambda_i$ **do**
4: **if** $\text{HORIZON}(e_i) = \text{HORIZON}(e_j)$ **then**
5: **if** $e_i({}^d s) = e_j({}^d s) \wedge \lambda_j(t, {}^d s) = e_j({}^d s)$ **then**
6: $\Delta \leftarrow \Delta + \theta^{\kappa,\delta}_{ERROR} MD^{\kappa,\text{ERROR}}_{\text{PAIR}} \theta^{\kappa,\delta}_{JOINT} MD^{\kappa,\text{JOINT}}_{\text{PAIR}}$
7: **end if**
8: **if** $e_i({}^d s) \neq e_j({}^d s) \wedge \lambda_j(t, {}^d s) = e_j({}^d s)$ **then**
9: $\Delta \leftarrow \Delta + \theta^{\kappa,\delta}_{CONFLICT} MD^{\kappa,\text{CONFLICT}}_{\text{PAIR}} - \theta^{\kappa,\delta}_{ADDITIVE} MD^{\kappa,\text{ADDITIVE}}_{\text{PAIR}}$
10: **end if**
11: **end if**
12: **if** $e_j({}^d s') = d \wedge \lambda_j({}^d s')_t = e_j({}^d s')$ **then**
13: $\Delta \leftarrow \Delta + \theta^{\kappa,\delta}_{PAST} MD^{\kappa,\text{PAST}}_{\text{PAIR}}$
14: **end if**
15: **if** $\text{HORIZON}(e_i) = (\text{HORIZON}(e_j) - 1)$ **then**
16: $\Delta \leftarrow \Delta + \theta^{\kappa,\delta}_{FUTURE} MD^{\kappa,\text{FUTURE}}_{\text{PAIR}}$
17: **end if**
18: **end for**
19: **end for**
20: $p_i^{\text{ACTIVEH}}(t+1, {}^d s) \leftarrow p_i^{\text{ACTIVEH}}(t, {}^d s) + \theta^{\text{IMPACT}}\ \Delta$
21: **return** $p_i^{\text{ACTIVEH}}(t+1, {}^d s)$

As apparent in the algorithm, we only use pairwise meta dependencies to update the initial agent weight estimate since we only want to align and coordinate the latter. We will now explain important parts of the algorithm.

- **Line 2-3:** We iterate over all available kernels and agents.

- **Line 4-13:** All updates require the respective pairwise agent to be sufficiently confident in its expert hypothesis.
- **Line 4-6:** The joint performance of two experts of the same horizon is weighted by the independent error of the latter.
- **Line 8-9:** The weighted conflicting performances of two experts are subtracted to retrieve the relative performance of agent λ_i
- **Line 12-13:** If the current decision candidate was generated from the expert of the respective agent, we update the weight by the respective density-weighted meta dependency
- **Line 15-16:** For experts of the next horizon, the respective density-weighted meta dependencies are directly used.
- **Line 20:** The agent weight is updated by the manually weighted aggregate of all updates for all kernels.

Complexity Analysis 6.3. *Other than in passive coordination, we need to include complexities resulting from the novel decision process of length T. In the proposed heuristic, one recalculates agent weights in every coordination step $t \in 1,\dots,T$ by comparing their pairwise predictions. While calculations concerning agents of prior steps practically only need to be made once, pairwise comparisons for all agents of the same step have to be conducted for every coordination step. As a result, the complexity for all updates for a single decision candidate and single step (but all coordination steps) is within $O(|\Lambda^{h-1}|\lambda^h| + |\Lambda^h|^2 T)$. The total cost of the resulting MEP for a given episode additionally consists of the calculation of all meta dependencies and expert weights for all steps and respective decision candidates, i.e. $O(DH|\Lambda^h|(|\Lambda^{h-1}| + |\Lambda^{h+1}| + |\Lambda^h|T + |MD|))$ with D again the number of decision candidates for the state.*

Although our heuristic provides an intuitive way to update agent weights, it might suffer from changing data distributions as well as high variance in expert performances, as handcrafted, non-weighted rules are used. We thus continue with a learning-based approach for active coordination, which learns and update model parameters for agent-dependent meta dependencies.

6.6 Active Coordination with Reinforcement Learning

Until now, we ignored essential properties of the ECP for active coordination, as we did not account for future actions of agents and did not use knowledge about coordination step limit T. We now approach learning policies for MEPs by explicitly solving the ECP.

The central problem of solving the ECP is dealing with continuous action spaces. One solution would be to discretize the continuous actions by predefining possible agent weight increments, which lacks flexibility and yields highly correlated actions. We thus learn separate *expert coordination policies* for each agent with a Policy Search RL approach by parameterizing the policy and directly learning the action, i.e. by learning to adapt an agent's weight $p_i(t, {}^d s^h)$. To generalize learned policies, we use extend meta dependencies to compactly integrate multiple representations of the given decision candidate and actions predicted by other agents (similar to the heuristic). Finally, as the ECP does not require agents to find an explicit consensus (i.e. a correct joint action is maximally rewarded), we propose an additional heuristic to choose dominant actions.

Our approach for learning expert coordination policies comprises (i) defining coordination step dependent relational agent features (based on meta dependencies) as state representation, (ii) learning *relative agent error residuals* in the ECP with a Policy Search RL approach and (iii) choosing *robust actions* for the corresponding MEP after T coordination steps. We term the respective approach ACTIVEL.

6.6.1 Agent Meta Dependencies

We need novel state representations for agents to learn how to act in the ECP. We thus define *agent meta dependencies* $AMD(t, \lambda_i, {}^d s)$ for coordination step t, agent λ_i and decision candidate ${}^d s^h$, where corresponding meta dependencies $MD(e_i, {}^d s)$ are extended based on the state of the ECP. Similar to our active coordination heuristic, an agent meta dependency only consists of the value of the respective meta dependency if the respective agent is confident.

Single expert meta dependencies are directly used as agent meta dependencies, as they independently approximate the performance of λ_i:

$$AMD^{\kappa,\text{SINGLE}}_{\text{SINGLE}}(t, \lambda_i, {}^d s) = \text{MD}^{\kappa,1}_{\text{SINGLE}}(e_i, {}^d s) \tag{6.14}$$

The future impact of an agent is not conditioned any pairwise behavior, as inexact:

$$AMD^{\kappa,\text{FUTURE}}_{\text{PAIR}}(t, \lambda_i^h, \lambda_j^{h+1}, {}^d s^h) = \text{MD}^{\kappa,\text{FUTURE}}_{\text{PAIR}}(e_i^h, e_j^{h+1}, {}^d s^h) \tag{6.15}$$

All residual pairwise measures require that λ_j, i.e. the respective paired agent, *does not abandon its expert's prediction*. For independent error no further condition is needed:

$$AMD^{\kappa,\text{ERROR}}_{\text{PAIR}}(t, \lambda_i, \lambda_j, {}^d s) = \text{MD}^{\kappa,\text{ERROR}}_{\text{PAIR}}(e_i, e_j, {}^d s) \mathbb{1}_{\{\lambda_j(t, {}^d s) = e_j({}^d s)\}} \tag{6.16}$$

The past performance of e_i^h and e_j^{h-1} is conditioned on e_j^{h-1} agreeing with its expert and ${}^d s^h$ being its hypothesis.

$$AMD^{\kappa,\text{PAST}}_{\text{PAIR}}(t, \lambda_i, \lambda_j, {}^d s) = \text{MD}^{\kappa,\text{PAST}}_{\text{PAIR}}(e_i, e_j, {}^d s) \mathbb{1}_{\{\lambda_j(t, d^{h-1}) = e_j({}^d s^{h-1})\}} \mathbb{1}_{\{e_j(t, d^{h-1}) = {}^d s^h\}} \tag{6.17}$$

The joint performance of two experts λ_i, λ_j is additionally conditioned on two experts agreeing:

$$AMD^{\kappa,\text{JOINT}}_{\text{PAIR}}(t,\lambda_i,\lambda_j,{}^d s) = \text{MD}^{\kappa,\text{JOINT}}_{\text{PAIR}}(e_i,e_j,{}^d s)\mathbb{1}_{\{\lambda_j(t,{}^d s)=e_j({}^d s)\}}\mathbb{1}_{\{e_i({}^d s)=e_j({}^d s)\}} \tag{6.18}$$

Similarly, pairwise expert measures with regards to additive impact are only used if λ_i, λ_j disagree:

$$\begin{aligned}&\forall l \in \{\text{ADVANTAGE}, \text{ADDITIVE}\}: \\ &\quad AMD^{l,\kappa}_{\text{PAIR}}(t,\lambda_i,\lambda_j,{}^d s) = \text{MD}^{\kappa,l}_{\text{PAIR}}(e_i,e_j,{}^d s)\mathbb{1}_{\{\lambda^j(t,{}^d s^h)=e_j({}^d s^h)\}}\mathbb{1}_{\{e_i({}^d s^h)\neq e_j({}^d s^h)\}}\end{aligned} \tag{6.19}$$

We now define $AMD^{\kappa}(t,\lambda_i,{}^d s)$, the kernel-dependent agent meta dependency feature vector, for agent λ_i^h and decision candidate ${}^d s^h$ in coordination step t :

$$\begin{aligned}AMD^{\kappa}(t,\lambda_i^h,{}^d s) = \; & \langle AMD^{1,\kappa}_{\text{SINGLE}}(t,\lambda_i^h,{}^d s), \\ & \cup_{\forall l\in\{2,3,4,5\}\forall\lambda_j^h\in\Lambda^h\setminus\lambda_i^h} AMD^{l,\kappa}_{\text{PAIR}}(t,\lambda_i^h,\lambda_j^h,{}^d s^h), \\ & \cup_{\forall\lambda_j\in\Lambda^{h+1}} AMD^{6,\kappa}_{\text{PAIR}}(t,\lambda_i^h,\lambda_j^{h+1},{}^d s), \\ & \cup_{\forall\lambda_j\in\Lambda^{h-1}} AMD^{7,\kappa}_{\text{PAIR}}(t,\lambda_i^h,\lambda_j^{h-1},{}^d s)\ \rangle\end{aligned} \tag{6.20}$$

The agent meta dependency feature vector is finally defined as union over all available kernels, i.e.:

$$AMD(t,\lambda_i^h,{}^d s) = \cup_{\forall\kappa\in K} AMD^{\kappa}(t,\lambda_i^h,{}^d s) \tag{6.21}$$

Note that the agent meta dependency feature vector $AMD(t,\lambda_i^h,{}^d s)$ potentially changes after each coordination step, while $MD(e_i,{}^d s^h)$ is constant for ${}^d s^h$. We no, present our strategy to act in the ECP by adapting agent weight functions $p_i^{\text{ACTIVEL}}(t,{}^d s^h)$.

6.6.2 Policy Search RL for the ECP

As the ECP has a finite-horizon (i.e. T), we can learn T separate policy functions, which directly estimate the continuous action. More specifically, we learn T linear models based on agent features $AMD(t,\lambda_i^h,{}^d s)$, which are *additively* combined and driven towards *error residuals* with respect to the actions an agent takes. We therefore keep T copies of model parameters $\phi_{i,1}^{\text{ACTIVEL}},\ldots,\phi_{i,T}^{\text{ACTIVEL}}$ such that an agent weight at coordination step t is defined as:

$$p_i^{\text{ACTIVEL}}(t, {}^d s) = p_i^{\text{ACTIVEL}}(1, {}^d s) + \sum_{t' \in 2,\ldots,t} \gamma \, AMD(t, \lambda_i, {}^d s)^T \phi_i^{t'} \tag{6.22}$$

where $\gamma \in [0,1]$ is the discount factor hyperparameter, which we set to $\gamma = 1$, as the learned policy is non-stationary. Learning the resulting policy $\pi_i^{\text{ACTIVEL}}(t, {}^d s)$ thus reduces to gradient boosting [49] if all residual agents would not change their weights.

To cope with decision dynamics, we use the squared loss function $\mathbb{L}_{\text{SQUARED}}(\phi_{i,t}^{\text{ACTIVEL}}; AMD(t, \lambda_i^h, {}^d s), y_i^{t,{}^d s^h})$, as for linear regression. An agent-dependent label $y_i^{t,{}^d s^h}$ is then calculated as:

$$y_i^{t,{}^d s^h} = \begin{cases} \min(\text{DIST}_i^{t,{}^d s^h}, \phi_{i,t}^{\text{ACTIVEL}\,T} \phi_i^t) & \textbf{if} Q_t(s_t^{i,\text{ECP}}, a_t^{i,\text{ECP}}) < Q^h({}^d s^h, a_t^*) \\ \max(|\text{DIST}_i^{t,{}^d s^h}|, \phi_{i,t}^{\text{ACTIVEL}\,T} \phi_i^t) & \textbf{if} Q_t(s_t^{i,\text{ECP}}, a_t^{i,\text{ECP}}) > Q^h({}^d s^h, a_t^*) \\ 0 & \text{otherwise} \end{cases} \tag{6.23}$$

where $\text{DIST}_i^{t,{}^d s^h} = Q^h(s, a_t^*) - p_i(t, {}^d s^h) - \alpha$ with $\alpha \in \mathbb{R}$ the minimal distance to win or abandonment. Note that, to decide in which direction to update the agent policy, we use the value function of the ECP, Q_t, to assess the action of an agent and the value function of the MEP, Q^h, for the resulting joint action. When calculating the minimal distance value $\text{DIST}_i^{t,{}^d s^h}$ of the update, we only use the value function of the MEP, as the latter depicts the current average weight for the joint action. The label for the loss function states that we need to update ϕ_i^t in terms of the distance to choosing the correct action, if rewards for $\lambda_i(t, {}^d s^h)$ or $e_i({}^d s^h)$ with respect to a_t^* are not reflected in $p_i(t, {}^d s^h)$ or $Q^h(s, a_t^*)$, as it implies that λ_i should pursue its expert's action. Otherwise, no update occurs, as λ_i cannot actively improve the committee action.

To this end, we need to conduct GD updates for model parameters $\phi_{i,t}^{\text{ACTIVEL}}$, i.e.:

$$\phi_{i,t+1}^{\text{ACTIVEL}} \leftarrow \phi_{i,t}^{\text{ACTIVEL}} - \gamma \nabla_{\phi_{i,t}^{\text{ACTIVEL}}} \mathbb{L}_{\text{SQUARED}}(\phi_{i,t}^{\text{ACTIVEL}}; AMD(t, \lambda_i^h, {}^d s), y_i^{t,{}^d s^h}) \tag{6.24}$$

We will now discuss how agents derive their final decisions after T coordination steps.

6.6.3 Robust Agent Decisions

As agents have to generalize across decision candidates of different episodes, their predictions might be erroneous for unexplored regions, such that the ECP is prone not to converge within coordination steps $1, \ldots, T$. The problem is exacerbated for high-dimensional modalities, such as text or images, where generalization is difficult for small training sets. To make agent predictions more *robust*, we thus have agents choose the action which they most frequently voted for throughout T. We extend this heuristic with a measure of *dominance* with regards to the actions an agent chooses.

Definition 6.13 (Agent-specific dominance of an action)**.** *Given agent decisions in T coordination steps $\lambda_i(1,{}^d s^h),\ldots,\lambda_i(t,{}^d s^h)$, let $\sigma_i({}^d s^h,{}^d a^h) = |\{t|\exists t \in 1,\ldots,T : \lambda_i(t,{}^d s^h) = {}^d a^h\}|$ be the number of times λ_i chose a and $A_i^{{}^d s^h} = \{{}^d a^h|\exists t \in 1,\ldots,T : \lambda_i(t,{}^d s^h) = {}^d a^h\}$ the set of actions λ_i has chosen throughout $1,\ldots,T$.*

The agent-specific dominance of action ${}^d a^h$ for λ_i is defined as:

$$\Delta_i^{DOM}({}^d s^h,{}^d a^h) = \max\left(\Delta_i({}^d s^h,{}^d a^h),0\right) \tag{6.25}$$

where

$$\Delta_i(s^h,{}^d a^h) = \min\left(\{\Delta|\forall {}^c a^h \in A^i_{{}^d s^h} \setminus {}^d a^h : \sigma_i(s^h,{}^d a^h) - \sigma_i(s^h,{}^c a^h) - \Delta = 0\}\right) \tag{6.26}$$

holds.

$\Delta_i({}^d s^h,{}^d a^h)$ returns the minimal occurrence difference of action ${}^d a^h$ compared to the residual actions taken by λ_i. $\Delta_i^{\text{DOM}}({}^d s^h,{}^d a^h)$ only returns 0 if the minimal difference is negative, i.e. action ${}^d a^h$ is not dominating the residual actions. The resulting dominance measure for agents and action thus expresses the absolute dominance of an action based on all relative dominance values of an action compared to the residual ones.

Based on the dominance estimates, we now define a state action value function for each agent – $q_i : S^h \times A^h \to \mathbb{R}^{\geq 0}$ – which combines the average frequency of an action with its dominance. The agent-dependent state action value consists of the normalized choice frequency of a potentially penalized by minimal dominance $\Delta^{\text{MIN}} \in \mathbb{N}^{\geq 0}$ and impact factor $\mu \in \mathbb{R}^{\geq 0}$:

$$q_i(s^h,a) = \max\left(\frac{\sigma_i(s^h,a)}{|A^i_{s^h}|} - \mu\frac{\Delta_{\text{MIN}} - \Delta_i^{\text{DOM}}(s^h,a)}{\Delta^{\text{MIN}}},0\right) \tag{6.27}$$

The minimal dominance parameter is thus to be set according to the necessity of convergence or availability of a dominant action. A critical observation is that convergence might depend on the number of coordination steps T which – when set to low – might bias the dominance of an action.

Finally, the decision of an agent *after* T coordination steps $\lambda_i : S^h \to A^h$ is defined as action with maximal value:

$$\lambda_i({}^d s^h) = \max_{a \in A_i^{{}^d s^h}} q_i({}^d s^h,a) \tag{6.28}$$

Complexity Analysis 6.4. *As we approach active coordination in MEPs, the respective calculations are dependent on T coordination steps. Both, the complexity for conducting T active coordination updates for all agents as well as the overall complexity for the resulting MEP are the same as for the active coordination heuristic. More specifically, the set of agent meta*

dependencies requires the same pairwise agent comparisons as the active coordination heuristic. The GD updates do not add significant complexity for our lower bound, although gradient computations are becoming more complex with increasing number of agents or generally richer meta dependencies (i.e. higher numbers of available kernels or behavioral measures). Finally, the cost for the final agent decision with respect to dominance is based on pairwise agent comparisons of the current step. As a result, the total cost of the resulting MEP for a given episode, again, is within $O(DH|\Lambda^h|(|\Lambda^{h-1}|+|\Lambda^{h+1}|+|\Lambda^h|T+|MD|))$ *with D the number of decision candidates for the state. The coordination step dependent costs are the most significant and are directly reduced when communication among agents is optimized.*

6.7 Discussion

Given our simplifying assumption that each agent only uses the highest weighted action of its local decision, one might lose reciprocal effects. Our methods can be extended to *full coordination*, where an agent does not limit the available actions before coordination, but only weights them. The action space for the ECP would then comprise vectors with continuous weight adaptions for each available action. Calculating relational agent features then requires a larger number of expert comparisons, which is trivial to implement. One can apply our Policy Search RL approach as well as the robust action heuristic to all available actions individually, but there is open potential to exploit conflicting weight updates.

Our approaches assume that all experts are executed at all times, which might not be possible when large amounts of experts are available. This could be remedied by having a single *meta-agent* preselect experts (and agents) or by distributing budgets among agents, which requires novel approaches. In addition, we require an available bound T for coordination steps (i.e. the horizon for the coordination process is known). Other domains might require infinite-horizon approaches.

Finally note that we make the simplifying assumption that e_i represents the action choice of agent λ_i, whereas the chosen action might be proposed by various experts in E_i^h. Even more so, one could keep the complete set of expert predictions E_i^h for the coordination process and define agent function λ_i as well as agent weight function p_i to output a vector of all expert actions.

6.8 Summary

In this chapter, we formalized MEPs to study expert coordination for multi-step expert advice. A MEP has unique properties compared to prevalent decision-making frameworks and approaches crucial challenges, such as reducing biases due to expert correlation with the ECP.

To enable agents to assess their experts in the budget-free MEP setting, we explored online supervised learning, where we reuse meta dependencies as feature space for linear regression.

We approached the coordination problem in MEPs with three different approaches, assuming different degrees of available information. In the passive coordination setting, we reused the EWH algorithm to learn additional weights to adjust available expert confidences. When assuming perfect information for each agent, coordination for ECP can be learned, which we dealt with in two further approaches. The first active coordination method we introduced is based on a heuristic, which leverages pairwise meta dependencies as well as coordination step dependent agent predictions. For our learning-based active coordination approach, we defined agent meta dependencies by taking into account pairwise coordination step dependent agent decisions, which enable the application of function approximation to agent coordination for MEPs. Our expert coordination approach was based on gradient boosting and a robust action selection heuristic, which enabled to learn policies for the ECP as well as corresponding MEP.

Our work on MEPs and diverse coordination approaches addressed Research Question 3, where the corresponding Hypothesis 3 is partially confirmed and will be revisited after the empirical evaluation in Chapter 7.

Chapter 7

Evaluation of Learning with EPs & MEPs

This chapter deals with Research Question 1,2 and 3, which are concerned with developing a framework for multi-step tasks with expert advice as well as methods to solve the latter. These research questions have been partially approached by Chapter 5 and 6 by introducing the respective approaches based on Hypothesis 1, 2 and 3. The goal of this chapter is validating the respective contributions based on an empirical evaluation.
Our references for this chapter are [100, 104].

7.1 Introduction

We evaluate all introduced learning methods for finite multi-step tasks modeled as EP or MEP for task NERD, as introduced in Section 2.1. We thus refer to the challenges and contributions described in Chapter 5 & 6.

The resulting multi-step tasks as well as modeled decision processes have finite-horizon $H = 2$ with S^1 a state space over words, S^2 a state space over named entities and S^3 a state space over disambiguated named entities, i.e. resources available in a (semi-) structured knowledge base. For NER ($h = 1$), decision candidates are n-grams which we now define in general.

Definition 7.1 (n-grams)**.** *Given a piece of text of state s – $TEXT_s$ – defined as ordered set of tokens $TEXT_s^i$ where i denotes the position of the token, we define the set of n-grams for $TEXT_s$ as all possible sequences $TEXT_s^i, \ldots, TEXT_s^{i+n}$.*

For a piece of text $TEXT_s$, all possible n-grams with $n = 1, \ldots, |Text_s|$ make up the possible decision candidates for which the state-expert distribution is construed. After having queried NER experts of set E^1, we know the actual number of possible decision candidates, as determined by their hypotheses. For NED ($h = 2$), the set of decision candidates consists of all named entities proposed in $h = 1$ and the set of all states is constructed based on all valid combinations of decision candidates. Finally, decision candidates resulting from the NED step ($h = 3$) are disambiguated named entities proposed by queried NED experts of set E^2.

The chapter proceeds as follows: We first describe the setup of our evaluation (Section 7.2), where we explain all parameters, instantiations of our methods as well as baseline implementations. We then present and discuss the evaluation results (Section 7.3) and finally summarize the chapter in Section 7.5.

Kernel	Implementation
Candidate Lingual Type	POS tag
Candidate Word embeddings	Pretrained on articles
Candidate MinHash	Jaccard similarity
State Length	Number of characters
State Extra Characters	Count #, @
State Lingual Type	Average POS tag

Table 7.1: Used kernels for evaluation.

7.2 Setup

Our experimental setup description is divided into data sets, meta dependencies as well as their configurations, Web services used as experts and, finally, the evaluation measures with respective baseline- and weight learning approaches.

7.2.1 Data

The data sets we use are (i) Microposts 2014 corpus [10] which consists of a collection of tweets ($Z = 2034$) of no specific topic and (ii) Spotlight corpus [91] which consists of news articles of no specific topic, each consisting of several sentences ($Z = 59$), where Z corresponds to the number of resulting episodes for EPs or MEPs (i.e. the number of tweets or sentences, respectively). While Microposts 2014 comprises an average of 1.84 entities per tweet, Spotlight is denser with an average of 5.69 entities per sentence. Also, the tweets available in Microposts 2014 contain significantly more grammatical errors, abbreviations and extra characters such as hashes, symbols or hyperlinks than news articles in the Spotlight corpus. For modeling EPs and MEPs, we treat each tweet and each sentence within an article as single state sample, i.e. as piece of text $TEXT_s$ as defined for n-grams. Decision candidates thus comprise tokens or token sequences.

7.2.2 Meta Dependencies

The central variables for meta dependencies are the used kernels to calculate similarities among states and/or decision candidates. After presenting the kernels we use, we list the chosen values for density parameters M_κ and δ_κ.
Kernels: We summarize the kernel set K we used in Table 7.1, where *state* or *candidate* indicates whether the kernel is calculated for the complete sentence (or tweet), or for the decision candidate, respectively.

The **decision candidate kernels** assess a token or token sequence in terms of different perspectives. The *lingual kernel* is based on part-of-speech (POS) tags, where each possible

tag depicts a category and the kernel function simply compares the respective categories in a binary fashion. To directly compare tokens or token sequences we employ two different kernels. The *word embedding kernel* is used to generalize the comparison of texts of decision candidates based on the contexts the latter often occur. The embeddings were trained on large amounts of news articles on the Web [1] and are directly reused for our evaluation. In contrast, the *MinHash kernel* [2] compares the respective pieces of texts based on character overlap. Using the latter, we can better scale the computation of the kernel.

The **state-dependent kernels** are relatively general and represent the context of a decision candidate. The *length of a state* is defined as total amount of characters within a tweet or sentence in a news article and thus models the length of available context for a decision candidate. Both *extra characters* as well as *lingual types* then refine the assessment of the context, which might heavily influence the respective experts in its ability to choose among candidate hypotheses.

Densities: For calculating the densities, we need to set parameter M_κ which expresses the expected amount of decision candidates in the kernel-induced neighborhood. The latter is dependent on parameter δ_κ, defining the similarity threshold for a decision candidate to become part of a neighborhood. We vary M_κ and δ_κ for the kernels as follows, where Z corresponds to the number of episodes in a fold (see Definition 3.23).

- $M_{\text{length}} = M_{\text{extra}} = M_{\text{state lingual}} = \frac{Z}{4}$
- $\delta_{\text{length}} = \delta_{\text{extra}} = \delta_{\text{state lingual}} = 0.7$
- $M_{\text{embeddings}} = M_{\text{minhash}} = M_{\text{candidate lingual}} = \frac{Z}{10}$
- $\delta_{\text{embeddings}} = \delta_{\text{minhash}} = \delta_{\text{candidate lingual}} = 0.5$

Kernels for states get updated more often, as they either are relatively general or consist of the average of all occurring candidates, and thus get assigned higher thresholds with regards to the required similarity value as well as the number of states in the neighborhood. In contrast, the kernels for decision candidates are more specific and thus might never return any valuable neighborhoods if high thresholds are used, as the used training sets are relatively small. We thus choose lower thresholds for both parameters.

7.2.3 Experts

We exclusively use experts available as Web services and do not tune any of their respective parameters. The selection process was based on NERD benchmark GERBIL, where we chose pure NER approaches and pure NED approaches, i.e. we did not use Web services which

[1] `https://code.google.com/archive/p/word2vec/` (*accessed on 05/01/2018*)
[2] `https://github.com/ekzhu/datasketch/` (*accessed on 05/01/2018*)

NER Expert	NED Expert
Stanford Tagger (ST) [45]	AIDA (AIDT) [56]
Spotlight Spotter (SPS) [91]	Spotlight Tagger (SPT) [91]
FOX [119]	AGDISTIS (AGDT) [130]

Table 7.2: Experts for NER & NED.

	Microposts '14			Spotlight		
Appr.	F1	Prec	Rec	F1	Prec	Rec
ST	0.49	0.89	0.34	0.22	**0.86**	0.13
FOX	0.49	**0.91**	0.33	0.22	0.84	0.13
SPS	**0.56**	0.4	**0.89**	**0.64**	0.53	**0.81**
AGDT	0.45	0.6	0.36	0.35	0.7	0.24
AIDAT	0.32	0.45	0.25	0.24	0.29	0.2
SPT	**0.65**	**0.74**	**0.58**	**0.76**	**0.79**	**0.72**

Table 7.3: Baseline performances of experts.

solve the complete NERD task without allowing to input texts annotated with named entities for NED. Table 7.2 summarizes the experts we used for NER and NED, where the horizontal alignment expresses how the experts are actually used in combination in the real-world.

Table 7.3 shows the baseline *precision*, *recall* and *F1-measure* outcomes for the chosen data sets, which are the proposed evaluation metrics by GERBIL and defined in Section 3.2.5. Experts SPS for NER and SPT for NED provide the highest recall for tweets and news articles. SPS does, however, yield relatively low precision scores, resulting in numerous wrongly identified named entities. Note that FOX and ST strongly correlate, as FOX is an *ensemble learning* approach using ST. Central learning challenges thus are to detect correlations of FOX and ST as well as learning for which data characteristics SPS is correct.

7.2.4 Evaluation measures

We evaluate our approach in terms of two target measures: a) accuracy estimation of experts and b) outcome optimization of the EP or MEP. For each target measure, we perform 10-fold cross-validation and use precision, recall and F1-measure (as proposed by GERBIL) as metrics.

a) Accuracy Estimation:

The goal is to accurately estimate weights (i.e. w_{e^h}) for all experts and thus to predict if the suggested action of an expert is correct or not. Here, we are not in the full EP or MEP setting

and thus do not deal with budget constraints, i.e. we execute all experts once for a single state.

We define the following properties for all performance metrics:A *true positive* is correctly predicting that an expert's hypothesis is correct, a *true negative* is correctly predicting that an expert's hypothesis is wrong, a *false positive* is a wrongly predicting that an expert's hypothesis is correct and a *false negative* is wrongly predicting that an expert's hypothesis is wrong.

We now describe all baselines as well as instantiations of our approaches. Note that we do not use MEP coordination approaches, as they adapt the expert weights to optimize the weighted majority vote.

Gl-Avg: The first baseline is the straightforward global performance weighting for experts on *left-out* training data, similarly to [113]. The expert weight is calculated as precision of an expert on the left-out training data of the respective fold.

Single-Avg: The second baseline is based on single-expert meta dependencies as defined in Section 5.3.1. Based on all single-expert meta dependencies for a decision candidate ${}^{d}s$ for expert e, i.e. $MD^{\kappa,l}_{\text{SINGLE}}(e,{}^{d}s)$ (with κ the kernel, l the instantiation for behavioral measure and condition and θ the density), we compute the weighted average with densities as weighting factor:

$$w_e = \frac{\sum_{MD^{\kappa,\text{PRECISION}}_{\text{SINGLE}}(e,{}^{d}s)\in MD_{\text{SINGLE}}(e,{}^{d}s)} \theta^{\kappa,\delta}_{\text{PRECISION}}({}^{d}s^{h}) MD^{\kappa,\text{PRECISION}}_{\text{SINGLE}}}{\sum_{MD^{\kappa,\text{PRECISION}}_{\text{SINGLE}}(e,{}^{d}s)\in MD_{\text{SINGLE}}(e,{}^{d}s)} \theta^{\kappa,\delta}_{\text{PRECISION}}({}^{d}s^{h})} \tag{7.1}$$

OnGD-Single: An instantiation of our online learning approach (see Section 6.3), tagged OnGD-Single, where we use GD with all meta dependencies to learn expert weights. Here, we use the online learning framework *Vowpal Wabbit* [3] [81] with standard GD configuration as learner. The parameters are set as follows: $\eta = 0.2$.

OnGD-All: An instantiation of our online learning approach (see Section 6.3), tagged OnGD-All, where we use the same GD configuration as for OnGD-Single with all meta dependencies to learn expert weights. The parameters are set as follows: $\eta = 0.2$.

OnEWH-All: An instantiation of our online model-free RL approach (see Section 5.4), tagged OnEWH-All, where we extend the EWH algorithm to deal with expert budgets with all meta dependencies to learn expert weights. Note that we only train on tweet states, even when predicting for news article states. The parameters are set as follows: $p_{min} = 0.05, \eta = 0.8, \beta = 1.5$.

BatchPSL-All: An instantiation of our batch model-free RL approach (see Section 5.5), tagged BatchPSL, where we rely on an existing implementation of PSL [4] to integrate our model. Note that, again, we only train on tweet states, even when predicting for news article states. The parameters are set as follows: $Z_{POOL} = Z_{BATCH} = \frac{Z}{10}$.

b) Outcome optimization:

[3] `https://github.com/JohnLangford/vowpal_wabbit/` (*accessed on 05/01/2018*)

[4] `https://github.com/linqs/psl` (*accessed on 05/01/2018*)

Our second goal is to optimize the overall outcome of the EP or MEP, i.e. to maximize the expected cumulative reward.

We define the following properties for all performance metrics: A *true positive* is a correctly found (disambiguated-) named entity, a *true negative* is the correctly predicted abstinence of a (disambiguated-) named entity, a *false positive* is a wrongly found (disambiguated-) named entity and a *false negative* is the wrongly predicted abstinence of a named entity.

We again describe all baselines as well as instantiations of our approaches. The general budget parameters – if not defined otherwise – are $B^0_{\text{QUERY}} = |E^0|$ and $B^1_{\text{QUERY}} = |E^1|$.

AIDA: The NERD approach as deployed in the real-world with $e^1 = \text{ST}$, $e^2 = \text{AIDAT}$.

DBS: The NERD approach as deployed in the real-world with $e^1 = \text{SPS}$, $e^2 = \text{SPT}$.

AGD: The NERD approach as deployed in the real-world with $e^1 = \text{FOX}$, $e^2 = \text{AGDT}$.

EP-Gl-Avg: The first EP baseline, where we use Gl-Avg as described before, now integrated into an EP.

EP-Loc-Avg: The second EP baseline, where we use Loc-Avg as described bfore, now integrated into an EP.

EP-OnGD-All: Our online learning approach OnGD-All now integrated into an EP.

EP-OnEWH-All: Our online model-free RL approach OnEWH-All now integrated into an EP. The additional parameters for expert execution budgets are $B^0_{\text{QUERY}} = 2$ and $B^1_{\text{QUERY}} = 4$, i.e. two (state,expert) pairs can be queried for NER and four (state,expert) pairs for NED.

EP-BatchPSL-All Our batch model-free RL approach BatchPSL-All now integrated into an EP. The additional parameters for expert execution budgets are $B^0_{\text{QUERY}} = 2$ and $B^1_{\text{QUERY}} = 4$, i.e. two (state,expert) pairs can be queried for NER and four (state,expert) pairs for NED.

MEP-PCoord-Free: Our passive coordination approach integrated into a MEP with no expert weight estimation technique. The parameters are set as follows: $\eta^{\text{EWH}} = 1.0$

MEP-PCoord-OnGD-All: Our passive coordination approach integrated into a MEP with OnGD-All as expert weight estimation technique. The parameters are set as follows: $\eta^{\text{EWH}} = 1.0$

MEP-ACoordH-Free: Our active coordination approach integrated into a MEP with no expert weight estimation technique. The parameters are set as follows: $\theta^{\text{IMPACT}} = 1.0, T = 10$

MEP-ACoordH-OnGD-All: Our active coordination approach integrated into a MEP with OnGD-All as expert weight estimation technique. The parameters are set as follows: $\theta^{\text{IMPACT}} = 0.1, T = 10$

MEP-ACoordL-Single-Free-All: Our active coordination approach *without gradient boosting* and no initial confidence. Here, one single learner is invoked and updated T times. The parameters are set as follows: $T = 7, \alpha = 0.5, \mu = 0.3, \Delta_{\text{MIN}} = 3, \eta^{\text{GD}} = 0.4$.

MEP-ACoordL-Single-Conf-All: Our active coordination approach *without gradient boosting* and expert confidence based on the provided Web service. Here again, one single learner is invoked and updated T times. The parameters are set as follows: $T = 7, \alpha = 0.5, \mu = 0.3, \Delta_{\text{MIN}} = 3, \eta^{\text{GD}} = 0.4$.

	Microposts '14			Spotlight		
Appr.	F1	Prec	Rec	F1	Prec	Rec
Gl-Avg	0.71	0.75	0.67	0.7	0.69	0.72
Loc-Avg	0.76	0.65	**0.93**	0.78	0.66	**0.94**
OnGD-Local	0.84	0.82	0.86	0.79	0.95	0.69
OnGD-All	**0.85**	0.78	**0.93**	**0.86**	**0.96**	0.78
OnEWH-All	**0.85**	**0.84**	0.87	0.82	0.84	0.8
BatchPSL-All	0.82	0.8	0.85	0.79	0.78	0.81

Table 7.4: Evaluation results for a) accuracy estimation with $h = 1$.

MEP-ACoordL-Grad-Free-All-1: Our active coordination approach with gradient boosting. The parameters are set as follows: $T = 7, \alpha = 0.5, \mu = 0.3, \Delta_{\text{MIN}} = 3, \eta^{\text{GD}} = 0.4$.

MEP-ACoordL-Grad-Free-All-2: Our active coordination approach with gradient boosting. The parameters are set as follows: $T = 14, \alpha = 0.5, \mu = 0.3, \Delta_{\text{MIN}} = 6, \eta^{\text{GD}} = 0.4$.

MEP-ACoordL-Grad-Free-All-3: Our active coordination approach with gradient boosting. The parameters are set as follows: $T = 14, \alpha = 1.0, \mu = 0.05, \Delta_{\text{MIN}} = 6, \eta^{\text{GD}} = 0.4$.

MEP-ACoordL-Grad-Conf-All-1: Version of MEP-ACoordL-Grad-Free-All-1 with expert confidence based on the provided Web service.

MEP-ACoordL-Grad-Conf-All-2: Version of MEP-ACoordL-Grad-Free-All-2 with expert confidence based on the provided Web service.

MEP-ACoordL-Grad-Conf-All-3: Version of MEP-ACoordL-Grad-Free-All-3 with expert confidence based on the provided Web service.

Based on our evaluation setup, we now present and discuss the respective results.

7.3 Results

We first summarize the evaluation results for a) accuracy estimation for $h = 1$ and $h = 2$ (see Table 7.4 & 7.5) and b) outcome optimization (see Table 7.6).

7.3.1 Summary of Results for Accuracy Estimation

NER: For tweets, our methods based on the EWH algorithm (OnEWH-All) and pure GD (OnGD-All) with all meta dependencies reach the best overall results in terms of F1-measure as well as precision and recall. For news articles, OnGD-All is dominating for all performance metrics. To this end, it is noticeable that using all meta dependencies for OnGD-All improves the results with respect to GD-Local (using only single-expert meta dependencies) for both data sets. Our SRL approach (BatchPSL-All) provides stable results close to prior methods and improves the results with respect to the pure majority voting (Gl-Avg) as well as the

	Microposts '14			Spotlight		
Appr.	F1	Prec	Rec	F1	Prec	Rec
Gl-Avg	0.56	0.63	0.51	0.42	0.39	0.46
Loc-Avg	0.71	0.6	0.87	0.56	0.44	0.78
OnGD-Local	0.64	0.74	0.57	0.44	0.64	0.33
OnGD-All	**0.85**	**0.8**	**0.92**	**0.78**	**0.74**	**0.83**
OnEWH-All	0.73	0.67	0.81	0.67	0.62	0.72
BatchPSL-All	0.8	0.73	0.88	0.74	0.72	0.77

Table 7.5: Evaluation results for a) accuracy estimation with $h = 2$.

	Microposts '14			Spotlight		
Appr.	F1	Prec	Rec	F1	Prec	Rec
DBS	0.33	0.24	**0.51**	0.45	0.38	0.57
AGD	0.28	0.59	0.19	0.13	0.56	0.07
AIDA	0.33	0.69	0.21	0.24	0.69	0.15
EP-Gl-Avg	0.41	0.51	0.34	0.36	0.48	0.29
EP-Loc-Avg	0.47	0.59	0.38	0.42	0.49	0.37
EP-OnGD-All	0.44	0.71	0.33	0.35	0.58	0.25
EP-OnEWH-All	0.54	0.6	0.5	0.43	0.44	0.42
EP-BatchPSL-All	**0.56**	0.72	0.46	0.48	0.64	0.39
MEP-PCoord-Free	0.35	0.69	0.24	0.25	**0.88**	0.15
MEP-PCoord-OnGD-All	0.46	0.67	0.34	0.48	0.69	0.36
MEP-ACoordH-Free	0.41	0.72	0.29	0.44	0.48	0.41
MEP-ACoordH-OnGD-All	0.5	**0.79**	0.37	**0.63**	0.7	0.57
MEP-ACoordL-Single-Free-All	0.37	0.52	0.29	0.44	0.49	0.39
MEP-ACoordL-Single-Conf-All	0.32	0.49	0.24	0.39	0.56	0.31
MEP-ACoordL-Grad-Free-All-1	0.48	0.61	0.41	0.58	0.6	0.57
MEP-ACoordL-Grad-Conf-All-1	0.42	0.53	0.34	0.54	0.56	0.52
MEP-ACoordL-Grad-Free-All-2	0.51	0.65	0.42	0.51	0.59	0.47
MEP-ACoordL-Grad-Conf-All-2	0.43	0.55	0.34	0.6	0.61	**0.59**
MEP-ACoordL-Grad-Free-All-3	0.41	0.6	0.31	0.52	0.56	0.48
MEP-ACoordL-Grad-Conf-All-3	0.45	0.59	0.37	0.51	0.54	0.48

Table 7.6: Evaluation results for b) outcome optimization

weighted majority voting with single expert meta dependencies (Loc-Avg) with no further weightings of the respective kernels.

NED: For both tweets and news articles, our method based on pure GD (OnGD-All) with all meta dependencies reaches the best overall results with regards to all measures. However, the relational learner BatchPSL-All now supersedes the residual approaches for both data sets and all measures. OnEWH-All (i.e. our EWH extension) still dominates the baseline methods Gl-Avg and Loc-Avg. Notice, however, that GD-Loc performs worse than the latter – especially compared to OnGD-All.

7.3.2 Summary of Results for Outcome Optimization

The monolithic baselines DBS, AGD and AIDA reach close results for tweets in terms of F1-measure, but significantly differ in precision and recall, as was observable for the single expert performances shown in Table 7.3. Here, AGD and AIDA reach relatively high precision but low recall and the results of DBS show low precision and relatively high recall. For news articles, the results for precision are closer, but DBS still performs worse. For recall, DBS outperforms AGD and AIDA even clearer than before.

While the non-weighted majority voting (EP-Gl-Avg) can already improve the F1-measure for tweets compared to prior baselines, it does only reach an average recall with respect to DBS. For news articles, DBS still provides superior results. The same conditions hold for the weighted majority voting based on single-expert meta dependencies (EP-Loc-Avg), where further small improvements are observed, but recalls stay average and DBS wins for news articles. To this end, our approach based on GD and all meta dependencies (EP-OnGD-All) cannot improve the results and – while yielding higher precision values for both tweets and news articles than prior non-monolithic baselines – reaches lower recall values and, in turn, F1-measures.

Our learning methods for EPs – EP-OnEWH-All and EP-BatchPSL-All – further improve the results for all performance metrics on tweets as well as news articles. Especially for tweets, precision values are considerably improved by EP-BatchPSL-All (and similarly for EP-OnEWH-All while remaining behind the latter). The same holds for recall, where EP-OnEWH-All provides the best overall results for non-monolithic recall, while only performing slightly worse than DBS. For news articles, where EP-BatchPSL-All and EP-OnEWH-All were only trained on tweets, the results are less expressive but the same trends as for tweets can be observed. EP-BatchPSL-All can further improve precision (in contrast to EP-OnEWH-All which only yields average precision improvements compared to DBS) but recall values for the latter and EP-OnEWH-All are only average improvements over prior non-monolithic methods.

Starting with coordination approaches, passive coordination as introduced in Section 6.4 was not able to improve the F1-measure of the non-weighted majority vote for tweets (MEP-PCoord-Free), as only precision was raised but recall decreased. Similar conditions are prevalent for MEP-PCoord-OnGD-All where, however, small improvements in F1-measure

are observable, as recall was improved, while precision remained stable. While for MEP-PCoord-Free the results are the same for news articles, MEP-PCoord-OnGD-All considerably improves the results of MEP-OnGD-All for all performance metrics.

The results for our active coordination heuristic (see Section 6.5) show the same tendencies as for passive coordination, while the individual outcomes for each performance metric are better. For both tweets and news articles without available expert weights (MEP-ACoordH-Free), the results stay stable with respect to MEP-Gl-Avg, where precision increases but recall slightly drops. With GD and all meta dependencies as expert weights (MEP-ACoord-OnGD-All), we can significantly increase F1-measure and precision for both tweets and news articles. Here, the highest overall precision is reached for both data sets and respective recall values increase with respect to MEP-OnGD-All.

Finally, observing the outcomes of our learning-based active coordination approach (see Section 6.6), we first discuss results of the single learner (i.e. not taking advantage of having a finite-horizon). Here, for MEP-ACoordL-Single-Free-All without expert weights as well as for MEP-ACoordL-Single-Conf-All with using Web service confidences only below-average outcomes for all performance metrics and both data sets are achieved. The results significantly improve by exploiting gradient boosting with individual models for individual coordination steps for all instantiated approaches (MEP-ACoordL-Grad-Free-All-x, MEP-ACoordL-Grad-Conf-All-x with $x = 1,2,3$) and both data sets. The performance of MEP-ACoordL-Grad-Free-All-2 is most notable, as the F1-measure for tweets improves on the results of MEP-ACoordH-OnGD-All by yielding significantly higher recall without using initial expert weights. For news articles, the results for using initial expert weights (e.g. MEP-ACoordL-Grad-Conf-All-2) are competitive by yielding the highest overall recall of all methods, while achieving adequate results for precision.

7.4 Discussion

We first deal with our EP approaches and then interpret the general behavior of meta dependencies, as observable throughout all learning methods. We subsequently discuss the performances of our MEP techniques, which include coordination.

7.4.1 (EP-)OnGD

For our online learning approach based on GD, we distinguished among OnGD-Local with single-expert meta dependencies and OnGD-All with all meta dependencies. OnGD-All did yield superior performances for expert accuracy estimation on both data sets. The latter can be explained by the ability to learn a joint model over all experts while not having to account for missing features caused by expert budgets. The superior performance of OnGD-All for accuracy estimation is partially based on the performances of the individual experts. Both ST and FOX are relatively precise, but leave out several true positives (see Table 7.3), which makes

them easier to assess. As SPS is imprecise, it is significantly harder to find true positives. For outcome optimization, the performance of EP-WM-GD-All is only average with respect to F1-measure and recall, with relatively high precision.

7.4.2 (EP-)OnEWH-All

Other than OnGD, our online learner OnEWH-All is based on the EWH algorithm extended with measures to deal with partial information. Its accuracy estimation results did show stable improvements for all cases, where it especially performed well for NER and tweets. Even more so, did reach better results for OnEWH-All for most cases. While OnGD-All still dominated OnEWH-All for NER, the results for outcome optimization were different. EP-OnEWH-All did yield considerable improvements over the non-weighted- as well as weighted majority votes (EP-Gl-Avg and EP-Loc-Avg). Still, compared to the relational learner (EP-BatchPSL-All), EP-OnEWH-All performed slightly worse although the individual expert accuracies were dominant on average. The combined expert weights thus incorporated a bias, which might be due to expert correlation.

7.4.3 (EP-)BatchPSL-All

Our SRL approach (BatchPSL-All) exploits the relational nature of meta dependencies and is sufficiently flexible to deal with non-queried experts. The resulting performances for accuracy estimation did yield improvements over all baselines and remained competitive for both NER and NED as well as tweets and news articles. As became apparent for outcome optimization, EP-BatchPSL-All – other than online learners EP-OnGD-All and EP-OnEWH-All – also learned to disclose true positives by achieving good results for precision and recall. EP-BatchPSL-All thus yields the best overall F1-measure, thereby dominating all residual approaches on average. As the SRL approach exploits mini-batches of states for training as well as pools for prediction, it eventually better generalizes the performances of experts for available states.

7.4.4 Qualitative Results of Meta Dependencies

The learned meta-weights show that single- as well as pairwise intra-step expert meta dependencies both drive weights towards one (i.e. positive rules received high weights), while pairwise inter-step expert meta dependencies have the opposite effect (i.e. positive rules either received low weights or negative rules received high weights). The meta dependencies thus complement each other to provide well-balanced estimates. Although kernels for state text length and -extra characters are heuristics, they improve the predictive power. As both the PSL model as well as the online learners are easily extensible and scale well, adaptation to new tasks or domains is straightforward. As the used textual kernels were most influential, additional textual kernels, covering more decision candidates, might substantially improve

the predictions. Our approaches thus also enable to exploit numerous *embedded* text representations at once. Finally, our results suggest that meta dependencies learned on tweets also improve learning expert weights for news articles, making knowledge transfer possible.

Until now, we only dealt with EP results based on expert weight learning methods for SAS. Based on the latter, we now discuss the results of our passive- as well as active coordination techniques.

7.4.5 PCoord

In passive coordination, we used the EWH algorithm to adapt expert weights according to the joint performance of respective agents, without using information about the action taken by other agents. Only MEP-PCoord-OnGD-All, our approach using GD with all meta dependencies, was able to generate considerable improvements with respect to EP-OnGD-All. The technique is thus not sufficiently expressive to deal with the non-weighted case (MEP-PCoord-Free). However, when initial expert weights are available, the method can stabilize the dececentralized majority voting of agents.

7.4.6 ACoordH

Other than in passive coordination, our active learning heuristic used agent meta dependencies to gradually align agent weights throughout the ECP. Our results suggested that active coordination can significantly improve both non-weighted as well as weighted majority votings, where in the weighted case (MEP-ACoordH-OnGD-All) the best precision- and F1-measure scores are reached for tweets and news articles. As using active coordination with respectively adapted meta dependencies dominates passive coordination, our results suggest that the latter are adequate for the coordination task. The outcomes for recall were successfully optimized for news articles, but our method was not able to achieve the same goal for tweets. As the approach is a heuristic, it cannot dynamically adapt its weight updates for states with heterogeneous characteristics, which is potentially more important for tweets, as the latter are generally shorter and contain larger degrees of colloquial languages. Finally, the choice of coordination step limit T did not significantly impact our results if $T > 5$ was chosen. In most cases, if no consensus was found after 5 coordination steps, the results remained the same.

7.4.7 ACoordL

In our learning approach for active coordination, we use a Policy-based RL approach to coordinate expert weights in the ECP. Available initial expert weights are constrained to uniform weights (as used in MEP-ACoordH-Free) as well as using available expert service confidences (not available for several experts, namely AGDT, AIDT, ST and FOX). The evaluation thus emphasizes the performances of non-weighted or *partially* weighted coordination techniques

(in contrast to MEP-ACoordH-OnGD-All). As the single learning instantiation without gradient boosting is only used as baseline (MEP-ACoordL-Single-x-All) with $x = \{\text{Free,Conf}\}$) and provides average performance, we do not deal with it here.

The centralized approach EP-BatchPSL-All still outperforms all configuration for tweets in terms of all performance metrics, but the coordination approach (MEP-ACoordL-Grad-Conf-All-2) dominates the latter for news articles and competes with the active coordination heuristic with GD expert weights (MEP-ACoordH-OnGD-All). Note that, while the approach deals with decentralized coordination, it was able to achieve the highest recall for all available approaches throughout the evaluation. The active coordination learner is thus better able to adapt uniform initial weights and can also exploit slightly adapted initial weights to derive adequate agent weights for the weighted majority voting.

On average, the best results were reached by MEP-ACoordL-Grad-x-All-2 (with $x = \{\text{Free,Conf}\}$) where the central difference to MEP-ACoordL-Grad-x-All-1 is having more coordination steps ($T = 14$ versus $T = 7$) and the central difference to MEP-ACoordL-Grad-x-All-3 is a lower distance to the majority voting ($\alpha = 0.5$ versus $\alpha = 1.0$) as well as higher non-dominance penalization ($\mu = 0.3$ versus $\mu = 0.05$). While running our experiments we also made the empirical observation that reaching stable joint actions over all coordination steps decreased for early episodes if initial expert confidences were available and increased after the models started to generalize. If no expert confidences were given, reaching stable joint actions converged faster. With respect to parameter tuning, we suggest to (i) test larger values for T (ii) keep distance α between agent weights low, (iii) try out stronger non-dominance penalization and (iv) potentially test other initial expert weights. Those findings might help to reproduce the results and open up questions for more empirical and theoretical research.

7.4.8 Design Choices

Our central design choice (other than parameters) for our experiments are the used kernels. As we chose straightforward measures (e.g. raw text- or embedded vector similarities), the approaches are transferable to other NLP tasks. For other tasks, e.g. in image processing, one needs to use or develop other kernel functions. Further design choices for our experiments include the available experts for NER and NED steps as well as the number of agents. The selected experts exhibit central problems for multi-step tasks, namely expert correlation as well as yielding high precision with low recall or vice versa. While the choice of *one-to-one* mappings from agents to experts results in all experts being queried, the practical complexity is constrained by expert budget $C^{\text{EXP}} = 1$, where only a single expert-state configuration can be chosen.

7.5 Summary

In this chapter, we evaluated the suitability of EPs and MEP, and their respective RL- and coordination approaches for NERD. The results for our proposed learning methods for EPs and MEPs showed promising performances in terms of accuracy estimation and outcome optimization for NERD on news articles and tweets. While our SRL approach based on PSL (EP-BatchPSL-All) reached the best F1-measure for NERD outcome optimization on tweets, our coordination approach for MEPs (MEP-ACoordH-OnGD-All) did outperform all others with regards to F1-measure on news articles. Similar conditions hold for precision and recall for both data sets, where different proposed learning approaches dominate. The differences with regards to the performances are balanced with differences in terms of computational complexity. We presented lightweight approaches based on the EWH algorithm (e.g. MEP-ACoordH-OnGD-All) with low complexity and – on average – adequate performance, but also developed richer models (e.g. EP-BatchPSL-All) with higher complexities and superior performance. As a consequence, all methods have certain advantages and have to be chosen based on the individual task, available data and potentially available end user preferences (e.g. precision versus recall or available resources for computation).

Our NERD evaluations for Online RL and Batch RL for corresponding EPs answered Research Question 1 & Research Question 2, where the corresponding Hypothesis 1 & 2 can be confirmed.

Our NERD evaluations for a passive coordination approach as well as two active coordination approaches for MEPs answered Research Question 3, where the corresponding Hypothesis 3 can be confirmed.

Part III

Web Automation

This part deals with the problem of Web automation for multi-step tasks with expert advice. We first extend EPs with semantic representation, yielding SEPs, to approach resolving heterogeneity of experts and data, and subsequently introduce suitable Web components and their interactions for Web automation (Chapter 8). To evaluate the proposed Web components, we apply the latter to two medical assistance- as well as one NLP scenario in Chapter 9.

Chapter 8

Automating Semantic Expert Processes

This chapter proposes the SEP framework as well as a Web architecture to automatically solve multi-step tasks on the Web. It deals with Research Question 4 & 5, where we approach heterogeneity of expert functionalities, -implementations and -interfaces, and then deal with the problem of Web automation. We approach heterogeneity by lifting experts with lightweight semantics in terms of LAPIs (Hypothesis 4), and with respect to Hypothesis 5, use a decision-theoretic and data-driven approach to find, query and combine them. This chapter introduces the conceptual approaches to prior research questions and hypotheses, and Chapter 9 will deal with instantiating as well as evaluating them for the scenarios presented in Chapter 2.
Our references for this chapter are [51, 101–103].

8.1 Introduction

Service enabling platforms such as *Algorithmia* increase the motivation for developers to publish their experts in order to make them universally and reliably accessible as well as to earn money. These platforms expose experts via *RESTful APIs* which can be directly accessed via HTTP or a wide range of programming languages, and have end users pay for single expert calls. This trend is also perceivable in sensitive fields such as *medicine*, where monetary aspects are not of primary focus but component reuse and -composition are becoming increasingly relevant. Resulting Web components offer great potential to support physicians with their daily work by automating the processing of patient-related data with *State-of-the-Art* experts. Examples include *image processing*, where prominent toolkits such as MITK offer a high variety of component-based image processors with exposed hyperparameters, or *organ simulation*, where the medical markup simulation language (MSML) [127] eases the configuration of *HiFlow*^{3} [4] simulations in a XML-based format.

To solve multi-step tasks with such APIs, several problems emerge. Exposed interfaces of individual experts usually in terms of names and functionalities of *input parameters*, which renders automatic expert discovery difficult. In addition to interfaces, expert functionalities might vary in granularity, where some *expert services* offer the *bundled* functionality of several steps at once. Integrating and aligning experts with overlapping functionality is thus

necessary and difficult. Finally, composed expert pipelines potentially generate faulty solutions for several problem instances, as they usually are static (i.e. the same experts are always used) even though *feedback* (e.g. training sets) about the respective mistakes is available.

By tackling these challenges, we work towards *autonomous* Web architectures which are able to find, query and weight eligible experts for multi-step tasks, and improve their performance (in terms of correct solutions) over time by exploiting feedback. We build on prior work in the Semantic Web and propose the SEP framework, which extends EPs with semantic representations for states, actions and experts. By making experts available as LAPIs (i.e. semantically enriched Web services; see Definition 3.50) and enabling their data-driven execution with *Linked Data-Fu* (LD-Fu) [123], they can be easily discovered and used for composition. The resulting set of *semantic experts* is weighted and composed by *semantic meta components*, which are semantically lifted *Planning* and *Learning* approaches for sequential decision problems.

We evaluate our work on problem settings in medical assistance- as well as NLP (Chapter 9), where data sources, such as patient-related information or text, are heterogeneous as they often consist of diverse data distributions. We introduce *semantic meta components* for both scenarios, which make use of available structured knowledge and training sets to automatically select, weight and execute experts.

8.1.1 Challenges

The main challenges (see Section 1.2) for automating the selection, execution and composition of experts are as follows.

- If descriptions for experts are available to end users, the process of understanding the latter is often tedious and difficult, as they often are unstructured and short (*Challenge 6*).

- Interfaces of implemented experts are diverse, which might cause competing experts for the same step to require different input formats, or experts for two subsequent steps to require glue code to be written (*Challenge 7*).

- Given that an expert has been selected for a step, its automatic executing is difficult, as mappings from data to input parameters might be complex and should not be hard-coded (*Challenge 8*).

- To enable flexible use of expert services, descriptions about their functionality (e.g. the service class summarizing the functionality) must not be tailored to a single step. However, if general or no functional descriptions are available, one needs to actively find expert compositions from available input data to a predefined goal (*Challenge 9*).

- When there are multiple experts available for a single step, one needs to deal with conflicting hypotheses, as approached in Part II of this thesis, in a flexible Web architecture (*Challenge 10*).

8.1.2 Contributions

We now summarize our contributions with respect to the challenges, as introduced in Sec 1.4.

(i) We exploit Semantic Web technologies to lift expert descriptions with semantic annotations and publish them as LAPIs. The resulting set of semantic experts is easily discoverable and dynamically executable, while using lightweight semantic annotations (*Contribution 4*).

(ii) We formalize SEPs and disclose central goals for automatically solving multi-step tasks on the Web (*Contribution 5.1*).

(iii) We present semantic experts and semantic meta components for decision-making in medical scenarios (see Section 9.2). For the TPM scenario, we present an abstract semantic planner and a grounded semantic planner, which enable to automatically solve the semantic TPM task. For Surgical Phase Recognition, develop a grounded semantic learner, which is able to combine the predictions of semantic experts (*Contribution 5.2*).

(iv) We disclose novel challenges for SEPs for NLP in Web domains by lifting NED as well as NER experts and extending medical semantic meta components. More specifically, we develop a novel grounded semantic planner to deal with the availability of numerous decision candidates for NERD (*Contribution 5.3*).

(v) We evaluate semantic experts as well as semantic meta components with respect to time-efficiency as well as the ability to correctly solve semantic tasks (*Contribution 5.4*).

The remainder of this chapter is structured as follows. We formalize semantic tasks and introduce SEPs in Section 8.2, and describe the automation components of our Web architecture in Section 8.3. Section 8.4 defines necessary concepts for the individual automation components and explains how we they are used solve multi-step tasks on the Web. We discuss our proposed architecture with respect to SEPs in Section 8.5 and conclude the chapter in Section 8.6.

8.2 Problem Formalization

We now introduce the concept of *semantic tasks*, which extend our definition of tasks (see Definition 3.1) with concepts of the Semantic Web. A semantic task consists of structured

state-, action- and expert spaces, where a structured space can either be abstract or grounded. A *abstract structured space* is modeled as BGP (see Definition 3.48), i.e. a RDF graph with variables. BGPs express various degrees of knowledge about the respective spaces and are used to model generalizable patterns. A *grounded structured space*, on the other hand, is modeled as RDF graph (i.e. no variables are allowed).

Definition 8.1 (Semantic Tasks)**.** *A (grounded) semantic task extends a task with structured spaces, i.e. is a* 3*-tuple* $(X_{RDF}, B_{RDF}, Y_{BGP})$.

- *Let* X_{RDF} *be the grounded structured input space with* $x \in X_{RDF}$ *a RDF graph.*
- *Let* B_{RDF} *be the grounded structured background knowledge space, i.e. a RDF graph.*
- *Let* Y_{BGP} *be the abstract structured output space with* $y \in Y_{BGP}$ *a BGP.*

The structure of multi-step tasks remains equal, but input, background knowledge and output spaces are assumed to be structured. We extend the formalization of tasks with grounded and abstract elements to account for domain knowledge available before a task is solved. The textual information of NLP tasks is, for example, still available for grounded states and might be transformed to features used for *learning* expert weights. While input- and background knowledge spaces are grounded, the goal space is considered abstract. The intuition is that no resource exists for the given task, unless it was successfully solved or expert hypotheses are available. *Multi-step semantic tasks* follow Definition 3.1, where solution functions are lifted analogously to semantic tasks.

To this end, we are interested in three related outcomes: (i) Finding possible sequences of experts to solve a given task, (ii) querying experts based on the current state and (iii) integrating expert weight learners to choose better expert hypotheses based on training sets. We argue that semantic tasks can be implemented and solved as semantic liftings of EPs, namely SEPs. For outcome (i) we define *Abstract SEPs*, where experts are not queried and, thus, grounded states are not derived, but an abstract plan has to be found, comprising expert sequences which solve the semantic task. For outcomes (ii) and (iii), we define *Grounded* SEPs, where experts have to be chosen, queried and weighted.

With Abstract SEPs, our goal is to develop and integrate appropriate Planning techniques to discover and compose experts for a semantic task. Abstract SEPs are defined on BGP spaces, as experts are not executed but comprise generalizable structured descriptions.

Definition 8.2 (Abstract Semantic Expert Processes)**.** *An Abstract SEP extends an EP and is a* 8*-tuple* $(S_{BGP}, A_{BGP}, \gamma, E_{BGP}, R_{BGP}, T_{BGP}, H, C_{EXP})$.

- *Let* S_{BGP} *be the abstract structured state space, where* $s \in S_{BGP}$ *is a BGP.*
- *Let* Γ_{BGP} *is the abstract structured starting state distribution, i.e.* $\Gamma_{BGP} \in \Delta(S^1_{BGP})$.
- *Let* A_{BGP} *be the abstract structured action space, where* $a \in A_{BGP}$ *is a BGP.*

- *Let $\gamma \in [0,1]$ be the discount factor, as defined for EPs.*
- *Let E_{BGP} be the abstract structured expert space, where $e \in E_{BGP}$ is a function $e : S_{BGP} \rightarrow A_{BGP}$, referred to as expert function, which returns an abstract structured action based on an abstract structured state. Each expert is assigned a weight function $w_{e_{BGP}} : SBGP \rightarrow \mathbb{R}$.*
- *Let transition function $T_{BGP} : S_{BGP} \times A_{BGP} \rightarrow S_{BGP}$ and reward function $R_{BGP} : S_{BGP} \times A_{BGP} \rightarrow \mathbb{R}$ be defined for abstract structured state- and action spaces.*
- *Let horizon $H \in \mathbb{N}^+$ and expert budget $C_{EXP} \in \mathbb{N}^+$ be defined as for EPs.*

We extend the EP formalization (see Chapter 5) as it captures central problems and challenges of learning to optimize expert advice for multi-step tasks. All spaces are lifted to structured representations, similar to semantic tasks, and reward and transition functions are defined accordingly.

As a SEP extends an EP with semantic representations, we assume the same protocol and properties with respect to value functions, expert weights and resulting distributions for abstract structured spaces and functions, and do not explicitly redefine the latter in this chapter. The state-value functions $V_{\text{BGP}} : S_{\text{BGP}} \rightarrow \mathbb{R}$ and $Q_{\text{BGP}} : S_{\text{BGP}} \times A_{\text{BGP}} \rightarrow \mathbb{R}$ are thus defined based on EPs (see Definition 5.2 & 5.3) and assume the same properties with respect to expert weights $w_{e_{\text{BGP}}}$, i.e. they have to be optimized in terms of the current reward and maximal future actions in future abstract structured states (see Definition 5.4). Note that for abstract states, experts weights are less expressive, as they represent a *class* of grounded states. The overall goal for an Abstract SEP is finding abstract policy $\pi_{\text{BGP}} : S_{\text{BGP}} \rightarrow A_{\text{BGP}}$ which maximizes the expected cumulative reward of the decision process, $R^{\text{CUM}}_{\text{BGP}}$ (see Definition 5.7).

As for non-semantic multi-step tasks, we map elements from semantic multi-step tasks $(X_{\text{RDF}}, B_{\text{RDF}}, Y_{\text{BGP}})$ directly to Abstract SEPs. Here, the state space for the first horizon, S^1_{BGP}, is defined as joint space over BGPs based on RDF spaces X_{RDF} and subset of background knowledge $\hat{B}_{\text{RDF}}$, where the resulting BGP space contains all necessary information (and only the latter) to describe the abstract semantic task for a generalized number of instances, i.e. $S^1_{\text{BGP}} = X_{\text{BGP}} \cup \hat{B}_{\text{RDF}} \cup V$, where space V contains the used variables (see Definition 3.48). State space S^{H+1}_{BGP} then comprises the output space of the semantic multi-step task, i.e. $S^{H+1}_{\text{BGP}} = Y_{\text{BGP}}$.

We now give an example for the transition from the semantic NERD task to an Abstract NERD SEP.

Example 8.1 (Semantic Multi-Step Task to Abstract Semantic Expert Process)**.** *We first deal with translating the semantic NERD task to an Abstract NERD SEP.*

- *The semantic NERD task has the following elements: X_{RDF} is the input space over texts (on the Web), where we might model the respective author, the venue or platform, or*

the time of posting. Y_{RDF} is the output space over mappings from texts to named entities as well as knowledge graph resources, where one might include annotations about the available texts (as for X_{RDF}) or resources. Finally, $\hat{B}_{RDF}$ defines the background knowledge space, containing structured training samples of (X_{RDF}, Y_{RDF}).

- *The translation to Abstract NERD SEP is straightforward as available background knowledge is not required. The structured state space thus only comprises information about the texts. Here, the resulting BGP state spaces might enforce the availability of of a text or the author. As a consequence, we translate S^1_{BGP} from X_{RDF} by using necessary predicates and objects (such as classes) to model the abstract structure in terms of BGPs. We follow the same procedure for S^{H+1}_{RDF} based on Y_{RDF}.*

Based on the semantic NERD task transition, we shortly discuss the difference to the semantic TPM task.

- *The elements of the semantic TPM task can be directly mapped from NLP to image processing. Only $\hat{B}_{RDF}$, the background knowledge space, might comprise (besides samples as for NERD) learned* mask models *for image segmentation, which can be directly used for inference.*

- *The translation from the semantic TPM task to the respective Abstract TPM SEP generally only differs in using background knowledge for state spaces. The abstract state space for the first horizon, S^1_{BGP}, needs to comprise necessary mask models such that available experts can exploit the latter.*

Given a semantic task and an Abstract SEP, the problem of *Abstract Semantic Planning* entails only using available structured information of experts to find policy π_{BGP} (see Definition 3.43). Other than in RL, we do not need to learn or interact with the world, as the structured spaces allow us to enumerate all possible paths and rewards can be modeled to be exclusively positive for reaching the respective output spaces.

Definition 8.3 (Abstract Semantic Planning)**.** *Assuming a learned Abstract SEP model, i.e. learned or given R_{BGP} and T_{BGP}, or learned or given expert weight functions $w_{e_{BGP}}$, we need to find policy π_{BGP} to generate a semantic plan with start state s^1_{BGP}, which is a sequence $s^1_{BGP}, a^1_{BGP}, s^2_{BGP}, a^2_{BGP}, \ldots, a^{H+1}_{BGP}$.*

Given that an abstract plan has been generated and environmental feedback for expert queries is available, we can approach defining and solving Grounded SEPs. Other than in the abstract case, we need to query experts based on grounded RDF spaces and use expert weighting methods for selection.

Definition 8.4 (Grounded Semantic Expert Processes)**.** *A Grounded SEP extends an EP and is a 8-tuple $(S_{RDF}, A_{RDF}, \gamma, E_{RDF}, R_{RDF}, T_{RDF}, H, C_{EXP})$.*

- *Let S_{RDF} be the grounded structured state space, where $s \in S_{RDF}$ is a RDF graph.*
- *Let Γ_{RDF} is the grounded structured starting state distribution, i.e. $\Gamma_{RDF} \in \Delta(S^1_{RDF})$.*
- *Let A_{RDF} be the grounded structured action space, where $a \in A_{RDF}$ is a RDF graph.*
- *Let $\gamma \in [0,1]$ be the discount factor, as defined for EPs.*
- *Let E_{RDF} be the grounded structured expert space, where $e \in E_{RDF}$ is a function $e : S_{RDF} \to A_{RDF}$, referred to as expert function, which returns a grounded structured action based on a grounded structured state. Each expert is assigned a weight function $w_{e_{RDF}} : S \to \mathbb{R}$.*
- *Let transition function $T_{RDF} : S_{RDF} \times A_{RDF} \to S_{RDF}$ and reward function $R_{RDF} : S_{RDF} \times A_{RDF} \to \mathbb{R}$ be defined for grounded structured state- and action spaces.*
- *Let horizon $H \in \mathbb{N}^+$ and expert budget $C_{EXP} \in \mathbb{N}^+$ be defined as for EPs.*

The formalization of Grounded SEPs extends EPs with grounded structured state spaces. We again assume the same protocol and properties with respect to value functions, expert weights and resulting distributions for abstract structured spaces and functions, and do not explicitly redefine the latter in this chapter. The state-value functions $V_{\text{RDF}} : S_{\text{RDF}} \to \mathbb{R}$ and $Q_{\text{BGP}} : S_{\text{BGP}} \times A_{\text{BGP}} \to \mathbb{R}$ are extended based on EPs (see Definition 5.2 & 5.3) and assume the same properties with respect to expert weights $w_{e_{\text{RDF}}}$, i.e. they have to be optimized in terms of the current reward and maximal value for future grounded structured states (see Definition 5.4). The overall goal for a Grounded SEP is finding grounded policy $\pi_{\text{RDF}} : S_{\text{RDF}} \to A_{\text{RDF}}$ which maximizes the expected cumulative reward of the decision process, $R^{\text{CUM}}_{\text{RDF}}$ (see Definition 5.7).

Mapping elements from semantic multi-step tasks $(X_{\text{RDF}}, B_{\text{RDF}}, Y_{\text{BGP}})$ to Grounded SEPs is analogous to Abstract SEPs. The state space for the first horizon, S^1_{RDF}, is defined based on RDF space X_{RDF} and subset of background knowledge $\hat{B}_{\text{RDF}}$, where the resulting RDF space contains all modeled information to describe the grounded semantic task for an instance, i.e. $S^1_{\text{RDF}} = X_{\text{RDF}} \cup \hat{B}_{\text{RDF}}$. State space S^{H+1}_{RDF} is based on the output space of the semantic multi-step task, i.e. $S^{H+1}_{\text{RDF}} = Y_{\text{RDF}}$.

The problem of *Grounded Semantic Learning* is the analogy to expert weight leaning in EPs, which we now define.

Definition 8.5 (Grounded Semantic Learning)**.** *Given a Grounded SEP, grounded semantic learning entails estimating expert weight functions $w_{e_{RDF}}$ for all semantic experts $e_{RDF} \in E_{RDF}$ in order to maximize the expected cumulative reward R^{CUM}_{RDF}.*

We can use RL techniques proposed in Chapter 5 & 6 to learn a policy based on the unstructured elements of states in SEPs. The remaining challenge is thus to integrate appropriate RL techniques into Grounded SEPs.

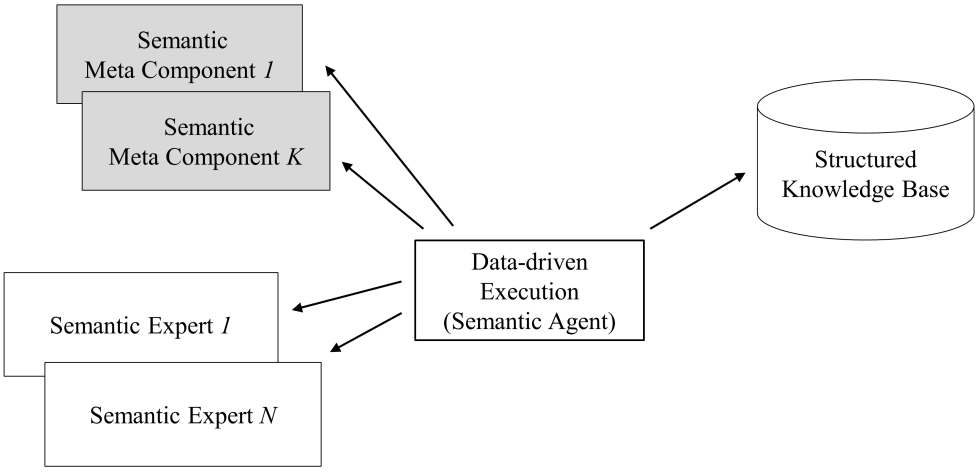

Figure 8.1: Schematic overview of automation components.

For a given Grounded SEP, the problem of finding a *grounded plan* (i.e. policy π_{RDF}) is referred to as *Grounded Semantic Planning* and entails to enable correct expert queries for a given state.

Definition 8.6 (Grounded Semantic Planning)**.** *Given a Grounded SEP, a grounded semantic plan with start state s^1_{RDF} is a sequence $s^1_{RDF}, a^1_{RDF}, s^2_{RDF}, a^2_{RDF}, \ldots, s^{H+1}_{RDF}$, where we have to find a respective state $\hat{s}^h_{RDF}$ for expert e^h_{RDF} such that $a^1_{RDF} = e_{RDF}(\hat{s}^1_{RDF})$.*

We now introduce the different components we disclosed to tackle prior defined learning and planning problems in both Abstract- and Grounded SEPs within a Web-based architecture.

8.3 Web Components for Automation

Our Web architecture comprises four core *automation component types* (see Figure 8.4). We will now explain their respective functionality.

1. A Structured Knowledge Base
2. Semantic Experts
3. Semantic Meta Components
4. A Data-Driven Execution Engine (Semantic Expert Advice Agent)

Non-functional requirements	Functional requirements
Domain Experts	*Inputs*
Service Endpoint	*Preconditions*
Example request & response	*Outputs*
Expert class	*Postconditions*

Table 8.1: Minimal description for semantic experts.

8.3.1 Structured Knowledge Base

A KB, such as a triple store or a virtually integrated collection of triples, stores both metadata and data, and provides a common and controlled vocabulary, modeled with RDF. As the Linked Data principles suggest (see Section 3.4.5), persistent URIs to resources have to be available and provide sufficient information for lookups. Appropriate concepts for experts and data have to be available to enable the integration of novel components.

The KB thus essentially comprises modeled or reused ontologies, which are used to annotate experts, data and semantic-meta components or to generate *provenance metadata* for the execution of semantic experts. Hence, all triples of grounded and abstract state-, action- and expert spaces are either in the KB or can be reached via *link traversal*.

8.3.2 Semantic Experts

In our Web architecture, experts are deployed as Web services to make them easily accessible. We follow the concept of LAPIs (see Definition 3.50) and refer to the resulting experts as *semantic experts*.

A semantic expert provides a *standardized description* with respect to its functionality, usage and origin by reusing concepts of the KB. The description defines how to communicate with the semantic expert and how to execute its methods. A minimal set of information of the description is summarized in Table 8.1.

Here, **non-functional** requirements define metadata which are not needed for using the semantic expert. The *service endpoint* denotes an exception to this rule, as it constitutes the service resource used for issuing HTTP requests. Other non-functional requirements comprise *domain experts* (i.e. developers of the expert functionality), *example requests* and *-responses* (i.e. exemplary groundings of preconditions and postconditions to support end users in their usage) or the *expert class*, which is situated in a taxonomy for general semantic expert discovery. **Functional requirements** deal with required conditions for *using* the semantic expert. We exclusively require structured annotations about *inputs* and *preconditions* as well as *outputs* and *postconditions*. Preconditions and postconditions, here, define strict rules about the states before- and after executing the semantic expert. As a consequence, we do use the expert class (or generate further structured annotations about the available functionality) to solve

SEPs, but *plan* and *learn* to *infer* the adequacy of a semantic expert.

We now generally define semantic experts (see Definition 8.7).

Definition 8.7 (Semantic Expert)**.** *A semantic expert is a* 5*-tuple* $(\texttt{Preconditions}_e, \texttt{Postconditions}_e, \texttt{Description}_e, e_{BGP}, e_{RDF})$.

- *Let* $\texttt{Preconditions}_e \in S_{BGP}$ *be the precondition BGP.*
- *Let* $\texttt{Postconditions}_e \in S_{BGP}$ *be the postcondition BGP.*
- *Let* $\texttt{Description}_e$ *be the RDF graph with all functional and non-functional requirements.*
- *Let* e_{BGP} *the abstract expert function which evaluates* $\texttt{Preconditions}_e$ *on a grounded state and generates an abstract resulting state without querying the expert.*
- *Let* e_{RDF} *the grounded expert function which evaluates* $\texttt{Preconditions}_e$ *on a grounded state and generates a grounded action by querying the expert.*

An intuitive example of an arbitrary image processor is given in Figure 8.2. The semantic expert converts images in PNG format to JPEG. It requires (via its preconditions) that inputs have to be typed with `kbont:Image` with format `kbont:PNG`, while outputs need to have format `kbont:JPEG`. We use the `rdf:type` predicate to model the expert class and model predicates `kbont:contributor` and `kbont:exampleRequest` for annotating domain experts and example requests. Note that we use `kb` and `kbont` as generic namespaces to describe instances and conceptual elements (i.e. properties or classes) available in the knowledge base [1]. The `MSM` namespace [2] corresponds to the *minimal service model* [73] which advocates and enables lightweight semantics for Web services. Listing 8.1 summarizes the used namespaces for this chapter.

```
1 @prefix kbont:   <http://example.org/kb/concepts#>.
2 @prefix kb:      <http://example.org/instances/id/>.
3 @prefix msm:     <http://cms-wg.sti2.org/minimal-service-model#>.
4 @prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
5 @prefix dc:      <http://purl.org/dc/elements/1.1/>.
```

Listing 8.1: (Generic-) namespaces used for KB concepts and instances.

Further note that the *degree of detail* for preconditions and postconditions is often dependent on the *wrapping process* of the respective expert, as hyperparameters or generated metadata might be hidden (and thus not exposed). The latter is usually dependent on the intended range of applications for the semantic expert. Larger numbers of hyperparameters and metadata might foster reusability of the semantic expert, but also complicate its usage. The same holds

[1] We will point out which exact ontologies we used in the respective scenarios.

[2] `http://iserve.kmi.open.ac.uk/ns/msm/msm-2014-09-03.html` (*accessed on 05/01/2018*)

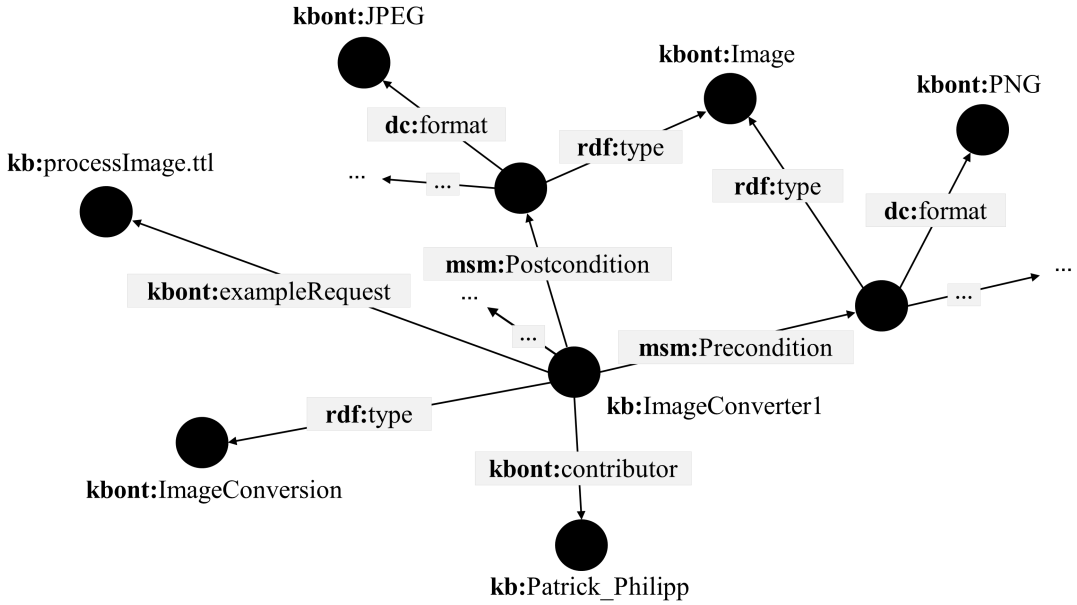

Figure 8.2: An exemplary semantic image conversion expert.

for the used or modeled properties and classes for inputs, preconditions, outputs and postconditions, where higher degrees of complexity increase applicability, but require more efforts for integration.

We now deal with semantic meta components, which integrate *Learning* (see Part II) and *Planning* methods into the Web automation architecture.

8.3.3 Semantic Meta Components

Semantic meta components for SEPs are solutions to abstract semantic planning, grounded semantic planning or grounded semantic learning. Our Web architecture enables flexible using, testing and exchanging of semantic meta components by requiring structured annotations for their usage and, if reasonable, their availability as LAPIs.

Definition 8.8 (Semantic Meta Components)**.** *A semantic meta component is a semantic expert, i.e. 5-tuple* $(\texttt{Preconditions}_e, \texttt{Postconditions}_e, \texttt{Description}_e, e_{BGP}, e_{RDF})$*, which implements a strategy for abstract semantic planning, grounded semantic planning or grounded semantic learning.*

Similar to a semantic expert, a semantic meta component specifies the amount of information it requires by its preconditions and is only called if the *agent* can provide all information. Examples for such preconditions are performance tables containing a history of validated results on training sets, or a list of applicable states for the usage of a semantic meta component.

8.3.4 Semantic Expert Advice Agent

Until now, we described required representations and implementations of semantic experts, semantic meta components, as well as the KB. To this end, we need a flexible mechanism for discovering and executing respective LAPIs.

Our goal for Web automation is to compose semantic experts in order to automatically solve semantic tasks. Resulting compositions have to be sufficiently *dynamic* with respect to available inputs for the semantic task such that the best set of experts is correctly queried. We take a *data-driven* perspective and have preconditions of semantic experts or semantic meta components determine when the respective LAPI has to be executed. To this end, we exploit LD-Fu, a data-driven execution engine which is used as baseline for composing semantic experts.

Linked Data-Fu

LD-Fu [123] is a *rule-based* execution engine, describing and implementing a formalism to virtually integrate data from different sources in real-time. Rules are defined in N3 syntax (see Section 3.4.3) and consist of a general `IF-THEN` structure, where a set of rules is referred to as *linked program*.

The `IF` condition of a rule (also referred to as *head*) has to comprise a BGP or be empty, expressing the starting point of a linked program. The `THEN` clause then defines the action to be conducted given the head is fulfilled. Our focus, here, lies on the ability of LD-Fu to use HTTP methods which enables us to dynamically execute available LAPIs.

LD-Fu fosters scalability of Web applications by enabling parallel queries of LAPIs. It relies on *polling* data sources to query LAPIs by a linked program, which is a contrary approach to *event-based* methods where triggers originate from the environment.

Linked Data-Fu for Semantic Experts

In our Web automation architecture, LD-Fu first searches eligible semantic experts for grounded states. By exploiting semantic meta components, we can then learn about semantic experts and plan correct semantic expert executions to solve semantic tasks. Each semantic expert is represented as single rule, which we automatically generate based on its description and current grounded state. The execution of a semantic expert is documented by publishing provenance as Linked Data. See Listing 8.2 for the exemplary generated provenance of the semantic image conversion expert.

```
@prefix kbont:   <http://example.org/kb/concepts#>.
@prefix kb:      <http://example.org/kb/instances/id/>.
@prefix msm:     <http://cms-wg.sti2.org/minimal-service-model#>.
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix dc:      <http://purl.org/dc/elements/1.1/>.
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#>.
@prefix sparql:  <http://www.w3.org/TR/rdf-sparql-query/#>.

kb:execution1    rdf:type       kbont:SemanticExpertExecution;
                 kbont:expert
                                kb:SemanticImageConversionExpert1;
                 dc:date        "2017-11-29T22:18:00"^^xsd:dateTime;
                 kbont:input   _:i;
                 kbont:output  _:o.

_:i              rdf:type       kbont:GroundedPrecondition;
                 rdf:value   """{

                    kb:image1   rdf:type    kbont:Image;
                                dc:format   kbont:PNG.

                                  }"""^^sparql:GraphPattern.

_:o              rdf:type       kbont:GroundedPostcondition;
                 rdf:value   """{

                    kb:image2   rdf:type    kbont:Image;
                                dc:format   kbont:JPEG;
                                kbont:convertedFrom
                                            kb:image1.

                                  }"""^^sparql:GraphPattern.
```

Listing 8.2: Exemplary provenance for semantic image conversion expert.

Figure 8.3 summarizes this general process for a simple semantic expert. When a grounded state fulfills the preconditions of the semantic expert, a HTTP `POST` request with grounded preconditions is issued to its service URI. While this enables automatically executing semantic experts for numerous use cases, there are complex preconditions which we deal with in Section 9.3.2.

The automatic matching between semantic experts and structured data is highly advantageous. If semantic experts have to be trained with training sets, the agent automatically uses all annotated training data for execution. This is generally possible by simply defining rules for LD-Fu. Even more important, with a growing number of diverse semantic experts, we could automatically solve new semantic tasks without additional manual effort, if semantic descriptions match.

We refer to the LD-Fu engine instantiated in our Web architecture as *Semantic Expert Advice Agent* (in short *semantic agent*). The semantic agent exploits semantic meta components for *Learning* and *Planning* to take decisions for solving multi-step tasks with semantic experts.

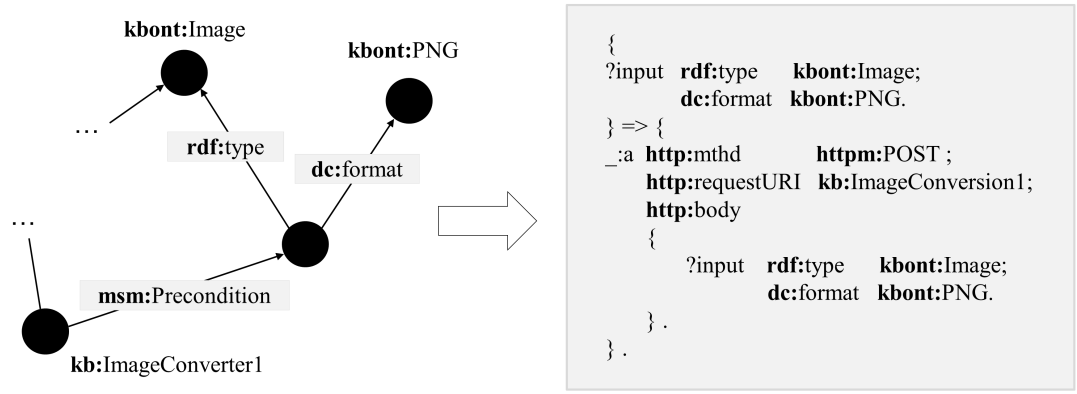

Figure 8.3: A LD-Fu rule for the semantic image conversion expert.

Definition 8.9 (Semantic Expert Advice Agent)**.** *A semantic expert advice agent solves multi-step tasks with available semantic experts by solving Abstract- and Grounded SEPs with abstract semantic planners, grounded semantic planners and grounded semantic learners, available as semantic meta components.*

Semantic experts and semantic meta components can be discovered by their type as well as preconditions and postconditions. However, a good selection of the latter is based on two aspects, namely finding *conceptually eligible* components or experts and finding a *well performing* components or experts, given the current state. We now deal with the respective problem settings in detail.

8.4 Solving Semantic Tasks with Semantic Meta Components

We now explain the required interplay between semantic meta components to solve abstract- as well as grounded semantic tasks with Abstract SEPs and Grounded SEPs, respectively. We first address *abstract semantic planning*, where we ignore state- and action groundings, and exclusively exploit descriptions of semantic experts and states. The second case deals with *grounded semantic learning* and builds on selected semantic experts via abstract semantic planning. As dealt with in Part II of this thesis, we need to learn weights for experts to find the optimal hypothesis. In the *grounded semantic planning* setting, we need to automatically find expert configurations to solve a multi-step task for a given grounded starting state and abstract goal.

In Figure 8.4 we give a generic overview of interactions between the semantic agent and semantic meta components, semantic experts and the KB. The semantic agent first queries the knowledge base to get all available semantic experts and then calls an abstract semantic planner. It executes the resulting set of semantic experts and subsequently calls a grounded semantic learner, capable of assessing and combining the generated candidate outputs. The

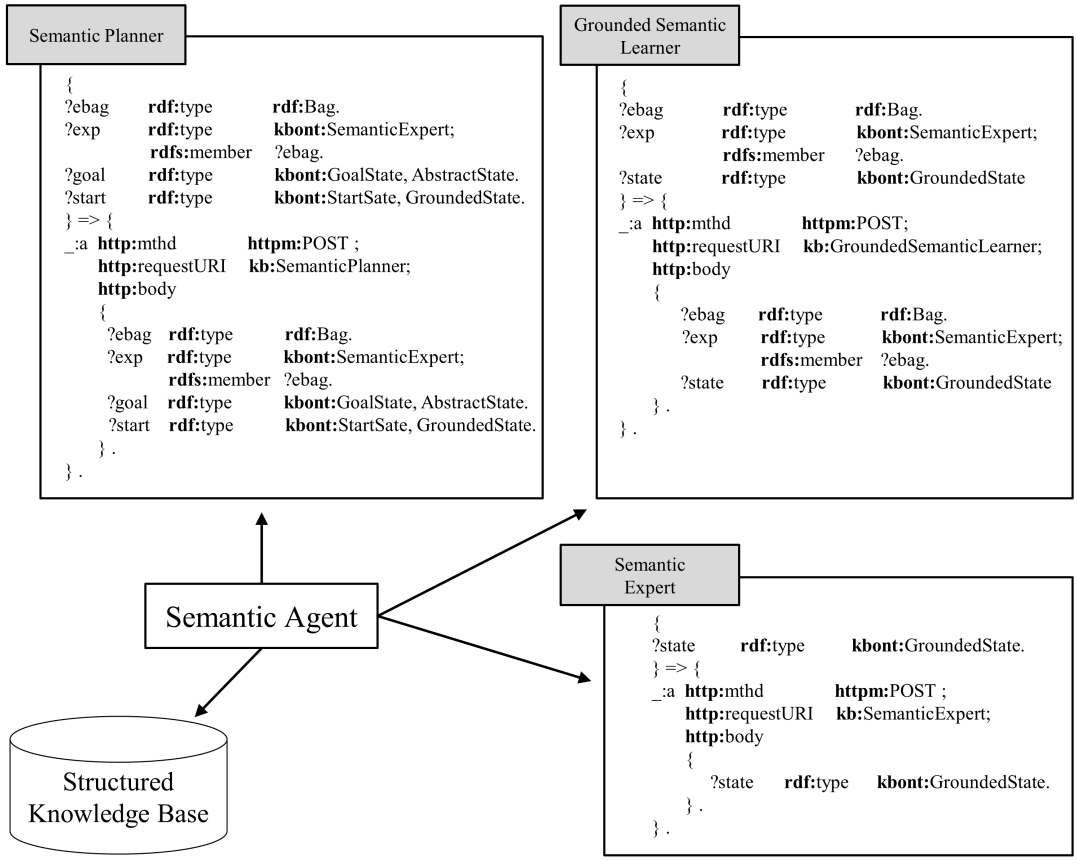

Figure 8.4: Interactions of semantic meta components.

grounded semantic planner is needed to execute selected grounded experts and might be implemented *within* a semantic agent service.

8.4.1 Abstract Semantic Planners

For a semantic task and corresponding Abstract SEP, one first evaluates a stating state s^1_{BGP} in terms precondition matching of semantic experts. The semantic agent therefore generates a set of rules which can be evaluated against s^1_{BGP}. We cannot assume, however, that rules only trigger semantic experts which help reaching the goal state, as there might be a large amount of conceptually eligible semantic experts for a state. An *abstract semantic planner* is able to reduce the set of semantic experts we query by checking if the latter support reaching the goal state.

Figure 8.4 illustrates the agent rule to call the abstract semantic planner depending on its preconditions. We describe one implementation of an abstract semantic planner in Sec-

tion 9.2.2 for the TPM scenario.

8.4.2 Grounded Semantic Learners

In the grounded semantic learning setting, we want to solve a Grounded SEP by having access to several *exchangeable* semantic experts which solve the same step (i.e. we solve the *Learning problem*; see Part II). Eligible semantic expert candidates are known (e.g. due to an abstract semantic planner), but one has to deal with uncertainty about their expected performances given a grounded state s_{RDF}. A grounded semantic learner selects semantic experts before their execution by estimating their performance, which is crucial when large numbers of semantic experts are eligible for a Grounded SEP.

Figure 8.4 describes the generic rule to execute any available grounded semantic learner. Section 9.2.3 introduces a semantic learner for medical phase recognition which can be reused for other tasks, such as NERD.

8.4.3 Grounded Semantic Planners

Other than in abstract semantic planning, grounded semantic planning requires information about the grounded state s_{RDF} of the respective Grounded SEP. Grounded semantic planners are needed when preconditions and postconditions of a LAPI do not suffice to solve a task, as state representation or semantic expert descriptions are too complex. A grounded semantic planner is thus executed to find correct configurations for available input parameters of the chosen semantic experts. However, we might need to *repeatedly plan* for solving a Grounded SEP, namely potentially after each step is processed. We thus assume that the grounded semantic planner is part of the implementation of the semantic agent.

We present a grounded semantic planner for the semantic NERD task in Section 9.3.

8.5 Discussion

We first discuss the added value of the SEP framework and then deal with the more general goal of developing self-adaptive systems.

8.5.1 On the Added Value of SEPs

Modeling fine-grained semantic annotations, and developing and integrating meta components adds manual work for domain- and machine learning experts. Still, resulting semantic meta components enable to gradually learn which semantic experts to select given state characteristics and how to automatically compose them. Pipelines are thus dynamically changed for each state (and decision candidate) to maximize the correctness of task solutions, which usually remain static unless manually changed (e.g. or via fine-grained manual preconditions).

This also holds for automatically taking into account novel semantic experts with adequate semantic interfaces.

8.5.2 On The Goal of an Self-Adaptive System

Since we enable to choose among different semantic meta components for planning as well as learning, one could have them compete as well. This, however, is what Vilalta & Drissi [136] depict as curse of infinite bias. We want the Web architecture to be self-adaptive and improve with experience, which it partially achieved by automatically considering novel data sources (e.g. training sets) or semantic experts. However, each semantic meta component has a certain bias with respect to its used methodology or training process. Hence, while it is interesting to have semantic meta components compete with each other, the problem of how to deal with their respective bias is important on its own.

8.6 Summary

In this chapter, we lifted EPs and tasks to semantic representations in order to work towards Web automation, where available semantic experts are discovered, weighted and queried without human intervention. We first defined semantic tasks, where input-, background knowledge and output spaces are considered to be RDF graphs. To solve semantic tasks, we defined Abstract- as well as Grounded SEPs, extending EPs with BGP and RDF representations. A suitable Web architecture to solve resulting abstract semantic planning, grounded semantic planning and grounded semantic learning problems then consists of a KB, semantic experts, semantic meta components and a data-driven execution engine, which we refer to as semantic agent.

Semantic experts were defined as LAPIs, where lightweight semantic annotations were proposed for modeling preconditions and postconditions. Our work on semantic experts addressed Research Question 4, where the corresponding Hypothesis 4 is partially confirmed and will be revisited after our applications and evaluations of semantic experts to the medical assistance scenarios as well as the NERD use case.

Based on semantic meta components, we can flexibly discover suitable experts for a SEP via abstract semantic planners, configure executions with grounded semantic planners and weight them with grounded semantic learners. By using LD-Fu as rule-based execution engine for the semantic agent, we can query semantic meta components and eventually execute semantic experts.

Our work on the Web architecture for the automation problem addressed Research Question 5, where the corresponding Hypothesis 5 is partially confirmed and will be revisited after our applications and evaluations of the Web architecture with diverse semantic meta components to the medical assistance scenarios as well as the NERD use case.

Chapter 9

Applications & Evaluations of Semantic Expert Processes

This chapter deals with Research Question 4 & 5, where the heterogeneity of functionalities, implementations and interfaces with respect to the goal of Web automation is approached. It extends Chapter 8 in terms of providing applications and evaluations of the proposed SEP framework to fully confirm Hypothesis 4 & 5, where semantic liftings for LAPIs as well as a decision-theoretic and data-driven automation framework are proposed.
Our references for this chapter are [51, 101–103], where we additionally note the application of the proposed methods in [114].

9.1 Introduction

SEPs lift EPs to richer state-, action and expert representations to solve semantic tasks. They constitute an important step towards the automation of multi-step tasks in Web-based environments, where such tasks are very common (e.g. in NLP or image processing). To solve SEPs, we require semantic meta components, namely grounded semantic learners, which are semantic extensions to learning methods proposed in Part II, and abstract semantic planners as well as grounded semantic planners to discover and correctly execute semantic experts. The *semantic agent* manages the communication among semantic meta components and flexibly executes resulting semantic experts on a data-driven basis.

In this chapter, we apply the Web automation framework to two different fields, namely *medical assistance* and *NLP*. For medical assistance, we deal with two scenarios, each requiring different semantic meta components. Surgical phase recognition is an intra-surgical, single-step task where sensor information has to be mapped to surgical phases. Based on available training sets, we deal with grounded semantic learning. The second scenario deals with medical image processing, more specifically TPM, where a number of headscans of a single brain tumor patient are automatically processed. Here, we deal with abstract semantic planning as well as grounded semantic planning to select and compose adequate semantic experts. Finally, the NLP scenario entails to solve NERD, where named entities have to be extracted from text and linked to KBs. We will deal with the joint task of abstract semantic planning, grounded semantic planning and grounded semantic learning.

Both fields pose distinct challenges for semantic tasks and serve as proof-of-concepts for SEPs. We provide additional evaluations for properties of semantic experts (e.g. time performance) and developed semantic meta components (e.g. precision or recall for grounded semantic learners). We summarized the challenges and contributions in Chapter 8.

9.2 Medical Assistance Scenarios

We introduce applications of our Web automation framework to two medical assistance scenarios, namely *Surgical Phase Recognition* and TPM. Before introducing the respective semantic experts and semantic meta components, we present the shared Web automation architecture for medical assistance.

9.2.1 Web Automation for Medical Assistance Tasks

Our Web architecture for medical assistance tasks was developed within the Cognition-Guided Surgery project [1]. Every expert considered in the scenarios was lifted and published as semantic expert. We modeled all semantic descriptions with domain experts and developers of the experts, and integrated them in a shared instance of a Semantic MediaWiki (SMW) [78], referred to as *Surgipedia*.

SMWs extend MediaWikis [2] with structured representations, where a single page corresponds to a unique resource. The MediaWiki syntax is extended to deal with RDF, such that triples can be directly edited, added or deleted. The latter is facilitated by using *templates* and *forms*, which are intuitive means to enable end users to quickly model and publish RDF triples. A SMW also allows to integrate available ontologies and maps them to locally modeled concepts or predicates, which eases reuse for end users. Finally, as SMW pages can be looked up using *RESTful* content negotiation to return RDF via HTTP `GET` requests, one can easily fulfill all recommended *Linked Data Principles* (see Section 3.4.5) by using SMWs to structure data.

Figure 9.1 illustrates the developed SMW form for end users to generate semantic description for their semantic experts (see Section 8.3.2). Preconditions and postconditions have to be modeled with additional forms, as the latter often corresponds to complex BGPs.

To this end, all semantic annotations are based on *Medical Subject Headings* (MeSH) [3], *Radiology Lexicon* (RadLex) [4], *SNOWMED-CT* [5] and the *Foundational Model of Anatomy* (FMA) [6]. We use an instance of *XNAT* [7] to store patient-relevant data and provide a RDF

[1] `http://www.cognitionguidedsurgery.de/` (*accessed on 05/01/2018*)
[2] `https://www.mediawiki.org/wiki/MediaWiki` (*accessed on 05/01/2018*)
[3] `https://meshb.nlm.nih.gov` (*accessed on 05/01/2018*)
[4] `https://www.rsna.org/RadLex.aspx` (*accessed on 05/01/2018*)
[5] `http://www.snomed.org/snomed-ct` (*accessed on 05/01/2018*)
[6] `http://si.washington.edu/projects/fma` (*accessed on 05/01/2018*)
[7] `http://www.xnat.org/` (*accessed on 05/01/2018*)

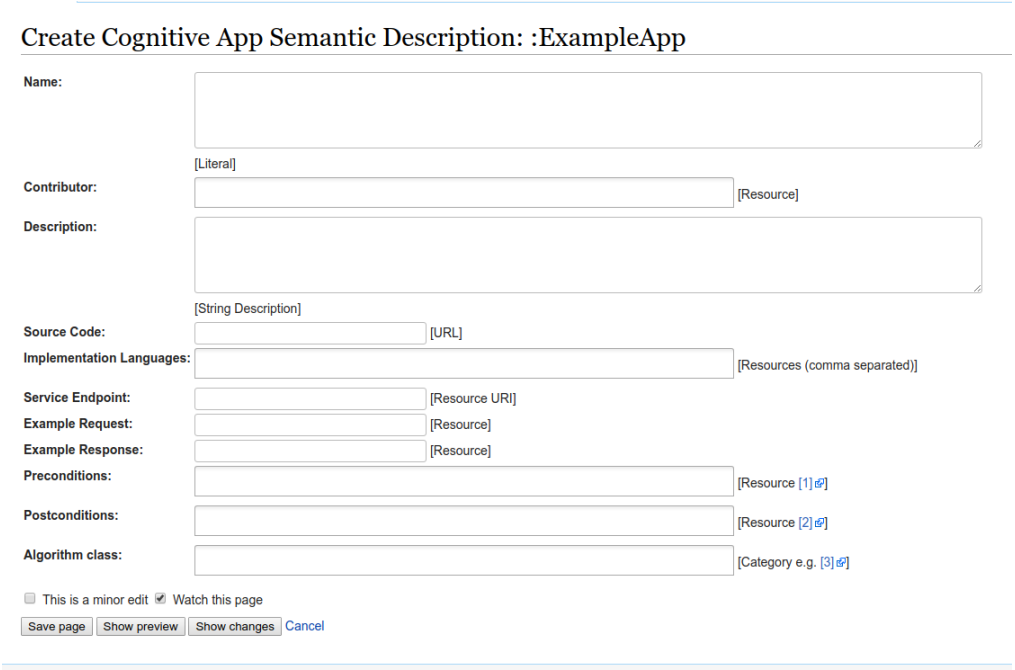

Figure 9.1: The SMW form to annotate experts for medical assistance.

wrapper which lifts XNAT with semantic concepts. The KB can be considered as *union* between *Surgipedia* (where all ontologies have been integrated) and its links to other resources, such as *XNAT*. The complete architecture with two semantic meta components is illustrated in Figure 9.2.

9.2.2 Semantic Tumor Progression Mapping

The TPM scenario consists of five image processing steps (as introduced in Section 2.3), where several headscans of a single patient are processed to generate a spatially aligned, intuitive overview of the tumor progression in order to support radiologists in their daily diagnostic and treatment workload. For our application, we use six experts of the MITK library which are summarized in Table 9.1 together with needed inputs and produced outputs. If not indicated otherwise, the experts expect and produce files in the NRRD format.

For step *Brain Normalization*, one available expert (*Standard Brain Normalization*) requires a brain image and a brain mask generated by a *Brain Mask Generation* expert, but a second expert (*Robust Brain Normalization*) requires additional (finer-grained) manual brain mask annotations. As a consequence, the *Standard Brain Normalization* expert can always be queried within the pipeline, but manually created image annotations are needed for the *Robust Normalization Expert*. Also, *Map Generation* does not require the *Tumor Segmentation* expert to

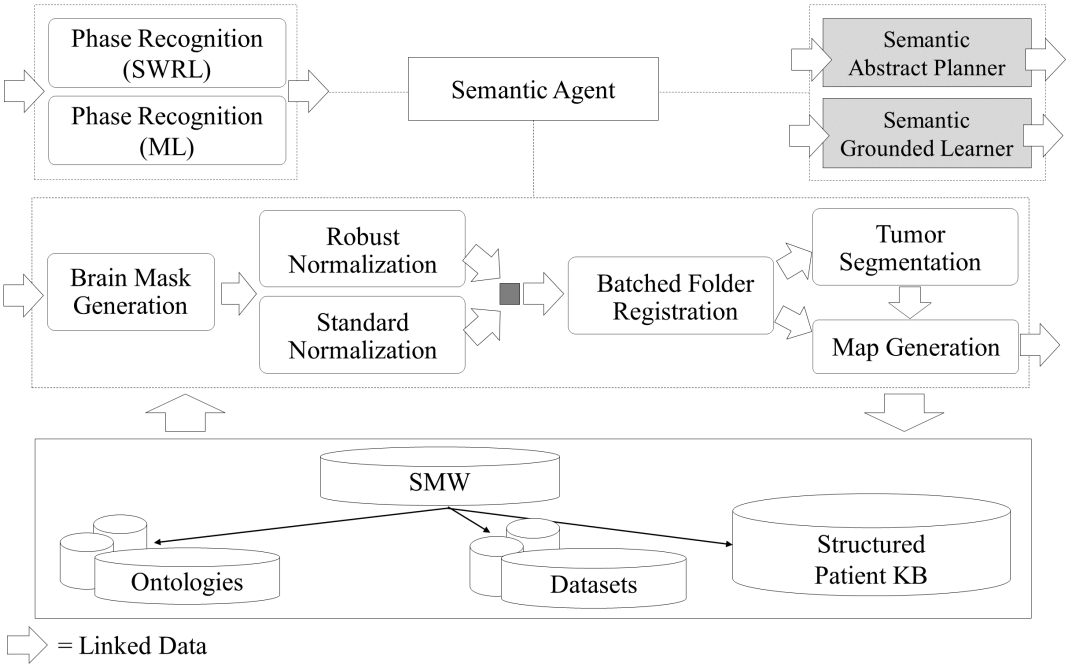

Figure 9.2: The automation architecture for medical assistance.

Expert	**Input**	**Output**
Brain Mask Generation	Headscan (NRRD or MHA) Atlas image Atlas mask	Brain image Brain mask
Standard Brain Normalization	Brain image Brain mask	Normalized brain image
Robust Brain Normalization	Brain image Minimum brain mask Maximum brain mask Exclude brain mask	Normalized brain image
Batched Folder Registration	Normalized brain image	Registered brain image
Tumor Segmentation	Registered brain image	Brain segmentation
Map Generation	Registered brain image Brain segmentation	Map (PNG)

Table 9.1: Needed inputs and generated outputs for TPM.

generate respective segmentations and can create a TPM without the latter. The TPM aggregates multiple (possibly tumor-segmented) brain masks of the same patient into a single PNG file. All residual input and output files are available in the NRRD format.

We first illustrate how we lifted TPM experts with semantics and then present our abstract semantic planner, which finds eligible semantic experts for the semantic TPM task.

Semantic TPM Experts

We developed and published semantic experts for every step in the TPM generation process. As an illustration, we shortly discuss the modeled preconditions and postconditions for semantic experts of the step *Brain Mask Generation*, which are depicted in Listing 9.1 & 9.2. Note that all semantic descriptions for TPM experts are presented in Appendix A and that individually presented pre- and postconditions are typed as `sparql:GraphPattern`.

A semantic *Brain Mask Generation* expert requires a typed patient-specific headscan resource in format `spc:NRRD` or `spc:MHA`, and a generic, patient-unrelated *brain atlas image* and *brain atlas mask*. After a respective semantic expert has been queried, the postconditions specify that a patient-specific *output brain image* with format `spc:NRRD` and a patient-specific *output brain mask* of the same format have to be available. Both generated images have to link to the input headscan, as recommended by the LAPI definition.

```
@prefix spc:     <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Category-3A>.
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix dc:      <http://purl.org/dc/elements/1.1/>.

?headscan        rdf:type         spc:Headscan;
                 dc:format        ?format.
FILTER (?format = spc:NRRD || ?format = spc:MHA)

?brainAtlasImg   rdf:type         spc:BrainAtlasImage;
                 dc:format        spc:NRRD.

?brainAtlasMask  rdf:type         spc:BrainAtlasMask;
                 dc:format        spc:NRRD.
```

Listing 9.1: Preconditions of the semantic Brain Mask Generation expert.

```
@prefix spp:     <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Property-3A>.
@prefix spc:     <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Category-3A>.
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix dc:      <http://purl.org/dc/elements/1.1/>.

?headscan        rdf:type         spc:Headscan;
                 dc:format        ?format.
FILTER (?format = spc:NRRD || ?format = spc:MHA)

?brainImage      rdf:type         spc:BrainImage;
                 dc:format        spc:NRRD;
                 spp:headscan     ?headscan.
```

```
?brainMask      rdf:type        spc:BrainMask;
                dc:format       spc:NRRD;
                spp:headscan    ?headscan.
```

Listing 9.2: Postconditions of the semantic Brain Mask Generation expert.

To this end, it is important to discuss a central property of preconditions and postconditions we refer to as *multiplicity*. The needed inputs and outputs for the Brain Mask Generation expert are unambiguous with respect to the used ontologies, as – for the stipulated preconditions – a single headscan, single atlas image and single atlas mask is needed. However, experts might be required to process an *a priori* unknown number of inputs. The resulting multiplicity problem is available for the *Brain Registration* expert, which spatially aligns all available *normalized* brain masks of a single patient.

To express this property in preconditions and postconditions, we propose to use the available `rdf:Bag` class for collections [8], where included resources are *unordered*. Listing 9.3 shows the resulting preconditions of the step, where a number of normalized brain masks are required as input, which are part of a `rdf:Bag`.

```
@prefix spp:   <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Property-3A>.
@prefix spc:   <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Category-3A>.
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#>.
@prefix dc:    <http://purl.org/dc/elements/1.1/>.

?norm_bag       rdf:type        rdf:Bag.

?normMask       rdf:type        ?norm;
                rdfs:member     ?norm_bag;
                dc:format       spc:NRRD;
                spp:headscan    ?headscan;
                spp:brainMask   ?brainMask.
FILTER (?norm = spc:NormalizedBrainMask ||
        ?norm = spc:RobustNormalizedBrainMask).

?brainMask      rdf:type        spc:BrainMask;
                dc:format       spc:NRRD;
                spp:headscan    ?headscan.

?headscan       rdf:type        spc:Headscan;
                dc:format       ?format.
FILTER (?format = spc:NRRD || ?format = spc:MHA)
```

Listing 9.3: Preconditions of the semantic Brain Registration expert.

To characterize and define the respective bags, we model a single *representative* resource of it in preconditions or postconditions. For the semantic Brain Registration expert, we define that the bag requires resources of type `spc:NormalizedBrainMask` *or* `spc:RobustNormalizedBrainMask`, for which property `rdfs:member` is used to

[8] `https://www.w3.org/TR/rdf-schema/#ch_bag` (*accessed on 05/01/2018*)

define the membership of a resource. Note that one can alternatively use property `rdf:li` to model the reciprocal triple, where a bag *consists of* resources.

Based on available semantic TPM experts, we now deal with their execution by the *semantic agent*. We generate a set of rules (i.e. a linked program), which states that respective semantic experts are only executed if their preconditions hold, as generally described in Section 8.3.4. For the semantic Brain Mask Generation expert, the resulting rule is depicted in Listing 9.4.

```
{
@prefix spc:    <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Category-3A>.
@prefix exp:    <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/id/>.
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix dc:     <http://purl.org/dc/elements/1.1/>.
@prefix httpm:  <http://www.w3.org/2011/http-methods#>.
@prefix http:   <http://www.w3.org/2011/http#>.

?headscan       rdf:type        spc:Headscan;
                dc:format       ?format.
FILTER (?format = spc:NRRD || ?format = spc:MHA)

?brainAtlasImg  rdf:type        spc:BrainAtlasImage;
                dc:format       spc:NRRD.

?brainAtlasMask rdf:type        spc:BrainAtlasMask;
                dc:format       spc:NRRD.
} => {
_:a http:mthd httpm:POST;
    http:requestURI exp:BMG;
    http:body
    {
        ?headscan       rdf:type        spc:Headscan;
                        dc:format       ?format.
        FILTER (?format = spc:NRRD || ?format = spc:MHA)

        ?brainAtlasImg  rdf:type        spc:BrainAtlasImage;
                        dc:format       spc:NRRD.

        ?brainAtlasMask rdf:type        spc:BrainAtlasMask;
                        dc:format       spc:NRRD.
    }.
}.
```

Listing 9.4: Rule for the semantic Brain Mask Generation expert.

In addition to rules for semantic experts, we need rules to query and aggregate the needed input data from the KB. In the medical assistance scenario, we assume a *starting* resource which points to *department* resources in a clinic. The latter point to individual *patient* resources which finally link to available structured information as well as raw files. The structure was reused from the REST interface of XNAT and extended according to the *Linked Data Principles*. Listing 9.5 depicts the set of rules for gathering patient-related information. The process of aggregating structured data at *runtime* (other than querying a single, global triple store) is referred to as *virtual data integration*.

```
@prefix rdf:       <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix vocxnat:   <http://aifb-ls3-vm2.aifb.kit.edu:8080/xnatwrapper/xnat#>.
@prefix xnat:      <http://aifb-ls3-vm2.aifb.kit.edu:8080/xnatwrapper/id/>.
@prefix httpm:     <http://www.w3.org/2011/http-methods#>.
@prefix http:      <http://www.w3.org/2011/http#>.

{
_:a http:mthd httpm:GET;
    http:requestURI xnat:xnatlist.
}

{
?list vocxnat:hasProject ?project.
} => {
_:a http:mthd httpm:GET;
    http:requestURI ?project.
}.

{
?project vocxnat:hasSubject ?subject.
} => {
_:a http:mthd httpm:GET;
http:requestURI ?subject.
}.

{
?subject vocxnat:hasFile ?file.
} => {
_:a http:mthd httpm:GET;
http:requestURI ?file.
}.
```

Listing 9.5: Linked program for gathering patient-related resources.

Having gathered and integrated the needed set of triples to query all semantic experts, *goal-oriented plans* are needed to (i) find semantic experts which solve respective steps of an Abstract TPM SEP and (ii) query the latter with correct inputs to solve a Grounded TPM SEP. We now show that we can reduce respective SEPs to MDPs in order to directly use available *Planning* techniques for the abstract and the grounded case.

An Abstract Semantic Planner for TPM

By assuming a controlled vocabulary for all semantic experts, we can reduce both Abstract TPM SEPs as well as Grounded TPM SEPs to MDPs. We first discuss task properties of Abstract- and Grounded TPM SEPs, and then deal with their reduction.

TPMs are *finite* multi-step tasks, where we can confidently set an upper bound to the number of steps needed to generate solutions. They are, however, not always *layered*, as several steps need to be repeated if multiple headscans of a single patient are available. An Abstract TPM SEP is *finite* and *layered*, as we are only interested in finding abstract plans to reach an abstract goal state. A Grounded TPM SEP is *finite* and *non-layered* to account for multiplicity. We

now reduce SEPs to infinite MDPs for Abstract SEPs and deal with Grounded SEPs in the subsequent section.

Definition 9.1 (Reduction of SEPs to Infinite-horizon MDPs)**.** *We reduce a SEP to an infinite-horizon MDP (see Definition 3.34),* $M_{SEP} = (S_{SEP}, A_{SEP}, \gamma, Tr_{SEP}, R_{SEP})$.

- *Let* S_{SEP} *the state space.*
- *Let* A_{SEP} *the action space.*
- *Let* γ *the discount factor hyperparameter, as defined for MDPs.*
- *Let* $Tr_{SEP} : S_{SEP} \times A_{SEP} \to S$ *the transition function.*
- *Let* $R_{SEP} : S_{SEP} \times A_{SEP} \to \mathbb{R}$ *the reward function.*

We distinguish between the abstract case ($M_{\text{SEP}}^{\text{ABSTR}}$) with exclusively abstract spaces (as for Abstract SEPs) and the grounded case ($M_{\text{SEP}}^{\text{GRND}}$) with RDF spaces (as for Grounded SEPs).

For Abstract SEPs, we explicitly reduce the complexity of given start- and goal states in order to find a preselection of semantic experts for the respective semantic task. We thus do not deal with the multiplicity property.

Definition 9.2 (Reduction of Abstract SEPs to Infinite-horizon MDPs)**.** *Given infinite-horizon MDP* M_{SEP}^{ABSTR} *(see Definition 9.1), we define the following properties for its spaces and functions.*

- S_{SEP}^{ABSTR} *is defined as enumeration of all unique preconditions and postconditions of all available semantic experts, where a state* s' *resulting from a transition – i.e.* $s' = Tr_{SEP}^{ABSTR}(s,a)$ *– theoretically contains the union of pre- and preconditions of the semantic expert which generates action a, i.e.* $s' = \texttt{Preconditions}_e \cup e_{BGP}(s)$.
- *Action space* A_{SEP}^{ABSTR}*, as in EPs, is state-dependent and consists of queries of abstract expert functions* e_{BGP} *for the respective state, i.e.* $A_{SEP}^{ABSTR} = \{a | e_{BGP}(s) = a\}$.
- *To define the goal of the semantic TPM task, we add a* dummy *action to* A_{SEP}^{ABSTR} *which, when the preconditions of the semantic TPM expert are reached, generates a loop, i.e.* $s = Tr_{SEP}^{ABSTR}(s,a)$.
- *The definition of* Tr_{SEP} *is straightforward, as it is deterministic. Choosing an action a (i.e. semantic expert) in state s (i.e. union of preconditions with potentially prior visited pre- and postconditions) thus always results in* s' *(i.e. union of s with preconditions of a).*

- *The reward function R_{SEP}^{ABSTR} is a $S_{SEP}^{ABSTR} \times A_{SEP}^{ABSTR}$ matrix with $|A_{SEP}^{ABSTR}| = |E| + 1$ (see Equation 9.1).*

$$R_{SEP}^{ABSTR}(s,a) = \begin{cases} 1 & \textit{if a equals dummy expert and s equals goal} \\ 0 & \textit{otherwise} \end{cases} \tag{9.1}$$

By using any strategy to plan in the MDP (e.g value iteration), we efficiently find eligible semantic experts to solve the task.

The resulting abstract semantic planner takes as input semantic experts, respective patient- or background information and a goal state, and returns a set of eligible semantic experts. The corresponding semantic agent rule for the abstract semantic planner is depicted in Listing 9.6. Here, the preconditions for the semantic Brain Mask Generation expert (see Listing 9.1) would replace the state.

```
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix sep:    <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/sep#>.
@prefix exp:    <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/id/>.
@prefix httpm:  <http://www.w3.org/2011/http-methods#>.
@prefix http:   <http://www.w3.org/2011/http#>.

{
?col          rdf:type      rdf:Bag;
              rdf:li        ?exp.
?exp          rdf:type      sep:SemanticExpert.
?goal         rdf:type      sep:GoalState,
                            sep:GroundedState.
?start        rdf:type      sep:StartState,
                            sep:AbstractState.
} => {
_:a   http:mthd         httpm:POST;
      http:requestURI   exp:abstractplanner;
      http:body
    {
        ?col       rdf:type   rdf:Bag;
                   rdf:li     ?exp.
        ?exp       rdf:type   sep:SemanticExpert.
        ?goal      rdf:type   sep AbstractState, sep:GoalState.
        ?start     rdf:type   sep:GroundedState, sep:StartState.
    }.
}.
```

Listing 9.6: LD-Fu rule for executing the abstract semantic planner.

A Grounded Semantic Planner for TPM

An abstract semantic planner discovers suitable semantic experts for a task, but cannot deal with challenges resulting from grounding preconditions for available states in RDF. We already discussed the problem of *multiplicity*, where semantic experts might require an unknown number of inputs for their input parameters. To illustrate further challenges, consider

the abstract goal of a semantic TPM task in Listing 9.7. As a generated TPM consists of timely-ordered processed brain masks, an unordered `rdf:Bag` is not suitable. We thus exploit a different class for collections, namely `rdf:Seq` [9], which are *ordered* sequences of resources.

```
@prefix spp:    <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Property-3A>.
@prefix spc:    <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Category-3A>.
@prefix sawsdl: <http://www.w3.org/ns/sawsdl#>.
@prefix xnat:   <http://aifb-ls3-vm2.aifb.kit.edu:8080/xnatwrapper/id/>.
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#>.
@prefix dc:     <http://purl.org/dc/elements/1.1/>.
@prefix sparql: <http://www.w3.org/TR/rdf-sparql-query/#>.
@prefix sep:    <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/sep#>.

xnat:goal1       rdf:type           sep:AbstractState, sep:GoalState;
                 sawsdl:modelReference
        [ a           sep:AbstractCondition;
          rdf:value   """{
            ?tpm1           rdf:type        spc:TumorProgressionMap;
                            spp:mapContent  ?tpm_seq.
            ?reg_bag        rdf:type        rdf:Bag.
            ?tpm_seq        rdf:type        rdf:Seq;
                            rdf:_1          ?reg_mask1;
                            rdf:_2          ?reg_mask2.
            ?reg_mask1      rdf:type        spc:RegisteredBrainMask;
                            dc:format       spc:NRRD;
                            spp:headscan    xnat:headscan1;
                            rdfs:member     ?reg_bag;
                            spp:normalizedMask
                                            ?brainMask1.
            ?reg_mask2      rdf:type        spc:RegisteredBrainMask;
                            dc:format       spc:NRRD;
                            spp:headscan    xnat:headscan2;
                            rdfs:member     ?reg_bag;
                            spp:normalizedMask
                                            ?brainMask2.
            ?normMask1      rdf:type        ?norm;
                            dc:format       spc:NRRD;
                            spp:headscan    xnat:headscan1.
            ?normMask2      rdf:type        ?norm;
                            dc:format       spc:NRRD;
                            spp:headscan    xnat:headscan2.
            FILTER (?norm = spc:NormalizedBrainMask ||
                    ?norm = spc:RobustNormalizedBrainMask).
            OPTIONAL {
               ?normMask1   spp:manualSegmentation
                                            xnat:seg1.
               ?normMask2   spp:manualSegmentation
                                            xnat:seg2.
                 }}"""^^sparql:GraphPattern].
```

Listing 9.7: Exemplary goal of a semantic TPM task.

[9] `https://www.w3.org/TR/rdf-schema/#ch_seq` (*accessed on 05/01/2018*)

The goal BGP defines that three headscans have to be used to generate the TPM, consisting of two registered brain masks. The semantic agent has to correctly query a semantic Brain Mask Generation expert and a semantic Brain Normalization expert for each headscan (and resulting brain mask) individually to then register all normalized brain masks and create the final TPM once. Given a large amount of headscans for a patient, where only two headscans should be chosen for the resulting TPM, we need to first ground all possible states with respect to available preconditions of semantic experts and then plan with respect to the goal.

In addition to grounding all possible expert configurations for a start state, we need to specifically deal with *multiplicity*, i.e. the availability of containers where multiple inputs of the same type are possible state spaces. An example for such containers was presented for the preconditions of semantic Brain Registration experts (see Listing 9.3), where a bag of brain masks with different headscans of a single patient is required for execution. We thus have to instantiate all possible bag configurations (i.e. bags of different sizes with different brain masks) to be able to correctly plan.

We can now define the elements of $M_{\text{SEP}}^{\text{GRND}}$, the infinite-horizon MDP for Grounded SEPs (Definition 9.3).

Definition 9.3 (Reduction of Grounded SEPs to infinite-horizon MDPs)**.** *Given infinite-horizon MDP M_{SEP}^{GRND} (see Definition 9.1), we define the following properties for its spaces and functions.*

M_{SEP}^{GRND} is a reduction of a Grounded SEP, where (i) grounded expert functions e_{RDF} generate grounded action hypotheses $a_{RDF} \in A_{RDF}$ as well as grounded states $s_{RDF} \in S_{RDF}$ or (ii) abstract expert functions e_{BGP} are queried to generate abstract action hypotheses $a_{BGP} \in A_{BGP}$ as well as abstract states $a_{BGP} \in A_{BGP}$.

M_{SEP}^{GRND} is instantiated by calling `initializeGroundedPlanning`$(s_{RDF}^{1}, \pi_{ABSTR}, E, s_{BGP}^{H+1})$ *(Algorithm 4) with the available start state s_{RDF}^{1}, learned abstract policy π_{ABSTR}, expert set E and available goal s_{BGP}^{H+1}.*

We first describe an algorithm for retrieving all valid groundings based on a semantic expert, which is needed to tackle the problem of multiplicity (see Algorithm 3), before presenting the general procedure to retrieve the model for $M_{\text{SEP}}^{\text{GRND}}$ (see Algorithm 4), as used in Definition 9.3.

The details of the algorithm are as follows:

- **line 1:** The collector set is initialized. It will eventually consist of all possible variable bindings for an expert's preconditions and a state.
- **line 2-5:** For all bags available in the precondition, we must find all possible configurations. We therefore assume that preconditions exactly define resources of the bag in order to query the state for all possible resources.
- **line 6-10:** For all available candidate resources with respect to the bag, we construct possible configurations and add them to the collector set.

Algorithm 3 `getValidGroundings`(e,s)

```
1: collector ← ∅
2: for all bag ∈ Preconditions_e do
3:     tempCollector ← ∅
4:     q ← getQuery(bag)
5:     candidates ← queryState(q, s)
6:     for all n ∈ { 1,...,|candidates| do
7:         tempCollector ← tempCollector ∪ subsets(n, candidates)
8:     end for
9:     collector ← collector ∪ tempCollector
10: end for
11: S,A,Tr ← getGroundings(collector, e, s)
12: return S,A,Tr
```

- **line 11-12:** Based on all residual variables in the preconditions (which are not within bags) and the additional configurations based on all bags, we can now derive all possible groundings for the preconditions and the state. More specifically, method `getGroundings` queries the given state based on the preconditions of the current expert (i.e. $\texttt{Preconditions}_e$) potentially grounded by bag configurations (in `collector`). The results of the query then comprise all adequate input configurations for the expert. The returned state space S consists of the resulting states after the expert is executed for the respective input configuration, retrieved via its postconditions (i.e. $\texttt{Postconditions}_e$) – the abstract function of the expert e_{BGP} is thus executed. A resulting state s' from executing the expert is defined as $s' = s \cup e_{\text{BGP}}(s_{\text{CONF}})$, where s_{CONF} is a possible input configuration. The action space A comprises the respective input configurations for the expert. The transition function Tr simply captures tuples (s,a,s') based on the semantic expert, where the semantic expert can be executed multiple times for different input configurations within a single step.

We now have a strategy to find all possible configuration for a state and expert. The general procedure to instantiate $M_{\text{SEP}}^{\text{GRND}}$ is summarized in Algorithm 4.

The details of the algorithm are as follows:

- **line 1-4:** Spaces S and A as well as functions Tr and R are initialized, where the latter are represented as tables.

- **line 5-8:** `abstractPlan` corresponds to a learned abstract policy (i.e. abstract plan) and thus comprises eligible experts for solving the respective TPM task, where $\texttt{abstractPlan}_h$ returns the expert for step h. Based on the abstract plan, we loop over all eligible experts (in order of the steps). For each expert, we initialize tables of temporary states, actions and transitions which result from executing the expert (grounded

Algorithm 4 `initializeGroundedPlanning`$(s, \texttt{abstractPlan}, E, \texttt{goal})$

1: $S \leftarrow \{s\}$
2: $A \leftarrow \emptyset$
3: $Tr \leftarrow \emptyset$
4: $R \leftarrow \emptyset$
5: **for all** $\texttt{h} \in 1, \ldots, \texttt{abstractPlan}$ **do**
6: $\quad \texttt{tempStates} \leftarrow \emptyset$
7: $\quad \texttt{tempActions} \leftarrow \emptyset$
8: $\quad \texttt{tempTransitions} \leftarrow \emptyset$
9: $\quad$ **for all** $s \in S$ **do**
10: $\quad\quad S_{\text{TEMP}}, A_{\text{TEMP}}, Tr_{\text{TEMP}} \leftarrow \texttt{getValidGroundings}(\texttt{abstractPlan}_h, s)$
11: $\quad\quad \texttt{tempStates} \leftarrow \texttt{tempStates} \cup S_{\text{TEMP}}$
12: $\quad\quad \texttt{tempActions} \leftarrow \texttt{tempActions} \cup A_{\text{TEMP}}$
13: $\quad\quad \texttt{tempTransitions} \leftarrow \texttt{tempTransitions} \cup Tr_{\text{TEMP}}$
14: $\quad$ **end for**
15: $\quad S \leftarrow S \cup \texttt{tempStates}$
16: $\quad A \leftarrow A \cup \texttt{tempActions}$
17: $\quad Tr \leftarrow Tr \cup \texttt{tempTransitions}$
18: $\quad S, A, Tr, R \leftarrow \texttt{setGoalBasedRewards}(S, A, T, \texttt{goal})$
19: **end for**
20: **return** S, A, Tr, R

or abstract).

- **line 9-14:** Given an expert within the abstract plan, we have to take into account all prior generated states as inputs. Method `getValidGroundings`, as introduced before, is called to get all correctly grounded states and actions for an expert and a state. The outcomes are added to respective temporary state and action sets, as they should only be added to the eventual state and actions after we finished step h.
- **line 15-20:** The enriched state and action spaces as well as the transition and reward functions are finally returned and can be used for planning. The reward function, analogous to the abstract case (see Equation 9.1), is defined based on the goal state (i.e. `goal`) where a dummy action is added from goal state to goal state with reward of one.

Complexity Analysis 9.1. *The algorithm has to be executed for all experts per state, which is repeated H times at maximum, with H a bound for the number of steps. Although several semantic experts might have to be executed more than once – e.g. the Brain Mask Generator expert – one can confidently set a bound on the number of steps of the semantic task. As a result, the complexity is within* $O(|E||S|H)$. *The number of steps* $|S|$ *is dependent on the algorithm, which – when unconstrained – produces* $2^{|D|}$ *states with D the set of decision candidates (e.g. headscan resources in a state). That is, in the worst case a single bag results in all combinations of lengths* $1,\ldots,|D|$. *The overall complexity is thus within* $O(|E||S|2^{|D|}H)$.

Note that the algorithm and its complexity depict the most general case where we do not have any knowledge about the resources of respective bags. As LAPIs have to make the relationship from inputs to outputs *explicit*, we can easily reduce (and in most cases exactly determine) the involved resources, thus eliminating factor $2^{|D|}$.

Time Performance Evaluation

We now evaluate the suitability of the semantic expert concept in terms of the execution time of the resulting LAPIs.

Setup: All experts were accessed through the MITK library (as summarized in Table 9.1). MITK offers access to experts via standard command line calls, where short textual descriptions for available parameters exist. For the lifting process, we worked closely with domain experts (i.e. developers) of MITK to correctly wrap the respective experts and only expose parameters which are needed for eventually generating TPMs. For the implementation of the semantic experts, we used JavaEE with JAX-RS as well as Apache Jena [10]. The resulting semantic image processing experts have been deployed in a virtual machine with the following specifications: QEMU Virtual CPU Version 0.9.1, 2266,796 MHz, 4 GB RAM, Ubuntu 12.04.4 LTS.

[10] `https://jena.apache.org/` (*accessed on 05/01/2018*)

Experiment	Execution time (in seconds)	Valid result
Local	592.48	Yes
Semantic	803.26	Yes

Table 9.2: Evaluation results for local versus semantic expert executions.

To compare the runtimes, we first manually executed the experts as available prior to the lifting process, i.e. on a local machine via command line tools of MITK. We then ran each semantic expert with respective RDF graph inputs as gathered by the semantic agent. The inputs where gathered via the *XNAT* RDF wrapper within the presented Web architecture (see Figure 9.2) and fed back into the platform after each step.

We used two test patient samples with two headscans (in NRRD format), where available background information consisted of an atlas image (in NRRD format) as well as an atlas mask (in NRRD format). We averaged the time needed for processing both headscans.

Results: Table 9.2 shows the resulting time needed to run each (semantic-) expert in seconds (s).

The TPM generation process was successful for using the descriptions of the individual semantic image processing experts and an initial implementation of LD-Fu without an abstract semantic planner. The semantic (i.e. automatic) execution of the semantic experts by the proposed semantic agent produced the same (and thus correct) results as the manual local execution. While the resulting overhead for the semantic TPM task is about 210 seconds, this does not impede the diagnostic process of radiologists as the TPM can be generated in advance. The overhead is mainly produced by downloading and uploading the required and produced files from and to *XNAT*.

We continue to evaluate the outputs of the proposed abstract semantic planner.

Semantic Planning Evaluation

We first evaluate our abstract semantic planner in terms of its ability to exclusively find semantic experts needed for the abstract semantic TPM task. We then generalize the results to grounded semantic planning for grounded semantic TPM tasks.

Semantic Abstract Planning Setup: The abstract semantic planner instantiates an infinite-horizon MDP $M_{\text{SEP}}^{\text{ABSTR}}$ and automatically constructs $Tr_{\text{SEP}}^{\text{ABSTR}}$ and $R_{\text{SEP}}^{\text{ABSTR}}$ based on available semantic experts, the TPM goal and given patient- as well as background information. Consider a grounding for the semantic Brain Mask Generation expert and an abstract goal describing a TPM (see Listing 9.7) with all paths to the TPM being possible. Besides the 6 semantic experts involved in the TPM process, the action space consists of 2 semantic phase recognizers of the phase recognition scenario we present in Section 9.2.3. For the proof-of-concept, we use discount factor of $\lambda = 0.9$ and perform value iteration (see Algorithm 1) for a given patient with two headscans (in NRRD format), additional segmentation for Robust Brain Nor-

malization (in NRRD format) and background information consisting of an atlas image (in NRRD format) as well as an atlas mask (in NRRD format). Listing 9.8 summarizes the start state.

```
@prefix spp:    <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Property-3A>.
@prefix spc:    <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Category-3A>.
@prefix sawsdl: <http://www.w3.org/ns/sawsdl#>.
@prefix xnat:  <http://aifb-ls3-vm2.aifb.kit.edu:8080/xnatwrapper/id/>.
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix dc:     <http://purl.org/dc/elements/1.1/>.
@prefix sparql: <http://www.w3.org/TR/rdf-sparql-query/#>.
@prefix sep:    <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/sep#>.

xnat:start1   rdf:type      sep:GroundedState, sep:StartState;
              sawsdl:modelReference
        [ a           sep:GoundedCondition;
          rdf:value   """{
            xnat:headscan1     rdf:type       spc:Headscan;
                               dc:format      spc:NRRD;
                               spp:manualSegmentation
                                              xnat:seg1;
                               spp:patient    xnat:patient1.
            xnat:headscan2     rdf:type       spc:Headscan;
                               dc:format      spc:NRRD;
                               spp:manualSegmentation
                                              xnat:seg2;
                               spp:patient    xnat:patient1.
            xnat:patient1      rdf:type       spc:Patient.
            xnat:brainAtlasImg rdf:type       spc:BrainAtlasImage;
                               dc:format      spc:NRRD.
            xnat:brainAtlasMsk rdf:type       spc:BrainAtlasMask;
                               dc:format      spc:NRRD.
            xnat:seg1          rdf:type       spc:ManualSegmentation;
                               dc:format      spc:NRRD;
                               spp:patient    xnat:patient1.
            xnat:seg2          rdf:type       spc:ManualSegmentation;
                               dc:format      spc:NRRD;
                               spp:patient    xnat:patient2.
                     }"""^^sparql:GraphPattern].
```

Listing 9.8: Start state of a semantic TPM task.

Semantic Abstract Planning Results: When running value iteration on $M_{\text{SEP}}^{\text{TPM}}$, we derive converged state values after 6 iterations, which are summarized in Table 9.3.

States with values greater than zero depict preconditions of semantic experts which have to be executed to reach the goal (except for the *dummy* action). As a result, value iteration only discloses state values greater than zero if the latter potentially enable to reach the goal state, where the semantic phase recognition experts are assigned state values of zero. The respective output of the semantic abstract planner is depicted in Listing 9.9.

State number	State description	State value
1	Brain Mask Generation	0.32805
2	Standard Normalization	0.3645
3	Robust Normalization	0.3645
4	Batched Folder Registration	0.405
5	Tumor Segmentation	0.45
6	Map Generation	0.45
7	Dummy	1.0
8	Rule-based Phase Recognition	0.0
9	ML-based Phase Recognition	0.0

Table 9.3: Evaluation results for abstract semantic planning.

```
@prefix spc:  <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Category-3A>.
@prefix xnat: <http://aifb-ls3-vm2.aifb.kit.edu:8080/xnatwrapper/id/>.
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix exp:  <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/>.
@prefix sep:  <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/sep#>.

_:pbag        rdf:type      rdf:Bag, sep:PolicyBag.

_:pol1        rdf:type      rdf:Seq;
              rdfs:member   _:pbag;
              rdf:_1        exp:BMG;
              rdf:_2        exp:BN;
              rdf:_3        exp:BR;
              rdf:_4        exp:TS;
              rdf:_5        exp:MG.

_:pol2        rdf:type      rdf:Seq;
              rdfs:member   _:pbag;
              rdf:_1        exp:BMG;
              rdf:_2        exp:RBN;
              rdf:_3        exp:BR;
              rdf:_4        exp:TS;
              rdf:_5        exp:MG.

exp:BMG      rdf:type      sep:SemanticExpert, spc:BrainMaskGeneration.
exp:BN       rdf:type      sep:SemanticExpert, spc:BrainNormalization.
exp:RBN      rdf:type      sep:SemanticExpert, spc:RobustBrainNormalization.
exp:BR       rdf:type      sep:SemanticExpert, spc:BrainRegistration.
exp:TS       rdf:type      sep:SemanticExpert, spc:TumorSegmentation.
exp:TPM      rdf:type      sep:SemanticExpert, spc:MapGeneration.

xnat:goal1   rdf:type      sep:State, sep:GoalState, sep:AbstractState.
xnat:start1  rdf:type      sep:State, sep:StartState, sep:GroundedState.
```

Listing 9.9: Abstract semantic TPM plan.

Grounded Semantic Planning Setup: For grounded semantic planning, we reuse an abstract semantic plan to gradually ground the preconditions of semantic experts of each step. Assuming the goal in Listing 9.7 and respective start state (see Listing 9.8) as well as the abstract plan, the grounded semantic planner starts by generating semantic expert executions for $h = 1$, i.e. Brain Mask Generation. Given the start- and goal state, a number of semantic Brain Mask Generation expert executions are possible, namely executing the semantic expert once by using `xnat:headscan1`, once by using `xnat:headscan2`, or twice by using `xnat:headscan1` and `xnat:headscan2` individually. By running Algorithm 4 on the TPM start state, abstract plan, available experts and goal state, we instantiate $M_{\text{SEP}}^{\text{GRND}}$ for TPM with discount factor $\lambda = 0.9$. The MDP consists of all possible (abstract) semantic expert executions for all steps, where for each step each expert might be executed multiple times. We run value iteration on $M_{\text{SEP}}^{\text{GRND}}$ to find an appropriate plan.

Grounded Semantic Planning Results: By running value iteration on $M_{\text{SEP}}^{\text{GRND}}$, all states with positive values support to reach the goal. The actions to reach these states are state configurations for experts which need to be executed. For our TPM evaluation case in step $h = 1$, we need to take two actions, namely executing the semantic Brain Mask Generation expert for both headscans individually. Listing 9.10 comprises the output of the grounded semantic planner for step $h = 1$.

```
@prefix spp:    <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Property-3A>.
@prefix spc:    <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Category-3A>.
@prefix sawsdl: <http://www.w3.org/ns/sawsdl#>.
@prefix xnat:   <http://aifb-ls3-vm2.aifb.kit.edu:8080/xnatwrapper/id/>.
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#>.
@prefix sparql: <http://www.w3.org/TR/rdf-sparql-query/#>.
@prefix dc:     <http://purl.org/dc/elements/1.1/>.
@prefix exp:    <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/>.
@prefix sep:    <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/sep#>.

_:ebag        rdf:type      rdf:Bag.
exp:BMG       rdf:type      sep:SemanticExpert, spc:BrainMaskGeneration;
              rdfs:member   _:ebag.
xnat:grounding1
              rdf:type      spc:ExpertExecution;
              sep:expert    exp:BMG;
              sawsdl:modelReference
        [ a           sep:GoundedCondition;
          rdf:value   """{
            xnat:headscan1     rdf:type       spc:Headscan;
                               dc:format      spc:NRRD.
                               spp:patient    xnat:patient1.
            xnat:patient1      rdf:type       spc:Patient.
            xnat:brainAtlasImg rdf:type       spc:BrainAtlasImage;
                               dc:format      spc:NRRD.
            xnat:brainAtlasMsk rdf:type       spc:BrainAtlasMask;
                               dc:format      spc:NRRD.
                       }"""^^sparql:GraphPattern].
xnat:grounding2
```

```
31 ||               rdf:type      spc:ExpertExecution;
32 ||               sep:expert    exp:BMG;
33 ||               sawsdl:modelReference
34 ||         [ a           sep:GoundedCondition;
35 ||           rdf:value   """{
36 ||             xnat:headscan2     rdf:type       spc:Headscan;
37 ||                                dc:format      spc:NRRD.
38 ||                                spp:patient    xnat:patient1.
39 ||             xnat:patient1      rdf:type       spc:Patient.
40 ||             xnat:brainAtlasImg rdf:type       spc:BrainAtlasImage;
41 ||                                dc:format      spc:NRRD.
42 ||             xnat:brainAtlasMsk rdf:type       spc:BrainAtlasMask;
43 ||                                dc:format      spc:NRRD.
44 ||                       }"""^^sparql:GraphPattern].
45 ||
46 || xnat:goal1    rdf:type      sep:State, sep:GoalState, sep:AbstractState.
47 || xnat:start1   rdf:type      sep:State, sep:StartState, sep:GroundedState.
```

Listing 9.10: Output of the grounded semantic planner for the step $h = 1$.

In general, the process needs to be repeated for each step $h \in 1 \ldots, H$ as we need to learn expert weights based on the respective state groundings. For TPM without learning, we could find an appropriate grounded plan by planning once on the grounded start state and use the BGP state configurations for later steps. We will deal with a different case in Section 9.3 for NERD.

9.2.3 Semantic Surgical Phase Recognition

In surgical phase recognition (see Section 2.2), one aims to predict the current surgical phase by various sensory information – in our scenario the latter are restricted to activity triples consisting of the used surgical instrument, the performed action and the treated human organ. We base our application and evaluation of phase recognition on two experts, i.e. (i) An expert based on manually defined rules in the Semantic Web Rule Language (SWRL) for a domain ontology [63] and (ii) a supervised learning-based expert (referred to as ML-based phase recognition expert), fitting a random forest model [23] for multiclass classification with annotated surgeries as training data (i.e. surgical activities labeled with the correct surgical phase). We first present the lifting process for semantic phase recognition experts and then proceed with our grounded semantic learner.

Semantic Surgical Phase Recognition

We lifted both phase recognition experts to semantic experts, and defined their inputs and outputs in terms of semantic pre- and postconditions. See Listing 9.11 for the preconditions of the semantic ML-based phase recognition expert and note that all semantic descriptions for Surgical Phase Recognition experts are presented in Appendix A.

The semantic phase recognition experts need to be initialized with a separate laparoscopic ontology with concepts for the surgical setting. This is due to the implementation of the expert

as well as the lifting to the semantic expert. The ML-based phase recognizer, in addition, has to be trained with samples.

```
@prefix spp:   <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Property-3A>.
@prefix spc:   <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Category-3A>.
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

?col              rdf:type        rdf:Bag;
                  rdf:li          ?trainingSample.

?trainingSample   rdf:type        spc:Surgery.

?ontology         rdf:type        spc:Ontology.

?event            rdf:type        spc:SurgicalEvent;
                  spp:instrument  ?instrument;
                  spp:action      ?action;
                  spp:structure   ?structure.

?instrument       rdf:type        spc:Instrument.
?action           rdf:type        spc:Action.
?structure        rdf:type        spc:TreatedStructure.
```

Listing 9.11: Preconditions of the semantic ML-based expert.

The postconditions (see Listing 9.12) state that results have to be typed as `spc:Phase`, which ensures that only modeled phases occur and recognized phases belong to the event of the preconditions.

```
@prefix spp:   <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Property-3A>.
@prefix spc:   <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Category-3A>.
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix dc:    <http://purl.org/dc/elements/1.1/>.

?phase            rdf:type        spc:Phase;
                  spp:value       ?value.

?event            rdf:type        spc:SurgicalEvent;
                  spp:instrument  ?instrument;
                  spp:action      ?action;
                  spp:structure   ?structure.
                  spp:phase       ?phase.
```

Listing 9.12: Postconditions of the semantic ML-based expert.

A Grounded Semantic Learner for Surgical Phase Recognition.

Our grounded semantic learner is based on methods proposed in Part II and is able to weight the competing semantic phase recognition experts to eventually choose the best result. As it only works for semantic phase recognition experts, we define less general preconditions. Listing 9.13 depicts the resulting rule for executing the grounded semantic learner. It only assumes available semantic experts chosen by an abstract semantic planner and further triples about the

activities. Note that we can elegantly define the generality of the grounded semantic learner based on the concept types we use. If it was able to optimally choose among two or more image processors as well, we could easily express that via preconditions and postconditions.

```
@prefix sep:    <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/sep#>.
@prefix spc:    <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Category-3A>.
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix httpm:  <http://www.w3.org/2011/http-methods#>.
@prefix exp:    <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/>.

{
?col          rdf:type       rdf:Bag;
              rdf:li         ?exp.
?exp          rdf:type       spc:PhaseRecognitionExpert.
?grounding    rdf:type       sep:State.
} => {
_:a   http:mthd          httpm:POST;
      http:requestURI    exp:groundedlearner;
      http:body
    {
        ?col         rdf:type  rdf:Bag;
                     rdf:li    ?exp.
        ?exp         rdf:type  spc:PhaseRecognitionExpert.
        ?grounding   rdf:type  sep:State.
    }.
}.
```

Listing 9.13: LD-Fu rule for executing the grounded semantic learner.

The grounded semantic learning component assesses the performance of a given semantic phase recognizer based on training samples close to the state (i.e. decision candidate for the phase recognition scenario). Our initial approach is a batch learning variant of Single-Avg as introduced for evaluation in Chapter 7 using single-expert meta dependencies (as introduced in Sec 5.3) without further weightings of the latter. To get batch training samples, one trains the ML-based phase recognizer on parts of the training sets (i.e. all surgeries but one) and predicts on the remaining surgery. The process is repeated until we have predictions for all surgeries. For the rule-based phase recognizer, one simply has to execute the latter on all left out surgeries, as no training takes place. Finally, the optimal hypothesis is generated based on the weighted combination of available semantic experts.

Grounded Semantic Learner Evaluation

We evaluate the performance of the grounded semantic learner for surgical phase recognition by running the automation architecture and quantifying the outcome optimization performance of the resulting grounded phase recognition SEP.

Setup:

Kernels: We define and exclusively use a straightforward kernel for surgical activities based on the exact pairwise match of structure, instrument and/or action. If for two activities the current triples plus their three predecessors match, the activities get assigned a similarity of

Expert	Surgery 1	Surgery 2	Surgery 3	Surgery 4	Surgery 5
SWRL	0.9315	0.7753	0.89	0.8137	0.7241
ML-based	0.9062	0.6635	0.9032	0.4484	0.6383

Table 9.4: Baseline performances of phase recognition experts.

Expert	Surgery 1	Surgery 2	Surgery 3	Surgery 4	Surgery 5
Batch-Local	0.9332	0.7786	0.9180	0.7782	0.7238

Table 9.5: Evaluation results for the grounded semantic learner.

1. The similarity linearly decreases to 0.25 if only the current activity triples match and has value 0 otherwise. The kernel value of two states s_1 and s_2 is thus high if the respective current and past activity triples of the states are identical (i.e. instrument, action and structure of s_1 at time t (written as s_1^t) are identical to instrument, action and structure of s_2 at time t, with t decreasing one by one). The kernel is defined in Equation 9.2.

$$\kappa(s_1,s_2) = \begin{cases} 1 & x = 0,1,2,3 : s_1^{t-x} = s_2^{t-x} \\ 0.75 & x = 0,1,2 : s_1^{t-x} = s_2^{t-x} \wedge s_1^{t-3} \neq s_2^{t-3} \\ 0.5 & x = 0,1 : s_1^{t-x} = s_2^{t-x} \wedge s_1^{t-2} \neq s_2^{t-2} \\ 0.25 & s_1^t = s_2^t \wedge s_1^{t-1} \neq s_2^{t-1} \\ 0 & s_1^t \neq s_2^t \end{cases} \tag{9.2}$$

Experts: The baseline performances of the two available experts for phase recognition are summarized in Table 9.4

Evaluation measures: We performed five-fold cross validation, where we trained on four surgeries and predicted on the residual one. The evaluation metric we used was precision, which is defined as average of correct predictions for the test set. Our exemplary batch expert weight learning approach for the evaluation is denoted Batch-Local.

Results: The results are summarized in Table 9.5.

The resulting precision of the grounded semantic learner either outperformed the best phase recognizer or was able to compete. More importantly, the grounded phase recognition SEP was successfully automated, as the grounded semantic learner and the respective semantic phase recognition experts were correctly executed.

Based on grounded semantic learning for surgical phase recognition and abstract planning for TPM, we can automate the respective semantic tasks. Until now, we only explored grounded semantic learning for a single-step semantic task, where the problem of considering

future impacts when taking decisions is not prevalent. As a consequence, a central challenge for grounded semantic planning did not occur, namely dealing with a large amount of conflicting groundings, due to exchangeable semantic experts.

To this end, we present semantic meta components for a NLP scenario, where prior mentioned complexities are available. Here, the grounded semantic learning technique can be completely reused, but the semantic description for the respective grounded semantic learners has to be adapted. However, a novel semantic meta component for grounded semantic planning has to be developed, which deals with conflicting state configurations.

9.3 The Named Entity Recognition & -Disambiguation Scenario

In NERD, one deals with ambiguity in unstructured text, mostly available on the Web. As textual mentions of entities often are ambiguous, it often remains unclear which real-world entity is referred to. It is thus important to find links between such mentions and entity resources, as available in KBs.

For the semantic NERD task, we reuse all experts of Chapter 7. We first present the semantic descriptions for NER and NED experts. Based on the latter, we then discuss how to integrate semantic meta components for grounded semantic learning in order to weight semantic NER- and NED experts. We finally discuss grounded semantic planning, where we extend our TPM approach to deal with conflicting hypotheses of available NER experts.

9.3.1 Semantic Descriptions for Semantic NER & NED Experts

Numerous NER, NED and NERD experts are available as Web services and provide rich descriptions of inputs and outputs. While inputs for NER are confined to text, outputs usually consist of mappings of tokens or token sequences to named entities. These mappings consist of start- and end positions of tokens as well as the predicted named entity type (which we do not need for solving NERD). Depending on the implementation of the expert and the Web service, the output might also provide a *confidence measure* for each mapping. NED Web services usually require an annotated version of the text, where a specific syntax needs to be used. The produced output consists of mappings from a token or token sequences to KB resources, the position of the token or token sequence and, if available, a confidence measure for each prediction. NER and NED Web services might provide further parameters about their functionality.

For modeling the preconditions and postconditions, we reuse the NLP Interchange Format (NIF) [11], a RDF vocabulary for describing both Web services and datasets for several NLP tasks. The preconditions for semantic NER experts are depicted in Listing 9.14. They confine text inputs to be sentences or complete paragraphs with adequate NIF annotations. Parameters specific to the respective expert can be set as well, but remain optional. For predicates and

[11] `http://persistence.uni-leipzig.org/nlp2rdf/` (*accessed on 05/01/2018*)

concepts not provided by NIF, we use namespace `sepnlp:` to refer to a novel, extending ontology.

```
@prefix nif:    <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#>.
@prefix sepnlp: <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/sepnlp#>.
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

?text             rdf:type             ?type.

FILTER (?type = nif:Sentence || ?textType = nif:Paragraph).

OPTIONAL {
?parameterConfig  rdf:type                sep:ParameterConfig;
                  sepnlp:parameter        ?parameter;
                  sepnlp:parameterValue   ?parameterValue.
}
```

Listing 9.14: Preconditions of semantic NER experts.

The preconditions of semantic NED experts are depicted in Listing 9.15. Inputs are NER annotations with information about start- and end indexes of a named entity candidate (i.e. a token or a token sequence). Annotations are collected in a `rdf:Bag`, where two annotations must not overlap with respect to their tokens. We model this fact by requiring triples with predicate `sepnlp:disjunct` which can be directly inferred by a respective LD-Fu rule [12].

```
@prefix nif:    <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#>.
@prefix sepnlp: <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/sepnlp#>.
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#>.

?col              rdf:type             rdf:Bag;
                  rdf:li               ?annotation1, ?annotation2.

?annotation1      sepnlp:disjunct      ?annotation2.

?annotation1      rdf:type             sepnlp:Annotation;
                  nif:isString         ?text1;
                  sepnlp:token         ?token1;
                  sepnlp:isEntity      "true"^^xsd:boolean.

?token1           rdf:type             sepnlp:Token;
                  nif:anchorOf         ?mention1;
                  nif:beginindex       ?start1;
                  nif:endIndex         ?end1;
                  nif:referenceContext ?sentence.

?annotation2      rdf:type             sepnlp:Annotation;
                  nif:isString         ?text2;
                  sepnlp:token         ?token2;
                  sepnlp:isEntity      "true"^^xsd:boolean.

?token2           rdf:type             sepnlp:Token;
```

[12] Note that one could also model SPARQL filters to directly check for disjunction.

```
                    nif:anchorOf          ?mention2;
                    nif:beginindex        ?start2;
                    nif:endIndex          ?end2;
                    nif:referenceContext ?sentence.
OPTIONAL {
?parameterConfig  rdf:type              sepnlp:ParameterConfig;
                    sepnlp:parameter      ?parameter;
                    sepnlp:parameterValue
                                          ?parameterValue.
}
```

Listing 9.15: Preconditions of semantic NED experts.

Finally, the postconditions of NED experts are stipulated in Listing 9.16. It stipulates that the annotation used as NED expert input is required to be enriched with a disambiguated resource.

```
@prefix nif:    <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#>.
@prefix sepnlp: <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/sepnlp#>.
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#>.

?col              rdf:type              rdf:Bag;
                    rdf:li                ?annotation1.

?annotation1      rdf:type              sepnlp:Annotation;
                    nif:isString          ?text;
                    sepnlp:token          ?token1.
                    sepnlp:isEntity       "true"^^xsd:boolean;
                    sepnlp:resource       ?resource.

?token1           rdf:type              sepnlp:Token
                    nif:anchorOf          ?mention;
                    nif:beginindex        ?start;
                    nif:endIndex          ?end;
                    nif:referenceContext ?text.
OPTIONAL {
?parameterConfig  rdf:type              sepnlp:ParameterConfig;
                    sepnlp:parameter      ?parameter;
                    sepnlp:parameterValue
                                          ?parameterValue.
}
```

Listing 9.16: Postconditions of semantic NED experts.

Every NER and NED expert considered in the scenarios was wrapped as semantic expert. We modeled the descriptions with NLP domain experts and developers of NER and NED experts, and made them available via the respectively developed LAPIs. Note that all semantic descriptions for NER and NED experts are presented in Appendix A. The KB comprises texts, semantic NER- and NED experts and resulting annotations by executing the latter.

9.3.2 A Grounded Semantic Planner for NERD

Similar to the TPM scenario, precondition groundings of semantic NED experts might produce multiple state configurations, as multiple token-/named entity mappings have been chosen. To give an example, consider the following tweet:

`Michael Jordan is a famous basketball player.`

There is no ambiguity for NER and, thus, all available semantic experts take the full text as input. Given the annotation candidates `Michael`, `Jordan`, `Michael Jordan` and `basketball player` one now has to construe valid annotations for NED which causes two challenges.

(i) We have to express that `Michael Jordan` and `Michael` must not be used for one text annotation due to their overlap.

(ii) It quickly becomes intractable to try out (i.e. query experts with) all valid annotations for NED and thus budgets are imposed.

We partially approached problem (i) by modeling the preconditions of NED experts. Given a bag of annotations, the tokens of two annotations must not overlap (i.e. predicate `sepnlp:disjunct` relates them) with regards to their start and end indexes. By now, we only dealt with simple bag constructs where a single bag element needs to be modeled in the respective pre- and postconditions of semantic experts. We thus extend Algorithm 4 to allow for modeling more complex collections, where relations among elements are taken into account. The algorithm is summarized subsequently (Algorithm 5).

Algorithm 5 `getValidGroundingsExtended`(e,s)

```
collector ← ∅
for all bag ∈ Preconditions_e do
    configurations ← ∅
    condition ← getCondition(bag)
    q ← getQueryExtended(bag)
    candidates ← queryState(q, s)
    for all n ∈ { 2,...,|candidates| do
        unfiltered ← subsets(n, candidates)
        configurations ← configurations ∪ filter(unfiltered, condition)
    end for
    collector ← collector ∪ configurations
end for
S,A ← getGroundings(configurations, Preconditions_e, s)
return S,A
```

The altered details of the algorithm are as follows [13]:

- **Line 4:** The condition for the bag is retrieved, which filters the modeled relations among all pairwise triples of the bag.
- **Line 5:** Other than in the standard case with a single element in the bag, we now need to actively select the element to retrieve all unfiltered resources.
- **Line 8:** Similar to the original algorithm, we first retrieve all possible configurations (in set `unfiltered`) regardless of the condition.
- **Line 9:** The configurations are filtered according to the condition (in method `filter`, which has to be tested for all pairs of resources. The rest of the algorithm is equivalent to Algorithm 3.

Example 9.1 illustrates the procedure for a piece of text with two available annotations. For grounded states in the KB, we use namespace `sepkb:`.

Example 9.1 (Grounding for NED)**.** *Given state s with text* `Michael Jordan is a famous basketball player` *and decision candidates* $^1s =$`Michael Jordan` *and* $^2s =$`basketball` *and NED expert e with preconditions as shown in Listing 9.15, we need call Algorithm 4 with Algorithm 5 to ground the available bag of annotations with respect to finding configuration decision candidates* ^1s *and* ^2s*. Listing 9.17 shows the resulting condition of a LD-Fu rule, where resource* `sepkb:token1` *denotes* ^1s *and resource* `sepkb:token2` *denotes* ^2s*.*

```
1  @prefix nif:    <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#>.
2  @prefix sepnlp: <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/sepnlp#>.
3  @prefix sepkb:  <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/sepkb/id/>.
4  @prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
5  @prefix xsd:    <http://www.w3.org/2001/XMLSchema#>.
6
7  ?col               rdf:type             rdf:Bag;
8                     rdf:li               ?annotation1,
9                                          ?annotation2.
10
11 ?annotation1       rdf:type             sepnlp:Annotation;
12                    nif:isString         ?text;
13                    sepnlp:token         sepkb:token1;
14                    sepnlp:isEntity      "true"^^xsd:boolean.
15
16 ?annotation2       rdf:type             sepnlp:Annotation;
17                    nif:isString         ?text;
18                    sepnlp:token         sepkb:token2;
19                    sepnlp:isEntity      "true"^^xsd:boolean.
20
21 sepnlp:token1      rdf:type             sepnlp:Token.
```

[13] For the general description see the details of Algorithm 3

```
22 sepnlp:token2     rdf:type             sepnlp:Token.
23
24 sepkb:token1      sepnlp:disjunct      sepkb:token2.
25
26 OPTIONAL {
27 ?parameterConfig  rdf:type             sepnlp:ParameterConfig;
28                   sepnlp:parameter     ?parameter;
29                   sepnlp:parameterValue
30                                        ?parameterValue.
```

Listing 9.17: LD-Fu rule for NED expert and exemplary state configuration.

In addition to restrictions, we cannot simply find a single trajectory to the goal state, as a number of NER or NED experts often disagree and propose conflicting decision candidates. The problem becomes more severe with an increasing number of steps. The resulting upper bound for the number of NED executions corresponds to all possible n-grams (with $n = 1,\ldots,|\texttt{tokens(s)}|$) of a state text. After several semantic NER experts haven been executed and a grounded semantic learner has weighted their outputs, we thus have to restrict the number of configurations (S^{CONF}) we want the grounded semantic planner to build.

We propose a *sampling* approach which generates probability distributions based on expert weights $w_e({}^d s)$ of the grounded semantic learner and draws a predefined number of samples (i.e. expert budget C_{EXP} of SEPs). We iteratively cluster all named entity candidates ${}^d s \in s$ until we derive valid named entity assignments for a configured state. More specifically, we first create clusters for each named entity candidate ${}^d s$ to decide if we consider it for a sampling round. As there might be several conflicting named entity assignments, we use the retrieved samples to create new clusters for each sampled candidate ${}^d s$ plus its dependent named entity candidates (i.e. the candidates which overlap ${}^d s$ in s). The sampling process continues until a valid annotation is reached, i.e. we have a mapping from tokens to named entities which do not overlap.

Before presenting the sampling algorithm, we define the resulting Grounded SEP for NERD, which extends Definition 9.3.

Definition 9.4 (Extended Reduction of Grounded SEPs to Infinite-horizon MDPs)**.** *Given infinite-horizon MDP M_{SEP}^{GRND} (see Definition 9.1) and the general properties of the latter (see Definition 9.4), M_{SEP}^{GRND} is instantiated by calling* $\texttt{initializeGroundedPlanning}(s^1_{RDF}, \pi_{ABSTR}, E, s^{H+1}_{BGP})$ *(Algorithm 4) with the available start state s^1_{RDF}, learned abstract policy π_{ABSTR}, expert set E and available goal s^{H+1}_{BGP}.*

For `initializeGroundedPlanning`*, we use* `getValidGroundingsExtended` *instead of* `getValidGroundings`*, where the resulting state space is sampled with* $\texttt{sample}(S^h)$.

The procedure is formalized and summarized in algorithm 6, having the set of candidate named entities s^{CAND} as input.

We now explain the algorithm in detail:

Algorithm 6 `sample`(s^{CAND})

```
1: clusters ← ∅
2: for all ^d s ∈ s^CAND do
3:     clusters ← clusters ∪ (^d s, ¬^d s)
4: end for
5: merge ← true
6: while merge do
7:     for all cluster ∈ clusters do
8:         P_CLUSTER ← getProbabilityDistribution(cluster)
9:         cluster ← draw(P_CLUSTER)
10:    end for
11:    merge ← checkValidity(clusters) // false if assignment is valid
12:    if merge then
13:        s^DEP = invalidOverlaps(clusters) // increasingly ordered by size
14:        clusters ← ∅
15:        for all ^d s ∈ s^DEP do
16:            if ¬inCluster(^d s, clusters) ∧ ∀^c s ∈ ^d s.DEPS : ¬inCluster(^c s, clusters)
   then
17:                clusters = clusters ∪ (^d s ∪ ^d s.DEPS)
18:            else
19:                if ¬inCluster(^d s, clusters) then
20:                    cluster = getCluster(^d s, clusters)
21:                    cluster = cluster ∪ ^d s
22:                end if
23:            end if
24:        end for
25:    end if
26: end while
27: s ← transformToState(clusters)
28: return s
```

- **Line 1:** The set of clusters is initialized, where the latter will constitute the eventual sampled state. That is, each cluster within the set of clusters should contain a single decision candidate which must not overlap with another cluster's decision candidate.

- **Line 2-3:** A cluster is created for each decision candidate, also containing its negation. Each cluster thus initially depicts the choice for choosing the decision candidate or not.

- **Line 5:** The merge variable is initialized as true. It depicts if the available cluster has to be merged, as their decision candidates constitute conflicts.

- **Line 6-9:** Until no valid set of clusters could be found, we need to draw a decision candidate from each cluster. $P_{\text{CLUSTER}} : S \rightarrow [0,1]$ maps decision candidates to probability values and constitutes a probability distribution, where $\sum_s P_{\text{CLUSTER}}(S = s) = 1$.

- **Line 11:** The validity of clusters is checked in terms of textual overlap of decision candidates of two clusters as well as cluster size (as the goal of the algorithm is to end up with clusters of length one). The method returns false if all clusters are valid and no merging has to follow.

- **Line 12-13:** If merging needs to happen, we need to gather all invalid overlaps (i.e. conflicts) of clusters. s^{DEP} is the resulting set with all decision candidates of the available clusters, where ${}^{d}s \in s^{\text{DEP}}$ store links to all conflicting decision candidates ${}^{c}s \in {}^{d}s.\text{DEPS}$ in set decision candidate-dependent set DEPS. The latter is ordered by the length of dependencies, as otherwise n-grams might cause to neglect numerous decision candidates made of subsets of the n-gram (see Complexity Analysis 9.2 for an example).

- **Line 14-24:** After re-initializing the clusters set, we iterate through all currently chosen decision candidates and use conflict link sets to build novel clusters. More specifically, if a decision candidate and all its conflicts (i.e. $({}^{d}s \cup {}^{d}s.\texttt{DEPS})$) are not yet in a cluster, they constitute a novel cluster. Otherwise, we add the current decision candidate to an available cluster, if the former was not already assigned as dependency in a prior iteration. The dependent decision candidates will then be added to or form novel clusters in subsequent rounds if they are not assigned yet.

- **Line 27-28:** Finally, we have to transform the clusters into the configured state by simply gathering all decision candidates from all available clusters. The configured state is then returned.

The algorithm takes into account all possible state configurations, as one biases the sampling process when not resolving overlaps of decision candidates, e.g. when ${}^{1}s \leftrightarrow {}^{2}s \leftrightarrow {}^{3}s \leftrightarrow \ldots \leftrightarrow {}^{d-1}s \leftrightarrow {}^{d}s$ holds, it yields sampling a single decision candidate of a single cluster (with all decision candidates in it). We now discuss the complexity of the algorithm.

Complexity Analysis 9.2. *The worst case happens due to decision candidate dependencies such that* ${}^{1}s \leftrightarrow {}^{d}s, {}^{2}s \leftrightarrow {}^{d}s, \ldots, {}^{d-1}s \leftrightarrow {}^{d}s$ *(caused, for example, by n-grams for textual decision candidates), as this might cause the while loop (lines 6-26) to sequentially conduct* $|D-1|$ *draws due to the ordering of the dependency sets (which can be done in* $O(|D|log|D|)$ *with sorting algorithms such as merge- or heap sort). The increasing ordering, as mentioned before, is essential to take into account decision candidates with smaller amounts of tokens if n-grams are available. The merge loop additionally iterates through all decision candidates, pointing to their dependencies (lines 15-24), causing quadratic complexity. The computational complexity of the algorithm is thus within* $O(|D| + |D|^2) = O(|D|^2)$.

9.3.3 Evaluation

We evaluate the grounded semantic planner by running the complete NERD SEP with abstract semantic planner as introduced in Section 9.2.2, grounded semantic learner as introduced in Section 9.2.3 and novel grounded semantic planner as introduced for the semantic NERD task in Section 9.3.2.

9.3.4 Setup

Kernels: We use the same setup in terms of kernels, densities and (semantic-) experts as described in our learning evaluation in Chapter 7. The only restrictions are the execution budgets $M^1 = M^2 = |E|$, where a single state is to be generated after each step.

Experts: We use the same experts as in our learning evaluation in Chapter 7.

Evaluation measures: We focus on outcome optimization of SEPs and, as lifted for surgical phase recognition (see Section 9.2.3), use the grounded semantic learner (see Section 9.2.2) we lifted based on Single-Avg (see Chapter 7), which uses single-expert meta dependencies to calculate expert weights. Other than for the MEP and EP evaluation, we are now in the batch learning setting, where we either re-train the grounded semantic learner at each query. While this significantly increases computational complexity, the results might be improved compared to the pure online learning case. The approach is, again, referred to as Batch-Single-Avg.

9.3.5 Results

The results are summarized in Table 9.6.

While the results for the batch learner are better than in the online case (see Table 7.6), the significant result of the evaluation is that the grounded semantic planner created grounded plans to correctly query the respective semantic experts.

	Microposts '14			Spotlight		
Approach	F1	Prec	Rec	F1	Prec	Rec
Batch-Single-Avg	0.51	0.74	0.4	0.4	0.5	0.37

Table 9.6: Evaluation results for the NERD SEP.

9.4 Discussion

We first discuss the generalizability of semantic meta components and semantic experts to new domains or tasks and then deal with the problem of different reward impacts for different semantic tasks.

9.4.1 On the Generalizability of Semantic Meta Components

The ability to generalize and reuse the semantic meta components we developed throughout this work is dependent on (i) the flexibility of the underlying meta component and (ii) its semantic description. (i) Our grounded semantic learner is built on a generic expert weight learning approach based on meta dependencies and not bound to NLP tasks. In addition, our abstract semantic planner is based on MDP planning and completely generalizable. (ii) The semantic descriptions we modeled are domain-dependent for both grounded semantic learning and abstract semantic planning, as they need to capture respective constraints for execution and correct functioning. The semantic descriptions for grounded semantic learners can, however, be automatically inferred from semantic experts, while semantic meta planners need to capture goal-dependent task parameters, such as the constraint of having one decision per token or token sequence for NERD.

To this end, one could relax the impact of the pre- and postconditions and shift the decision to grounded semantic learning components, i.e. model generic pre- and postconditions and learn their eligibility for a datapoint via feedback. This might be useful for generalizing the applicability of a semantic expert or for dealing with situations where few semantics are available.

Also, using the framework for novel tasks is dependent on the goal of the respective task, as semantic meta components as well as semantic experts have to be modeled accordingly. More specifically, if a novel task could theoretically be solved by available semantic experts in terms of their functionality, their semantic description might not capture all needed information of the task's goal. To this end, the respective LAPI might have to be adjusted, as extended semantic annotations might influence the underlying expert.

9.4.2 On Domain-dependent Impacts of Rewards

Grounded semantic learning entails to either learn R directly or approximate expert weights. We neglected the aspect that R might have different interpretations and impacts for different domains. While for NLP techniques applied to Web-based tasks choosing the highest estimated reward measured on past executions might not cause problems, it might have critical and ethical consequences for medical scenarios. Provenance metadata might have to be extended with more specific criteria for decision-making, where manually defined decision policies potentially need to be followed based on strict medical guidelines. Hence, a static semantic expert pipeline might be preferred for all data points, as it is trusted by physicians.

A related challenge is that institutions might differ in their perceptions of what conceptual provenance metadata is needed to ensure trustworthiness or that different guidelines are followed. Hence, reusing task outcomes published as Linked Data requires developing strategies to deal with uncertainties coming from missing information.

9.5 Summary

In this chapter, we presented applications and evaluations of SEPs in medical assistance as well as NLP. Based on our general Web automation architecture for medical assistance, we first dealt with the TPM use case, where we presented semantic experts as well as a abstract semantic planner and a grounded semantic planner to automatically discover them. Both semantic planners were developed as reduction of Abstract- or Grounded SEPs to infinite-horizon MDPs, which can be solved via standard MDP Planning algorithms. We evaluated the semantic experts in terms of time-efficiency with respect to the local execution of their wrapped functionality and were able to show that semantic expert concept is suitable for medical assistance experts. Our evaluations for semantic meta components additionally validated that we can automatically discover as well as compose expert pipelines for medical assistance.

We presented further semantic experts for Surgical Phase Recognition and developed a grounded semantic learner, which encapsulates a batch version of methods presented in Part II of this thesis. It exploited similarity measures defined on activity triples as well as single expert meta dependencies. The evaluation of the latter verified that respective semantic Phase Recognition experts were correctly executed and that the grounded semantic learners achieved robust and partially superior results.

We finally dealt with our NERD application, where we first presented semantic experts for NER and NED. While we could reuse the abstract semantic planner of the TPM scenario, we extended the grounded semantic planner with a state sampling technique as well as the ability to define pairwise bag constraints. The grounded semantic learner for NERD, similarly to Phase Recognition, wrapped a batch version of techniques proposed in Part II of this thesis. Our evaluation showed that the resulting Web architecture correctly executed all semantic experts and semantic meta components, and achieved good outcomes for the Grounded NERD SEP.

Our work on applying semantic experts to TPM, Surgical Phase Recognition and NERD as well as evaluating their time-efficiency answered Research Question 4, where the corresponding Hypothesis 4 can be confirmed.

Similarly, our work on the resulting Web architectures for medical assistance and NERD, as well as the developed semantic meta components and their evaluations in terms of abstract semantic planning, grounded semantic planning and grounded semantic learning answered Research Question 5, where the corresponding Hypothesis 5 can be confirmed.

Part IV

Conclusion

This part concludes the thesis by first summarizing the presented contents in terms of research questions, hypotheses and contributions (Chapter 10). We finally point out future research directions for each of the stipulated hypotheses in Chapter 11.

Chapter 10

Summary

In **this chapter**, we summarize the contents of this thesis by revisiting all stipulated hypotheses and corresponding research questions, and highlighting our respective contributions. We start with our hypotheses with respect to the learning problem.

Hypothesis 1. *By learning expert weights for individual task instances, which express an expert's ability to support generating the correct task solution, and by choosing experts with respect to the* exploration-exploitation *problem, we can maximize the number of correct solutions for multi-step tasks (Chapter 5).*

Research Question 1 dealt with finding an adequate formalization for solving multi-step tasks with budgeted expert advice. We introduced EPs, a decision theoretic framework extending MDPs with expert advice and budget constraints. The target property of expert weights in EPs can be analogously defined to the well-known Bellman equations for MDPs, stipulating that an expert weight for a decision candidate has to have a positive impact on future steps. By learning a policy which maximizes the expected cumulative reward over steps (and adequate engineered rewards), we model all relevant learning challenges. Hypothesis 1 can only be partially confirmed without adequate learning methods, which we target in Hypothesis 2.

Hypothesis 2. *RL using relational features for individual task instances enables to deal with expert budgets in order to maximize the number of correct solutions for multi-step tasks with expert advice (Chapter 5 & 7).*

In Research Question 2, we targeted learning methods for EPs to maximize the expected cumulative reward. Our contributions comprise two model-free RL approaches, where we directly approximated expert weights for decision candidates in the online and the (mini-) batch setting, respectively. While our Online RL method is lightweight and easily scalable, our (mini-) Batch RL approach relies on a complex LGM. We exploit a novel feature representation (referred to as meta dependencies), which consists of relational expert measures to quantify the relative performance of an expert in diverse neighborhoods of the current decision candidate. We describe our evaluations for all learning approaches to fully confirm Hypothesis 2 after introducing Hypothesis 3.

Hypothesis 3. *Framing expert selection and -combination for multi-step tasks as Multiagent coordination problem and solving the latter can improve the number of correct solutions for multi-step tasks while keeping the resulting system flexible (Chapter 6 & 7).*

Research Question 3 deals with the central problem of expert correlation in multi-step tasks,

which potentially introduces biases to learning methods for EPs. We extended EPs with multiple agents, where each agent manages exactly one expert in MEPs. The latter enable to treat the correlation problem as expert coordination. To this end, we explored both the lightweight independent learner- as well as rich joint-action learner protocol and developed respective methods.
We evaluated our learning methods for EPs as well as MEPs for the NERD task and showed that, for both tweets and news articles, our methods were able to alternately dominate all baselines, which consisted of – among others – the monolithic systems as deployed on the Web. We were thus able to confirm Hypothesis 2 & 3, and in retrospect also confirm Hypothesis 1 concerning our general-purpose framework.
We now deal with our hypotheses, research questions and resulting contributions for the problem of Web automation of multi-step tasks.
Hypothesis 4. *Given a multi-step task with available experts, structuring experts and task instances with lightweight annotations modeled with the Resource Description Language (RDF) as well as exploiting HTTP methods to execute the services is sufficient to find and execute expert services (Chapter 8 & Chapter 9).*

Associated Research Question 4 focused on the problem of heterogeneity in expert functionalities, implementations and interfaces available on the Web. We proposed lifting experts to *semantic experts*, which are LAPIs able to provide access via HTTP methods, introspection via machine-readable descriptions and provenance via execution-specific groundings of the latter. We applied the concept of semantic experts to NERD- as well as medical assistance experts and showed that no significant time overhead is generated. The latter are thus eligible for automating multi-step tasks on the Web, which confirms Hypothesis 4.
Hypothesis 5. *Expert services lifted with concepts of the Semantic Web can be executed on a data-driven basis, where decision-theoretic models enable planning their execution as well as integrating expert weight learning approaches to solve multi-step tasks (Chapter 8 & Chapter 9).*

Research Question 5 deals with enabling to efficiently find suitable experts for a step and automatically execute the latter without human intervention. Our contribution did comprise a data-driven approach for semantic expert execution based on a declarative rule-based engine LD-Fu. We extended EPs with semantic concepts – yielding SEPs – to enable the reuse of MDP planning techniques for semantic expert discovery and the integration of expert weight learning methods as proposed throughout this thesis. SEPs with respective *semantic meta components* have been applied to both the NERD- and the two medical assistance tasks and evaluated in terms of running exemplary SEPs. Hypothesis 5 was thus confirmed.

Chapter 11

Future Work

This chapter points out future work with respect to the proposed frameworks and methods of this thesis.

We begin by discussing future work with respect to the framework of EPs (**Hypothesis 1**), which enable to reuse, extend and develop RL algorithms to learn how to choose and combine experts and their hypotheses. An important direction with regards to real-world applicability is making budget constraints more flexible. The underlying assumption is that one could save expert executions for decision candidates (or data points) and steps, where one is confident about the performance of the respective experts. The saved budget can either lead to overall cost savings or be distributed to decision candidates (or data points) and steps where one is unsure. This extension is far from trivial, as EPs have to be extended with finer-grained decisions which entails to re-model state and action spaces. Reward functions for such an EP extension have to be carefully engineered and empirically evaluated, as it is not clear how to achieve convergent behavior.

To this end, working towards using *Inverse Reinforcement Learning* (IRL) might be the most innovative and fruitful direction, where one tries to recover the reward function and thus elegantly avoids to manually engineer the latter. By showing the agent *optimal* expert executions and hypothesis combinations, the former could learn to extract the important subset of decisions to take in order to maximize the expected cumulative reward for the EP extension. IRL methods significantly differ from RL methods and thus open an extending line of research.

A possible line of future work for learning methods for EPs (**Hypothesis 2**) comprises to evaluate pure RL algorithms such as Value-based Q-Learning or Actor-Critic algorithms (e.g. [133]) for EPs. The methods we proposed dealt with unstructured data (such as text) where, on the one hand, pure Deep RL (such as Value-based *Deep Q-Learning* [93] or Actor-Critic TRPO [115]) might not converge for relatively small hand-labeled reference data sets and, on the other hand, elaborate feature representations (e.g. the proposed meta dependencies) are needed to enable generalization. As a consequence, using pure RL algorithms with adequate feature representations is promising.

To this end, it is crucial to study convergence behaviors of methods proposed in this thesis, as convergence is the central means to quantify the performance of RL algorithms. While our empirical evaluation of the latter is important and a first step, theoretical frameworks such as *Probably Approximately Correct* (PAC) [77, 132] and *Knows What It Knows* (KWIK) [87]

fundamentally approach the study of convergence by proving respective guarantees.

Future works concerning the reduction of expert correlation for multi-step tasks by Multiagent coordination (**Hypothesis 3**) might lead towards the study of distributed systems, where responsible agents cannot be centralized because respective experts are owned by independent parties, or the system needs to scale to very large amounts of experts. The former reason is of interest on its own, as real-world scenarios often comprise independent stakeholders with mostly cooperative intentions but individual power. A prominent example is medical assistance, where experts might represent models trained from data of different institutions or physicians themselves. Here, such coordination protocols and rewards paired with adequate incentives might enable to apply MEPs in medical settings.

To achieve complete decentralization, one has to optimize communication among agents. The resulting constrained communication problem entails to deal with communication actions, as we already defined for MEPs. Recent advances in costly feature selection [47] based on bandit algorithms can be reused and extended, where one would model all agent combinations of the given communication budget length as possible communication actions. An alternative direction is using deep neural networks, which have been successfully applied to learning how to communicate [46].

Taking the opposite perspective for future works, a promising line of research deals with using Multiagent coordination for centralized decision-making [82, 134] which has lead to good results for game playing (which currently is the main proof-of-concept for novel RL algorithms). By adding a superior meta-agent, collecting the decisions of the coordination agents and taking global decisions, one could improve the overall performance for MEPs to solve multi-step tasks.

We next discuss future works for lifting experts to semantic experts (**Hypothesis 4**), leading to knowledge integration for multi-step tasks. A core assumption of the lifting process was the manual annotation and -wrapping process of expert services (or respective local implementations) to semantic experts. Here, building on advances in the automatic deployment of Docker containers for semantic experts is an essential next step, as this leads to better scalability as well as availability. Besides easing and automating the wrapping process of experts, the development of suggestion mechanisms with regards to manually creating semantic descriptions (and most importantly pre- and postconditions) based on supervised learning is important. Such methods can only be developed by extending prominent service provider platforms, such as Algorithmia, with semantic annotations for experts and providing real-world incentives for end users to build the manually annotated ground truth.

To this end, we also require semantic annotations to match perfectly, i.e. the same vocabularies have to be used for all semantic experts to enable exchangeability as well as pre- and postconditions matchings. There are future directions one might take to add flexibility. A first step towards learning semantic matchings would be to assume simple typed annotations with given upper classes, e.g. `spc:Image`, to restrict the search space. One might then approach to learn supervised models based on the eventual positive or negative outcomes of a task when using a specific expert with uncertainty in its annotations.

An additional future work direction we want to mention deals with enabling focused, goal-oriented developments and liftings of semantic experts by means of IRL. By studying end user behavior on service provide platforms, one can model a MDP and learn the intrinsic reward function of the class of end users on a specific platform. This enables better understand where Web automation is needed and where the best starting points for manually developing semantic experts are.

Finally, while we already generate provenance metadata based on semantic expert as well as semantic meta component executions, reusing established sophisticated ontologies (such as OPMW) is beneficial and might open further opportunities and research challenges. One example might be discovering and selecting semantic experts when no training data is available, which requires to examine generated metadata from other institutions (published as Linked Data). To this end, guaranteeing reproducibility of results and determining if sufficient contextual information is available to trust these outcomes are essential.

Future works for SEPs and proposed semantic meta components for diverse planning and learning problems (**Hypothesis 5**) might be concerned with extending SEPs to distributed systems, which are essential for Web- and Internet of Things (IoT) applications. The development of *self-governed components*, which autonomously aggregate their needed inputs and coordinate as well as communicate with others to serve end users is a novel and fruitful line of research. By working towards Multiagent SEPs, one has to fundamentally re-design semantic meta components as well as the data-driven execution by the semantic agent. The idea behind the latter (embodied by concepts underlying the tool LD-Fu) is still central, but needs to be enriched with communication and coordination capabilities. Numerous problems arise, as communication cannot be exhaustive and coordination must yield fair and sustainable consensuses.

Besides distributing SEPs, future works concerning further real-world constraints (as, for example, available in medical decision-making) are important. The latter might comprise security of accessing and storing semantic experts and their sensitive hypotheses or combination and choice of hypotheses when the involved decisions have detrimental impacts.

Besides, in case of abstract planning we only leveraged semantics to a small degree, namely in terms of pre- and postcondition matchings. To better exploit semantic liftings of experts, one could study potential advantages of relational MDPs, where abstract as well as grounded state- and action spaces of SEPs do not have to be lowered to tabular representations. As a consequence, one could exploit more efficient planning and learning algorithms for SEPs which are especially useful when the number of variables in semantic expert conditions or start state groundings increase in size. To this end, defining stochastic transition function with logical rules (as practiced in LOMDPs) also enables to model uncertainties with regards to imperfect matchings of semantic expert conditions or availability of the latter.

With regards to learning in SEPs we also only dealt with lifting batch learning approaches to semantic grounded learners. As an extensive part of our proposed expert weight learning techniques (see Chapter 5) supports online learning which yields higher degrees of flexibility, enabling such liftings is beneficial. However, this either requires to keep a state or store

learned expert weights to gradually update the latter.

Finally, as we mentioned critical reward impacts for sensitive domains such as medicine, further qualitative and quantitative evaluations are needed to work towards richer provenance models and thus higher end user acceptance of automatically pipelined workflows, e.g. end user questionnaires to quantify trust thresholds for suggested task solution

To summarize, while this thesis contributes to decision-making for multi-step tasks with expert advice, it also sets the ground for diverse, fruitful research directions.

Bibliography

[1] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, Marco Montali, and Paolo Torroni. Policy-based reasoning for smart web service interaction. In *SWAP 2006 - Semantic Web Applications and Perspectives, Proceedings of the 3rd Italian Semantic Web Workshop, Scuola Normale Superiore, Pisa, Italy*, 2006.

[2] Michael Albrecht, Patrick Donnelly, Peter Bui, and Douglas Thain. Makeflow: A portable abstraction for data intensive computing on clusters, clouds, and grids. In *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies, SWEET@SIGMOD, Scottsdale, AZ, USA*, SWEET '12, pages 1:1–1:13. ACM, 2012.

[3] Kareem Amin, Satyen Kale, Gerald Tesauro, and Deepak S. Turaga. Budgeted prediction with expert advice. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, Texas, USA*, pages 2490–2496, 2015.

[4] Hartwig Anzt, Werner Augustin, Martin Baumann, Thomas Gengenbach, Tobias Hahn, Andreas Helfrich-Schkarbanenko, Vincent Heuveline, Eva Ketelaer, Dimitar Lukarski, Andreas Nestler, Sebastian Ritterbusch, Staffan Ronnas, Michael Schick, Mareike Schmidtobreick, Chandramowli Subramanian, Jan-Philipp Weiss, Florian Wilhelm, and Martin Wlotzka. Hiflow^3: A hardware-aware parallel finite element package. In *Tools for High Performance Computing 2011 - Proceedings of the 5th International Workshop on Parallel Tools for High Performance Computing, ZIH, Dresden, September 2011*, pages 139–151, 2011. doi: 10.1007/978-3-642-31476-6_12.

[5] Danilo Ardagna and Barbara Pernici. Adaptive service composition in flexible processes. *IEEE Trans. Software Eng.*, 33(6):369–384, 2007. doi: 10.1109/TSE.2007.1011.

[6] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM J. Comput.*, 32(1):48–77, 2002. doi: 10.1137/S0097539701398375.

[7] Stephen H. Bach, Matthias Broecheler, Lise Getoor, and Dianne P. O'Leary. Scaling MPE inference for constrained continuous markov random fields with consensus optimization. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, USA*, pages 2663–2671, 2012.

[8] Stephen H. Bach, Bert Huang, Ben London, and Lise Getoor. Hinge-loss markov random fields: Convex inference for structured prediction. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2013, Bellevue, WA, USA*, 2013.

[9] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2):41–77, 2003. doi: 10.1023/A:1022140919877.

[10] Amparo Elizabeth Cano Basave, Giuseppe Rizzo, Andrea Varga, Matthew Rowe, Milan Stankovic, and Aba-Sah Dadzie. Making sense of microposts (#microposts2014) named entity extraction & linking challenge. In *Proceedings of the 4th Workshop on Making Sense of Microposts co-located with the 23rd International World Wide Web Conference (WWW'14), Seoul, Korea*, pages 54–60, 2014.

[11] Richard Bellman. *Dynamic programming*. Princeton University Press, 1957.

[12] V. Richard Benjamins and Christine Pierret-Golbreich. Assumptions of problem-solving methods. In *Advances in Knowledge Acquisition, 9th European Knowledge Acquisition Workshop, co-located with the International Conference on Knowledge Engineering and Knowledge Management, EKAW'96, Nottingham, UK*, pages 1–16, 1996.

[13] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975. doi: 10.1145/361002.361007.

[14] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284 (5):34–43, 2001.

[15] Alina Beygelzimer, John Langford, Lihong Li, Lev Reyzin, and Robert E. Schapire. Contextual bandit algorithms with supervised learning guarantees. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA*, pages 19–26, 2011.

[16] Alina Beygelzimer, Anton Riabov, Daby Sow, Deepak S. Turaga, and Octavian Udrea. Big data exploration via automated orchestration of analytic workflows. In *10th International Conference on Autonomic Computing, ICAC'13, San Jose, CA, USA*, pages 153–158, San Jose, CA, 2013. ISBN 978-1-931971-02-7.

[17] Alina Beygelzimer, Satyen Kale, and Haipeng Luo. Optimal and adaptive algorithms for online boosting. In *IJCAI*, pages 4120–4124, 2016.

[18] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009. doi: 10.4018/jswis.2009081901.

[19] Daniel Blankenberg, Gregory Von Kuster, Nathaniel Coraor, Guruprasad Ananda, Ross Lazarus, Mary Mangan, Anton Nekrutenko, and James Taylor. Galaxy: a web-based genome analysis tool for experimentalists. *Current protocols in molecular biology*, 2010. doi: 10.1002/0471142727.mb1910s89.

[20] Craig Boutilier. Sequential optimality and coordination in multiagent systems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden*, pages 478–485, 1999.

[21] Ronen I. Brafman and Moshe Tennenholtz. R-MAX - A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2002. doi: 10.1162/153244303765208377.

[22] Pavel Brazdil, Christophe Giraud-Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to Data Mining*. Springer, 1 edition, 2008. ISBN 3540732624, 9783540732624.

[23] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. doi: 10.1023/A:1010933404324.

[24] Matthias Bröcheler, Lilyana Mihalkova, and Lise Getoor. Probabilistic similarity logic. In *UAI 2010, Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA*, pages 73–82, 2010.

[25] Martin Carpenter and Daniel Kudenko. Baselines for joint-action reinforcement learning of coordination in cooperative multi-agent systems. In *Adaptive Agents and Multi-Agent Systems II: Adaptation and Multi-Agent Learning*, pages 55–72, 2005. doi: 10.1007/978-3-540-32274-0_4.

[26] Fabio Casati, Malú Castellanos, Umeshwar Dayal, and Ming-Chien Shan. Probabilistic, context-sensitive, and goal-oriented service selection. In *Service-Oriented Computing - ICSOC 2004, Second International Conference, New York, NY, USA*, pages 316–321, 2004. doi: 10.1145/1035167.1035213.

[27] Nicolò Cesa-Bianchi, Yoav Freund, David Haussler, David P. Helmbold, Robert E. Schapire, and Manfred K. Warmuth. How to use expert advice. *J. ACM*, 44(3):427–485, 1997. doi: 10.1145/258128.258179.

[28] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language (WSDL) 1.1. W3C recommendation, W3C, 2001. URL `http://www.w3.org/TR/wsdl`. accessed April 14, 2015.

[29] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, Madison, Wisconsin, USA*, pages 746–752, 1998.

[30] Tom Croonenborghs, Karl Tuyls, Jan Ramon, and Maurice Bruynooghe. Multi-agent relational reinforcement learning. In *Learning and Adaption in Multi-Agent Systems, First International Workshop, LAMAS, Utrecht, The Netherlands, Revised Selected Papers*, pages 192–206, 2005.

[31] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, and R. Kent Wenger. Pegasus, a workflow management system for science automation. *Future Generation Comp. Syst.*, 46:17–35, 2015. doi: 10.1016/j.future.2014.10.008.

[32] Marc Peter Deisenroth and Carl Edward Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA*, pages 465–472. Omnipress, 2011.

[33] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell. Res.*, 13:227–303, 2000. doi: 10.1613/jair.639.

[34] Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Kevin Murphy, Shaohua Sun, and Wei Zhang. From data fusion to knowledge fusion. *PVLDB*, 7(10):881–892, 2014. doi: 10.14778/2732951.2732962.

[35] Xin Luna Dong, Evgeniy Gabrilovich, Kevin Murphy, Van Dang, Wilko Horn, Camillo Lugaresi, Shaohua Sun, and Wei Zhang. Knowledge-based trust: Estimating the trustworthiness of web sources. *PVLDB*, 8(9):938–949, 2015. doi: 10.14778/2777598.2777603.

[36] Prashant Doshi, Richard Goodwin, Rama Akkiraju, and Kunal Verma. Dynamic workflow composition: Using markov decision processes. *Int. J. Web Service Res.*, 2(1):1–17, 2005. doi: 10.4018/jwsr.2005010101.

[37] Kurt Driessens and Jan Ramon. Relational instance based regression for relational reinforcement learning. In *Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 123–130, 2003.

[38] Saso Dzeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine Learning*, 43(1/2):7–52, 2001. doi: 10.1023/A:1007694015589.

[39] Thomas Fahringer, Radu Prodan, Rubing Duan, Jüurgen Hofer, Farrukh Nadeem, Francesco Nerieri, Stefan Podlipnig, Jun Qin, Mumtaz Siddiqui, Hong-Linh Truong, Alex Villazon, and Marek Wieczorek. *ASKALON: A Development and Grid Computing Environment for Scientific Workflows*, pages 450–471. Springer London, 2007.

[40] Dieter Fensel, Enrico Motta, Frank van Harmelen, V. Richard Benjamins, Monica Crubézy, Stefan Decker, Mauro Gaspari, Rix Groenboom, William E. Grosso, Mark A. Musen, Enric Plaza, Guus Schreiber, Rudi Studer, and Bob J. Wielinga. The unified problem-solving method development language UPML. *Knowl. Inf. Syst.*, 5(1):83–131, 2003. doi: 10.1007/s10115-002-0074-5.

[41] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, Montreal, Quebec, Canada*, pages 2962–2970, 2015.

[42] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. University of California, Irvine, 2000.

[43] Richard Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2(3/4):189–208, 1971. doi: 10.1016/0004-3702(71)90010-5.

[44] Rosa Filguiera, Iraklis Klampanos, Amrey Krause, Mario David, Alexander Moreno, and Malcolm Atkinson. dispel4py: A python framework for data-intensive scientific computing. In *Proceedings of the 2014 International Workshop on Data Intensive Scalable Computing Systems*, DISCS '14, pages 9–16. IEEE Press, 2014.

[45] Jenny Rose Finkel, Trond Grenager, and Christopher D. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *ACL 2005, 43rd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, University of Michigan, USA*, 2005.

[46] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems '16, Barcelona, Spain*, pages 2137–2145, 2016.

[47] Dean P. Foster, Satyen Kale, and Howard J. Karloff. Online sparse linear regression. In *Proceedings of the 29th Conference on Learning Theory, COLT 2016, New York, USA*, pages 960–970, 2016.

[48] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139, 1997. ISSN 0022-0000.

[49] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[50] Daniel Garijo and Yolanda Gil. A new approach for publishing workflows: abstractions, standards, and linked data. In *WORKS'11, Proceedings of the 6th Workshop on Workflows in Support of Large-Scale Science, co-located with , SC11, Seattle, WA, USA*, pages 47–56, 2011.

[51] Philipp Gemmeke, Maria Maleshkova, Patrick Philipp, Michael Götz, Christian Weber, Benedikt Kämpgen, Marco Nolden, Klaus Maier-Hein, and Achim Rettinger. Using linked data and web apis for automating the pre-processing of medical images. In *Proceedings of the 5th International Workshop on Consuming Linked Data (COLD 2014) co-located with the 13th International Semantic Web Conference (ISWC 2014), Riva del Garda, Italy*, pages 25–36, 2014.

[52] Yolanda Gil, Pedro A. Gonzalez-Calero, Jihie Kim, Joshua Moody, and Varun Ratnakar. A semantic framework for automatic generation of computational workflows using distributed data and component catalogues. *Journal of Experimental & Theoretical Artificial Intelligence*, 23(4): 389–467, 2011. doi: 10.1080/0952813X.2010.490962.

[53] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored MDPs. *J. Artif. Intell. Res.*, 19:399–468, 2003. doi: 10.1613/jair.1000.

[54] J. Octavio Gutiérrez-García and Kwang Mong Sim. Agent-based cloud service composition. *Appl. Intell.*, 38(3):436–464, 2013. doi: 10.1007/s10489-012-0380-x.

[55] James A. Hendler, Austin Tate, and Mark Drummond. AI planning: Systems and techniques. *AI Magazine*, 11(2):61–77, 1990.

[56] Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen Fürstenau, Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. Robust disambiguation of named entities in text. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP, Edinburgh, UK, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 782–792, 2011.

[57] Jiwei Huang, Ying Chen, Chuang Lin, and Junliang Chen. Ranking web services with limited and noisy information. In *2014 IEEE International Conference on Web Services, ICWS, 2014, Anchorage, AK, USA*, pages 638–645, 2014. doi: 10.1109/ICWS.2014.94.

[58] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991. doi: 10.1162/neco.1991.3.1.79.

[59] Anubhav Jain, Shyue Ping Ong, Wei Chen, Bharat Medasani, Xiaohui Qu, Michael Kocher, Miriam Brafman, Guido Petretto, Gian-Marco Rignanese, Geoffroy Hautier, Daniel K. Gunter, and Kristin A. Persson. Fireworks: a dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience*, 27(17):5037–5059, 2015. doi: 10.1002/cpe.3505.

[60] Joseph P. Joyce and George W. Lapinsky. A history and overview of the safety parameter display system concept. *Nuclear Science, IEEE Transactions on*, 30(1):744–749, Feb 1983. ISSN 0018-9499.

[61] Spiros Kapetanakis and Daniel Kudenko. Reinforcement learning of coordination in cooperative multi-agent systems. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, Edmonton, Alberta, Canada*, pages 326–331, 2002.

[62] Spiros Kapetanakis, Daniel Kudenko, and Malcolm J. A. Strens. Learning to coordinate using commitment sequences in cooperative multiagent systems. In *Adaptive Agents and Multi-Agent Systems II: Adaptation and Multi-Agent Learning*, pages 106–118, 2005. doi: 10.1007/978-3-540-32274-0_7.

[63] D. Katic, A. L. Wekerle, F. Gärtner, H. G. Kenngott, B. P. Müller-Stich, R. Dillmann, and S. Speidel. Knowledge-driven formalization of laparoscopic surgeries for rule-based intraoperative context-aware assistance. In *Information Processing in Computer-Assisted Interventions - 5th International Conference, IPCAI'14, Fukuoka, Japan*, pages 158–167, 2014.

[64] Michael J. Kearns and Satinder P. Singh. Near-optimal reinforcement learning in polynomial time. *Journal of Machine Learning Research*, 49(2-3):209–232, 2002. doi: 10.1023/A:1017984413808.

[65] Kristian Kersting and Luc De Raedt. Logical markov decision programs. In *Working Notes of the International Joint Conference on Artificial Intelligence, IJCAI, Workshop on Learning Statistical Models from Relational Data (SRL'03)*, pages 63–70, 2003.

[66] Kristian Kersting, Martijn van Otterlo, and Luc De Raedt. Bellman goes relational. In *Machine Learning, Proceedings of the Twenty-first International Conference (ICML'04), Banff, Alberta, Canada*, 2004. doi: 10.1145/1015330.1015401.

[67] Angelika Kimmig, Stephen Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. A short introduction to probabilistic soft logic. In *Workshop on Probabilistic Programming: Foundations and Applications, co-located with Conference on Neural Information Processing Systems*, pages 1–4, 2012.

[68] Angelika Kimmig, Lilyana Mihalkova, and Lise Getoor. Lifted graphical models: a survey. *Machine Learning*, 99(1):1–45, 2015. ISSN 1573-0565.

[69] Matthias Klusch, Andreas Gerber, and Marcus Schmidt. Semantic web service composition planning with owls-xplan. In *International Fall Symposium of the Association for the Advancement of Artificial Intelligence (AAAI) on Agents and the Semantic Web*, pages 55–62, 2005.

[70] Matthine Klusch. Semantic web service coordination. *CASCOM: Intelligent service coordination in the semantic web*, pages 59–104, 2008. doi: 10.1007/978-3-7643-8575-0_3.

[71] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press, 2009. ISBN 978-0-262-01319-2.

[72] 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. doi: 10.1109/MIC.2007.134.

[73] Jacek Kopecky, Karthik Gomadam, and Tomas Vitvar. hrests: An HTML microformat for describing restful web services. In *IEEE / WIC / ACM International Conference on Web Intelligence, WI'08, Sydney, NSW, Australia*, volume 1, pages 619–625. IEEE, 2008.

[74] Ramachandra Kota, Nicholas Gibbins, and Nicholas R. Jennings. Self-organising agent organisations. In *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'09), Budapest, Hungary, Volume 2*, pages 797–804, 2009. doi: 10.1145/1558109.1558122.

[75] Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35(3):48–60, 2014.

[76] Michael Kranzfelder, Christoph Staub, Adam Fiolka, Armin Schneider, Sonja Gillen, Dirk Wilhelm, Helmut Friess, Alois Knoll, and Hubertus Feussner. Toward increased autonomy in the surgical OR: needs, requests, and expectations. *Surg. Endoscopy*, 27(5):1681–1688, 2013. ISSN 0930-2794.

[77] Akshay Krishnamurthy, Alekh Agarwal, and John Langford. PAC reinforcement learning with rich observations. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems '16, Barcelona, Spain*, pages 1840–1848, 2016.

[78] Markus Krötzsch, Denny Vrandecic, and Max Völkel. Semantic mediawiki. In *The Semantic Web - ISWC'06, 5th International Semantic Web Conference, ISWC'06, Athens, GA, USA*, pages 935–942, 2006. doi: 10.1007/11926078_68.

[79] Florent Lalys, David Bouget, Laurent Riffaud, and Pierre Jannin. Automatic knowledge-based recognition of low-level tasks in ophthalmological procedures. *International Journal of Computer Assisted Radiology and Surgery*, 8(1):39–49, 2013. ISSN 1861-6410.

[80] Sascha Lange, Thomas Gabel, and Martin Riedmiller. *Batch Reinforcement Learning*, pages 45–73. Springer, 2012. ISBN 978-3-642-27645-3. doi: 10.1007/978-3-642-27645-3_2.

[81] John Langford, Lihong Li, and Tong Zhang. Sparse online learning via truncated gradient. *Journal of Machine Learning Research*, 10:777–801, 2009. doi: 10.1145/1577069.1577097.

[82] Romain Laroche, Mehdi Fatemi, Joshua Romoff, and Harm van Seijen. Multi-advisor reinforcement learning. *CoRR*, abs/1704.00756, 2017. URL `http://arxiv.org/abs/1704.00756`.

[83] Martin Lauer and Martin A. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *An algorithm for distributed reinforcement learning in cooperative multi-agent systems*, pages 535–542, 2000.

[84] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015. doi: 10.1038/nature14539.

[85] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. 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.

[86] Lihong Li, Michael L. Littman, and Christopher R. Mansley. Online exploration in least-squares policy iteration. In *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary, Volume 2*, pages 733–739, 2009. doi: 10.1145/1558109.1558113.

[87] Lihong Li, Michael L. Littman, Thomas J. Walsh, and Alexander L. Strehl. Knows what it knows: a framework for self-aware learning. *Machine Learning*, 82(3):399–443, 2011. doi: 10.1007/s10994-010-5225-4.

[88] Nebil Ben Mabrouk, Nikolaos Georgantas, and Valérie Issarny. Set-based bi-level optimisation for qos-aware service composition in ubiquitous environments. In *IEEE International Conference on Web Services, ICWS'15, New York, NY, USA*, pages 25–32, 2015. doi: 10.1109/ICWS.2015.14.

[89] David L. Martin, Mark H. Burstein, Drew V. McDermott, Sheila A. McIlraith, Massimo Paolucci, Katia P. Sycara, Deborah L. McGuinness, Evren Sirin, and Naveen Srinivasan. Bringing semantics to web services with OWL-S. *World Wide Web*, 10(3):243–277, 2007. doi: 10.1007/s11280-007-0033-x.

[90] Francisco S. Melo and Manuela M. Veloso. Learning of coordination: exploiting sparse interactions in multiagent systems. In *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'09), Budapest, Hungary, Volume 2*, pages 773–780, 2009. doi: 10.1145/1558109.1558118.

[91] Pablo N. Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. DBpedia spotlight: shedding light on the web of documents. In *Proceedings the 7th International Conference on Semantic Systems, I-SEMANTICS'11, Graz, Austria*, pages 1–8, 2011.

[92] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems '13, Lake Tahoe, Nevada, USA*, pages 3111–3119, 2013.

[93] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. doi: 10.1038/nature14236.

[94] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML'16, New York City, NY, USA*, pages 1928–1937, 2016.

[95] Thomas Neumuth, Gero Strauß, Jürgen Meixensberger, HeinzU Lemke, and Oliver Burgert. Acquisition of process descriptions from surgical interventions. In *Database and expert systems applications*, volume 4080 of *Lec. Not. in Comp. Sc.*, pages 602–611. Springer, 2006.

[96] Thomas Neumuth, Pierre Jannin, Gero Strauss, Juergen Meixensberger, and Oliver Burgert. Validation of knowledge acquisition for surgical process models. *Journal of the American Medical Informatics Association*, 16(1):72 – 80, 2009. ISSN 1067-5027.

[97] Tom Oinn, Mark Greenwood, Matthew Addis, M. Nedim Alpdemir, Justin Ferris, Kevin Glover, Carole Goble, Antoon Goderis, Duncan Hull, Darren Marvin, Peter Li, Phillip Lord, Matthew R. Pocock, Martin Senger, Robert Stevens, Anil Wipat, and Chris Wroe. Taverna: Lessons in creating a workflow environment for the life sciences: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1067–1100, August 2006. ISSN 1532-0626.

[98] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP'14, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543, 2014.

[99] Jan Peters and Stefan Schaal. Policy gradient methods for robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2006, Beijing, China*, pages 2219–2225, 2006. doi: 10.1109/IROS.2006.282564.

[100] Patrick Philipp and Achim Rettinger. Reinforcement learning for multi-step expert advice. In *Proceedings of the 16th Conference on Autonomous Agents and Multiagent Systems, AAMAS'17, São Paulo, Brazil*, pages 962–971, 2017.

[101] Patrick Philipp, Maria Maleshkova, Achim Rettinger, and Darko Katic. A semantic framework for sequential decision making. In *Engineering the Web in the Big Data Era - 15th International Conference, ICWE'15, Rotterdam, The Netherlands*, pages 392–409, 2015. doi: 10.1007/978-3-319-19890-3_25.

[102] Patrick Philipp, Maria Maleshkova, Darko Katic, Christian Weber, Michael Götz, Achim Rettinger, Stefanie Speidel, Benedikt Kämpgen, Marco Nolden, Anna-Laura Wekerle, Rüdiger Dillmann, Hannes Kenngott, Beat P. Müller-Stich, and Rudi Studer. Toward cognitive pipelines of medical assistance algorithms. *Int. J. Computer Assisted Radiology and Surgery*, 11(9):1743–1753, 2016. doi: 10.1007/s11548-015-1322-y.

[103] Patrick Philipp, Maria Maleshkova, Achim Rettinger, and Darko Katic. A semantic framework for sequential decision making. *J. Web Eng.*, 16(5&6):471–504, 2017.

[104] Patrick Philipp, Achim Rettinger, and Maria Maleshkova. On automating decentralized multi-step service combination. In *IEEE International Conference on Web Services, ICWS'17, Honolulu, HI, USA*, pages 736–743, 2017. doi: 10.1109/ICWS.2017.89.

[105] A Platanios, Avrim Blum, and Tom M Mitchell. Estimating accuracy from unlabeled data. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence, UAI 2014, Quebec City, Quebec, Canada*, pages 682–691, 2014.

[106] Emmanouil Antonios Platanios, Avinava Dubey, and Tom M. Mitchell. Estimating accuracy from unlabeled data: A bayesian approach. In *Proceedings of the 33nd International Conference on Machine Learning, ICML'16, New York City, NY, USA,*, pages 1416–1425, 2016.

[107] Marc J. V. Ponsen, Tom Croonenborghs, Karl Tuyls, Jan Ramon, and Kurt Driessens. Learning with whom to communicate using relational reinforcement learning. In *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary, Volume 2*, pages 1221–1222, 2009. doi: 10.1145/1558109.1558222.

[108] Frank Puppe. *Systematic introduction to expert systems - knowledge representations and problem-solving methods*. Springer, 1993. ISBN 978-3-540-56255-9.

[109] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, New York, USA, 1st edition, 1994. ISBN 0-471-61977-9.

[110] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In *Semantic Web Services and Web Process Composition, First International Workshop, SWSWPC'04, San Diego, CA, USA, Revised Selected Papers*, pages 43–54, 2004. doi: 10.1007/978-3-540-30581-1_5.

[111] John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976. doi: 10.1016/S0065-2458(08)60520-3.

[112] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Christoph Bussler, and Dieter Fensel. Web service modeling ontology. *Applied Ontology*, 1(1):77–106, 2005.

[113] Pablo Ruiz and Thierry Poibeau. Combining open source annotators for entity linking through weighted voting. In *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics, *SEM'15, Denver, Colorado, USA*, 2015.

[114] Nicolai Schoch, Patrick Philipp, Tobias Weller, Sandy Engelhardt, Mykola Volovyk, Andreas Fetzer, Marco Nolden, Raffaele De Simone, Ivo Wolf, Maria Maleshkova, Achim Rettinger, Rudi Studer, and Vincent Heuveline. Cognitive tools pipeline for assistance of mitral valve surgery. In *Proceedings of SPIE, Medical Imaging'16: Image-Guided Procedures, Robotic Interventions and Modeling*, pages 9786 – 9794, 2016. doi: 10.1117/12.2216059.

[115] John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning, ICML'15, Lille, France*, pages 1889–1897, 2015.

[116] Kwang Mong Sim. Agent-based cloud computing. *IEEE Trans. Services Computing*, 5(4): 564–577, 2012. doi: 10.1109/TSC.2011.52.

[117] Evren Sirin, James A. Hendler, and Bijan Parsia. Semi-automatic composition of web services using semantic descriptions. In *Web Services: Modeling, Architecture and Infrastructure, Proceedings of the 1st Workshop on Web Services: Modeling, Architecture and Infrastructure (WSMAI'03), In conjunction with ICEIS'03, Angers, France*, pages 17–24, 2003.

[118] Evren Sirin, Bijan Parsia, Dan Wu, James A. Hendler, and Dana S. Nau. HTN planning for web service composition using SHOP2. *J. Web Sem.*, 1(4):377–396, 2004. doi: 10.1016/j.websem.2004.06.005.

[119] René Speck and Axel-Cyrille Ngonga Ngomo. Ensemble learning for named entity recognition. In *The Semantic Web - ISWC'14 - 13th International Semantic Web Conference, Riva del Garda, Italy, Proceedings, Part I*, pages 519–534, 2014. doi: 10.1007/978-3-319-11964-9_33.

[120] Stefanie Speidel, Julia Benzko, Sebastian Krappe, Gunther Sudra, Pedram Azad, Beat Peter Müller-Stich, Carsten Gutt, and Rüdiger Dillmann. Automatic classification of minimally invasive instruments based on endoscopic image sequences. In *Proceedings of SPIE, Medical Imaging'09: Visualization, Image-Guided Procedures, and Modeling*, volume 7261, 2009.

[121] Sebastian Speiser and Andreas Harth. Integrating linked data and services with linked data services. In *The Semantic Web: Research and Applications - 8th Extended Semantic Web Conference, ESWC'11, Heraklion, Crete, Proceedings, Part I*, pages 170–184. Springer, 2011. doi: 10.1007/978-3-642-21034-1_12.

[122] Steffen Stadtmüller and Barry Norton. Scalable discovery of linked apis. *International Journal of Metadata, Semantics and Ontologies (IJMSO)*, 8(2):95–105, 2013. doi: 10.1504/IJMSO.2013.056603.

[123] Steffen Stadtmüller, Sebastian Speiser, Andreas Harth, and Rudi Studer. Data-fu: A language and an interpreter for interaction with read/write linked data. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil*, pages 1225–1236, 2013.

[124] Alexander L. Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L. Littman. PAC model-free reinforcement learning. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML'06), Pittsburgh, Pennsylvania, USA*, pages 881–888, 2006. doi: 10.1145/1143844.1143955.

[125] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *WWW*, pages 697–706, 2007.

[126] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998. ISBN 0262193981.

[127] Stefan Suwelack, Markus Stoll, Sebastian Schalck, Nicolai Schoch, Rüdiger Dillmann, Rolf Bendl, Vincent Heuveline, and Stefanie Speidel. The medical simulation markup language - simplifying the biomechanical modeling workflow. In *Medicine Meets Virtual Reality 21 - NextMed, MMVR 2014, Manhattan Beach, California*, pages 394–400, 2014. doi: 10.3233/978-1-61499-375-9-394.

[128] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'13, Chicago, IL, USA*, pages 847–855, 2013. doi: 10.1145/2487575.2487629.

[129] J. Tong, E. Haihong, M. Song, J. Song, and Y. Li. Web service QoS prediction under sparse data via local link prediction. In *10th IEEE International Conference on High Performance Computing and Communications & IEEE International Conference on Embedded and Ubiquitous Computing, HPCC/EUC'13, Zhangjiajie, China*, pages 2285–2290, 2013. doi: 10.1109/HPCC.and.EUC.2013.328.

[130] Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Michael Röder, Daniel Gerber, Sandro Athaide Coelho, Sören Auer, and Andreas Both. AGDISTIS - graph-based disambiguation of named entities using linked data. In *The Semantic Web - ISWC'14 - 13th International Semantic Web Conference, Riva del Garda, Italy, Proceedings, Part I*, pages 457–471, 2014. doi: 10.1007/978-3-319-11964-9_29.

[131] Ricardo Usbeck, Michael Röder, Axel-Cyrille Ngonga Ngomo, Ciro Baron, Andreas Both, Martin Brümmer, Diego Ceccarelli, Marco Cornolti, Didier Cherix, Bernd Eickmann, Paolo Ferragina, Christiane Lemke, Andrea Moro, Roberto Navigli, Francesco Piccinno, Giuseppe Rizzo, Harald Sack, René Speck, Raphaël Troncy, Jörg Waitelonis, and Lars Wesemann. GERBIL: general entity annotator benchmarking framework. In *Proceedings of the 24th International Conference on World Wide Web, WWW'15, Florence, Italy*, pages 1133–1143, 2015. doi: 10.1145/2736277.2741626.

[132] Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984. doi: 10.1145/1968.1972.

[133] Hado Van Hasselt and Marco A Wiering. Reinforcement learning in continuous action spaces. In *(IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 272–279. IEEE, 2007.

[134] Harm van Seijen, Mehdi Fatemi, Joshua Romoff, and Romain Laroche. Improving scalability of reinforcement learning by separation of concerns. *CoRR*, abs/1612.05159, 2016. URL `http://arxiv.org/abs/1612.05159`.

[135] Ruben Verborgh, Andreas Harth, Maria Maleshkova, Steffen Stadtmüller, Thomas Steiner, Mohsen Taheriyan, and Rik Van de Walle. *Survey of Semantic Description of REST APIs*, pages 69–89. Springer New York, 2014. doi: 10.1007/978-1-4614-9299-3_5.

[136] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95, 2002. doi: 10.1023/A:1019956318069.

[137] Tomas Vitvar, Jacek Kopecký, Jana Viskova, and Dieter Fensel. Wsmo-lite annotations for web services. In *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC'08, Tenerife, Canary Islands, Spain*, pages 674–689, 2008. doi: 10.1007/978-3-540-68234-9_49.

[138] Denny Vrandecic and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, 2014. doi: 10.1145/2629489.

[139] Thomas J. Walsh, Istvan Szita, Carlos Diuk, and Michael L. Littman. Exploring compact reinforcement-learning representations with linear regression. In *UAI'09, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada*, pages 591–598, 2009.

[140] Christopher J. C. H. Watkins and Peter Dayan. Technical note q-learning. *Machine Learning*, 8:279–292, 1992. doi: 10.1007/BF00992698.

[141] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992. doi: 10.1007/BF00992696.

[142] Ivo Wolf, Marcus Vetter, Ingmar Wegner, Thomas Böttger, Marco Nolden, Max Schöbinger, Mark Hastenteufel, Tobias Kunert, and Hans-Peter Meinzer. The medical imaging interaction toolkit. *Medical Image Analysis*, 9(6):594 – 604, 2005. ISSN 1361-8415.

[143] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992. doi: 10.1016/S0893-6080(05)80023-1.

[144] Ian Wood, Benjamin P. Vandervalk, E. Luke McCarthy, and Mark D. Wilkinson. OWL-DL domain-models as abstract workflows. In *Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies - 5th International Symposium, ISoLA'12, Heraklion, Crete, Greece, Proceedings, Part II*, pages 56–66. Springer, 2012. doi: 10.1007/978-3-642-34032-1_6.

[145] S. S. Yau and Y. Yin. QoS-based service ranking and selection for service-based systems. In *IEEE International Conference on Services Computing, SCC'11, Washington, DC, USA*, pages 56–63, 2011. doi: 10.1109/SCC.2011.114.

[146] Chih-Han Yu, Justin Werfel, and Radhika Nagpal. Collective decision-making in multi-agent systems by implicit leadership. In *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10), Toronto, Canada, Volume 1-3*, pages 1189–1196, 2010. doi: 10.1145/1838186.1838192.

[147] Zibin Zheng, Xinmiao Wu, Yilei Zhang, Michael R. Lyu, and Jianmin Wang. Qos ranking prediction for cloud services. *IEEE Trans. Parallel Distrib. Syst.*, 24(6):1213–1222, 2013. doi: 10.1109/TPDS.2012.285.

[148] Y. Zhou, L. Liu, C. S. Perng, A. Sailer, I. Silva-Lepe, and Z. Su. Ranking services by service network structure and service attributes. In *IEEE 20th International Conference on Web Services, Santa Clara, CA, USA*, pages 26–33, 2013.

[149] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. Chapman & Hall/CRC, 1st edition, 2012. ISBN 1439830037, 9781439830031.

List of Figures

List of Tables

List of Abbreviations

AGDT AGDISTIS Tagger. 106, 114
AIDT AIDA Tagger. 106, 114
API Application Programming Interface. 48, 119
AutoML Automatic Machine Learning. 52

BGD Batch Gradient Descent. 31
BGP Basic Graph Pattern. 46, 47, 48, 55, 121, 122, 123, 124, 128, 130, 135, 138, 148, 156
BOW Bag-of-Words. 68

CDP Contextual Decision Process. 51, 61
CT Computed Tomography. 18, 19, 20

ECP Expert Coordination Process. 84, 85, 89, 90, 91, 92, 96, 97, 98, 99, 101, 114
EP Expert Process. 12, 13, 15, 16, 51, 52, 56, 59, 61, 62, 64, 65, 66, 65, 66, 67, 68, 73, 74, 75, 76, 77, 78, 81, 82, 83, 84, 85, 86, 87, 88, 89, 103, 104, 106, 107, 108, 111, 112, 114, 115, 116, 117, 120, 122, 123, 124, 125, 135, 137, 145, 168, 175, 176, 177, 195
EWH Exponential Weights / Hedge. 36, 37, 63, 64, 74, 75, 76, 77, 81, 82, 84, 93, 94, 101, 107, 109, 111, 113, 114, 115

FMA Foundational Model of Anatomy. 138
FOAF Friend-of-a-Friend. 45, 46

GD Gradient Descent. 31, 92, 93, 99, 100, 107, 109, 111, 112, 114, 115
GERBIL General Entity Annotation Benchmark Framework. 17, 105, 106

HLMRF Hinge-Loss Markov Random Field. 78, 81
HRL Hierarchical Reinforcement Learning. 53
HTN Hierarchical Task Network. 56
HTTP Hypertext Transfer Protocol. 4, 11, 47, 48, 119, 127, 130, 131, 138, 176

i.i.d. Independent and identically distributed. 28, 29, 31, 33, 35, 52, 63
IoT Internet of Things. 179
IRL Inverse Reinforcement Learning. 177, 178

JavaEE Java Enterprise Edition. 151
JAX-RS Java API for RESTful Web Services. 151
JSON-LD JavaScript Object Notation for Linked Data. 57

KB (Structured-) Knowledge Base. 4, 17, 47, 58, 63, 126, 127, 128, 129, 132, 135, 137, 138, 143, 160, 162, 164
KWIK Knows-What-It-Knows. 177

LAPI Linked Application Programming Interface. 13, 48, 55, 119, 120, 121, 127, 129, 130, 134, 135, 137, 141, 151, 162, 169, 176
LD-Fu Linked Data-Fu. 120, 130, 131, 135, 146, 152, 158, 161, 164, 165, 176, 179, 195

LGM Lifted Graphical Model. 32, 175
LOMDP Logical Markov Decision Process. 56, 179
LSH Locality Sensitive Hashing. 73

MAS Multiagent System. 11, 12, 14, 15, 42, 44, 51, 53, 54, 83, 84, 85
MDP Markov Decision Process. 6, 12, 13, 26, 38, 39, 40, 41, 42, 43, 44, 54, 55, 56, 57, 61, 64, 65, 66, 68, 74, 81, 84, 86, 89, 144, 145, 146, 148, 152, 155, 165, 169, 170, 175, 176, 178, 179, 195
MEP Multiagent Expert Process. 12, 15, 16, 53, 54, 59, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 96, 97, 99, 100, 101, 102, 103, 104, 106, 107, 108, 112, 115, 116, 168, 175, 176, 178
MeSH Medical Subject Headings. 138
MHA MetaImage. 19, 139
MITK Medical Imaging Interaction Toolkit. 19, 119, 139, 151
ML Machine Learning. 153, 156, 157, 158, 159
MLN Markov Logic Network. 80
MMDP Multiagent Markov Decision Process. 42, 43, 44, 54, 84, 86, 90
MPE Most Probable Explanation. 80
MRG Multi-Relational Graph. 32
MRI Magnetic Resonance Imaging. 18, 19, 20, 195
MSM Minimal Service Model. 55, 128
MSML Medical Simulation Markup Language. 119

N3 Notation3. 46, 130
NED Named Entity Disambiguation. 4, 7, 9, 13, 17, 18, 63, 84, 103, 105, 106, 108, 113, 115, 121, 160, 161, 162, 163, 164, 165, 170
NER Named Entity Recognition. 4, 5, 13, 17, 18, 62, 63, 68, 84, 103, 105, 106, 108, 113, 115, 121, 160, 161, 162, 163, 165, 170
NERD Named Entity Recognition- and Disambiguation. 13, 15, 16, 17, 18, 25, 59, 62, 63, 64, 84, 86, 103, 105, 108, 115, 116, 121, 123, 124, 134, 135, 137, 156, 160, 165, 168, 170, 171, 176, 195
NIF Natural Language Processing Interchange Format. 160
NLP Natural Language Processing. 3, 4, 13, 17, 25, 52, 56, 57, 62, 68, 115, 117, 120, 121, 122, 124, 137, 160, 162, 169, 170, 195
NN Nearest Neighbors. 73
NRRD Nearly Raw Raster Data. 19, 139, 152

OPM Open Provenance Model. 55
OPMW Open Provenance Model for Workflows. 55, 57, 179
OWL Web Ontology Language. 45, 55, 56, 57

PAC Probably Approximately Correct. 177
PACS Picture Archiving and Communication System. 19
PGM Probabilistic Graphical Model. 32, 42
PNG Portable Network Graphics. 128, 139
POS Part-of-Speech. 104
PSL Probabilistic Soft Logic. 63, 64, 77, 78, 80, 81, 82, 107, 113, 115
PSM Problem Solving Methods. 55, 56

QoS Quality-of-Service. 53

RadLex Radiology Lexicon. 138
RDF Resource Description Framework. 11, 44, 45, 46, 47, 48, 56, 57, 121, 122, 123, 124, 125, 126, 128, 135, 138, 145, 146, 151, 160, 176

RDFS Resource Description Framework Schema. 45
REST Representational State Transfer. 48, 143
RL Reinforcement Learning. 6, 10, 11, 12, 13, 15, 40, 41, 44, 51, 52, 53, 54, 57, 59, 61, 63, 64, 65, 68, 73, 74, 77, 81, 82, 83, 84, 93, 96, 97, 101, 107, 114, 115, 116, 124, 125, 175, 177, 178
RMDP Relational Markov Decision Process. 52, 56
RRL Relational Reinforcement Learning. 52, 54

SAS Single-Agent System. 51, 83, 93, 114
SAWSDL Semantic Annotations for Web Service Description Language. 55
SEP Semantic Expert Process. 13, 15, 16, 56, 57, 117, 119, 120, 121, 122, 123, 124, 125, 126, 127, 129, 131, 132, 133, 134, 135, 137, 144, 145, 148, 158, 159, 165, 168, 170, 176, 179
SGD Stochastic Gradient Descent. 31, 92
SLA Service Level Agreement. 56
SMW Semantic MediaWiki. 138
SoC Separation of Concerns. 53, 83, 84
SPARQL SPARQL Protocol and Resource Description Framework Query Language. 46, 47, 48, 57
SPS Spotlight Spotter. 106
SPT Spotlight Tagger. 106
SQL Structured Query Language for Relational Database Systems. 46
SRL Statistical Relational Learning. 32, 77, 109, 113, 115
ST Stanford Tagger. 106, 114
STRIPS Stanford Research Institute Problem Solver. 55, 56
SWRL Semantic Web Rule Language. 156, 159

TPM Tumor Progression Mapping. 17, 19, 20, 121, 124, 133, 137, 138, 139, 141, 143, 144, 145, 146, 148, 149, 151, 152, 155, 156, 159, 160, 162, 170, 195, 216
TRPO Trust Region Policy Optimization. 177
TURTLE Terse Resource Description Framework Triple Language. 45, 46

UPML Unified Problem Solving Method Language. 56
URI Uniform Resource Identifier. 44, 45, 47, 126, 131

WSDL Web Service Description Language. 55
WSMO Web Service Modeling Ontology. 55
WWW World Wide Web. 44

XML Extensible Markup Language. 56

Appendix A

Appendix: Semantics

Based on the contents present in the thesis, we now provide additional materials with regards to (i) semantic description which lift experts to semantic experts and (ii) linked programs for the semantic agent which is handling the execution of the Web-based infrastructure for SEPs.

A.1 Semantic Experts

We present the full semantic descriptions for all semantic experts of the TPM task. To better present the descriptions, we first gather all used prefixes.

```
@prefix dc:     <http://purl.org/dc/elements/1.1/>.
@prefix msm:    <http://cms-wg.sti2.org/minimal-service-model#>.
@prefix owl:    <http://www.w3.org/2002/07/owl#>.
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#>.
@prefix sawsdl: <http://www.w3.org/ns/sawsdl#>.
@prefix sp:     <http://surgipedia.sfb125.de/wiki/Special:URIResolver/>.
@prefix spc:    <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Category-3A>.
@prefix spp:    <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Property-3A>.
@prefix spimg:  <http://surgipedia.sfb125.de/images/>.
@prefix nif:    <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#>.
@prefix sep:    <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/sep#>.
@prefix sepnlp: <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/sepnlp#>.
@prefix sepkb:  <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/sepkb/id/>.
@prefix sparql: <http://www.w3.org/TR/rdf-sparql-query/#>.
@prefix exp:    <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/>.
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#>.
```

Listing A.1: Prefixes for semantic descriptions for semantic experts.

A.1.1 Brain Mask Generation

```
exp:BMG
    rdf:type                sep:SemanticExpert, spc:BrainMaskGeneration;
    rdfs:label              "Semantic Brain Mask Generation Expert";
    spp:contributor         sp:Patrick_Philipp,
                            sp:Philipp_Gemmeke;
    spp:creator             sp:Christian_Weber,
                            sp:Michael_Goetz;
```

```
    spp:exampleRequest       spimg:BMG_example_request.ttl;
    spp:exampleResponse      spimg:BMG_example_response.ttl;
    spp:sourceCode
        <https://github.com/patrickraoulphilipp/cognitivepipelines>;
    spp:description
        "Segmentation of brain in a head scan."@en;
    owl:sameAs               sp:BMG_Description;
    sawsdl:modelReference
        [ a           msm:Postcondition;
          rdf:value   """{

            ?headscan       rdf:type        spc:Headscan;
                            dc:format       ?format.
            FILTER (?format = spc:NRRD || ?format = spc:MHA)
            ?brainImage     rdf:type        spc:BrainImage;
                            dc:format       spc:NRRD;
                            spp:headscan    ?headscan.
            ?brainMask      rdf:type        spc:BrainMask;
                            dc:format       spc:NRRD;
                            spp:headscan    ?headscan.

                        }"""^^sparql:GraphPattern],
        [ a           msm:Precondition;
          rdf:value   """{

            ?headscan       rdf:type        spc:Headscan;
                            dc:format       ?format.
            FILTER (?format = spc:NRRD || ?format = spc:MHA)
            ?brainAtlasImg  rdf:type        spc:BrainAtlasImage;
                            dc:format       spc:NRRD.
            ?brainAtlasMask rdf:type        spc:BrainAtlasMask;
                            dc:format       spc:NRRD.

                        }"""^^sparql:GraphPattern].
```

Listing A.2: Semantic description of semantic brain mask generation expert.

A.1.2 Standard Normalization

```
exp:BN
    rdf:type                 sep:SemanticExpert, spc:BrainNormalization;
    rdfs:label               "Semantic Brain Mask Normalization Expert";
    spp:contributor          sp:Patrick_Philipp,
                             sp:Philipp_Gemmeke;
    spp:creator              sp:Christian_Weber,
                             sp:Michael_Goetz;
    spp:exampleRequest       spimg:BN_example_request.ttl;
    spp:exampleResponse      spimg:BN_example_response.ttl;
    spp:sourceCode
        <https://github.com/patrickraoulphilipp/cognitivepipelines>;
    spp:description
        "Normalization of brain mask in a head scan."@en;
    owl:sameAs               sp:BN_Description;
    sawsdl:modelReference
        [ a           msm:Postcondition;
```

```
          rdf:value   """{

            ?headscan       rdf:type        spc:Headscan.
            ?brainMask      rdf:type        spc:BrainMask;
                            dc:format       spc:NRRD;
                            spp:headscan    ?headscan.
            ?normMask       rdf:type        spc:NormalizedBrainMask;
                            dc:format       spc:NRRD;
                            spp:headscan    ?headscan;
                            spp:brainMask   ?brainMask.

                        }"""^^sparql:GraphPattern],
        [ a           msm:Precondition;
          rdf:value   """{

            ?headscan       rdf:type        spc:Headscan;
                            dc:format       ?format.
            FILTER (?format = spc:NRRD || ?format = spc:MHA)
            ?brainMask      rdf:type        spc:BrainMask;
                            dc:format       spc:NRRD;
                            spp:headscan    ?headscan.

                        }"""^^sparql:GraphPattern].
```

Listing A.3: Semantic description of semantic brain normalization expert.

A.1.3 Robust Normalization

```
exp:RBN
    rdf:type                sep:SemanticExpert, spc:RobustBrainNormalization;
    rdfs:label
        "Semantic Robust Brain Mask Normalization Expert";
    spp:contributor         sp:Patrick_Philipp,
                            sp:Philipp_Gemmeke;
    spp:creator             sp:Christian_Weber,
                            sp:Michael_Goetz;
    spp:exampleRequest      spimg:RBN_example_request.ttl;
    spp:exampleResponse     spimg:RBN_example_response.ttl;
    spp:sourceCode
        <https://github.com/patrickraoulphilipp/cognitivepipelines>;
    spp:description
        "Robust Normalization of brain mask in a head scan."@en;
    owl:sameAs              sp:RBN_Description;
    sawsdl:modelReference
        [ a           msm:Postcondition;
          rdf:value   """{

            ?headscan       rdf:type        spc:Headscan;
                            dc:format       ?format.
            FILTER (?format = spc:NRRD || ?format = spc:MHA)
            ?brainMask      rdf:type        spc:BrainMask;
                            dc:format       spc:NRRD;
                            spp:headscan    ?headscan.
            ?mSeg           rdf:type        spc:ManualSegmentation;
                            dc:format       spc:NRRD;
```

```
                                spp:headscan    ?headscan.
            ?normMask           rdf:type        spc:RobustNormalizedBrainMask;
                                dc:format       spc:NRRD;
                                spp:headscan        ?headscan;
                                spp:brainMask   ?brainMask;
                                spp:manualSegmentation
                                                ?mSeg.

                        }"""^^sparql:GraphPattern],
        [ a             msm:Precondition;
          rdf:value     """{

            ?headscan           rdf:type        spc:Headscan;
                                dc:format       ?format.
            FILTER (?format = spc:NRRD || ?format = spc:MHA)
            ?brainMask          rdf:type        spc:BrainMask;
                                dc:format       spc:NRRD;
                                spp:headscan    ?headscan.
            ?mSeg               rdf:type        spc:ManualSegmentation;
                                dc:format       spc:NRRD;
                                spp:headscan    ?headscan.

                        }"""^^sparql:GraphPattern].
```

Listing A.4: Semantic description of semantic robust brain normalization expert.

A.1.4 Registration

```
exp:BR
    rdf:type                sep:SemanticExpert, spc:BrainRegistration;
    rdfs:label              "Semantic Brain Mark Registration Expert";
    spp:contributor         sp:Patrick_Philipp,
                            sp:Philipp_Gemmeke;
    spp:creator             sp:Christian_Weber,
                            sp:Michael_Goetz;
    spp:exampleRequest      spimg:TS_example_request.ttl;
    spp:exampleResponse     spimg:TS_example_response.ttl;
    spp:description
        "Registration of normalized brain masks in head scans."@en;
    owl:sameAs              sp:BR_Description;
    sawsdl:modelReference
        [ a             msm:Postcondition;
          rdf:value     """{

            ?norm_bag           rdf:type        rdf:Bag.
            ?reg_bag            rdf:type        rdf:Bag.
            ?headscan           rdf:type        spc:Headscan;
                                dc:format       ?format.
            FILTER (?format = spc:NRRD || ?format = spc:MHA)
            ?brainMask          rdf:type        spc:BrainMask;
                                dc:format       spc:NRRD;
                                spp:headscan    ?headscan.
            ?normMask           rdf:type        ?norm;
                                rdfs:member     ?norm_bag;
                                dc:format       spc:NRRD;
```

```
                                spp:headscan    ?headscan;
                                spp:brainMask   ?brainMask.
            FILTER (?norm = spc:NormalizedBrainMask ||
                    ?norm = spc:RobustNormalizedBrainMask).
            ?regMask        rdf:type        spc:RegisteredBrainMask;
                                rdfs:member     ?reg_bag;
                                dc:format       spc:NRRD;
                                spp:normalizedMask
                                                ?normMask.

                            }"""^^sparql:GraphPattern],
        [ a           msm:Precondition;
          rdf:value   """{

            ?norm_bag       rdf:type        rdf:Bag;
            ?headscan       rdf:type        spc:Headscan;
                            dc:format       ?format.
            FILTER (?format = spc:NRRD || ?format = spc:MHA)
            ?brainMask      rdf:type        spc:BrainMask;
                            dc:format       spc:NRRD;
                            spp:headscan    ?headscan.
            ?normMask       rdf:type        ?norm;
                            rdfs:member     ?norm_bag;
                            dc:format       spc:NRRD;
                            spp:headscan    ?headscan;
                            spp:brainMask   ?brainMask.
            FILTER (?norm = spc:NormalizedBrainMask ||
                    ?norm = spc:RobustNormalizedBrainMask).

                      }"""^^sparql:GraphPattern].
```

Listing A.5: Semantic description of semantic brain registration expert.

A.1.5 Tumor Segmentation

```
exp:TS
    rdf:type                sep:SemanticExpert;
    rdfs:label              "Semantic Tumor Segmentation Expert";
    spp:contributor         sp:Patrick_Philipp,
                            sp:Philipp_Gemmeke;
    spp:creator             sp:Christian_Weber,
                            sp:Michael_Goetz;
    spp:exampleRequest      spimg:TS_example_request.ttl;
    spp:exampleResponse     spimg:TS_example_response.ttl;
    spp:sourceCode
        <https://github.com/patrickraoulphilipp/cognitivepipelines>;
    spp:description
        "Tumor segmentation of registered brain masks in head scans."@en;
    owl:sameAs              sp:TS_Description;
    sawsdl:modelReference
        [ a           msm:Postcondition;
          rdf:value   """{

            ?headscan       rdf:type        spc:Headscan;
                            dc:format       ?format.
```

```
            FILTER (?format = spc:NRRD || ?format = spc:MHA)
            ?normMask       rdf:type        ?norm;
                            dc:format       spc:NRRD;
                            spp:headscan    ?headscan;
            FILTER (?norm = spc:NormalizedBrainMask ||
                    ?norm = spc:RobustNormalizedBrainMask).
            ?regMask        rdf:type        spc:RegisteredBrainMask;
                            dc:format       spc:NRRD;
                            spp:normalizedMask
                                            ?brainMask.
            ?tumor          rdf:type        spc:SegmentedTumorMask;
                            dc:format       spc:NRRD.
                            spp:registeredMask
                                            ?regMask.

                        }"""^^sparql:GraphPattern],
        [ a             msm:Precondition;
          rdf:value     """{

            ?headscan       rdf:type        spc:Headscan;
                             dc:format      ?format.
            FILTER (?format = spc:NRRD || ?format = spc:MHA)
            ?normMask       rdf:type        ?norm;
                            dc:format       spc:NRRD;
                            spp:headscan    ?headscan;
            FILTER (?norm = spc:NormalizedBrainMask ||
                    ?norm = spc:RobustNormalizedBrainMask).
            ?regMask        rdf:type        spc:RegisteredBrainMask;
                            dc:format       spc:NRRD;
                            spp:normalizedMask
                                            ?brainMask.

                        }"""^^sparql:GraphPattern].
```

Listing A.6: Semantic description of semantic tumor segmentation expert.

A.1.6 Map Generation

```
exp:MG
    rdf:type                sep:SemanticExpert;
    rdfs:label              "Semantic Map Generatio Expert";
    spp:contributor         sp:Patrick_Philipp,
                            sp:Philipp_Gemmeke;
    spp:creator             sp:Christian_Weber,
                            sp:Michael_Goetz;
    spp:exampleRequest      spimg:TPM_example_request.ttl;
    spp:exampleResponse     spimg:TPM_example_response.ttl;
    spp:description
        "Generates a the final tumour progression map of registered brain
        masks in head scans."@en;
    owl:sameAs              sp:TPM_Description;
    sawsdl:modelReference
        [ a           msm:Postcondition;
          rdf:value   """{

```

```
            ?tpm            rdf:type        spc:TumorProgressionMap.
                            spp:mapContent  ?tpm_bag.
            ?reg_bag        rdf:type        rdf:Bag.
            ?tpm_bag        rdf:type        rdf:Bag;
                            rdf:_1          ?normMask.
            ?headscan       rdf:type        spc:Headscan;
                            dc:format       ?format.
            FILTER (?format = spc:NRRD || ?format = spc:MHA)
            ?normMask       rdf:type        ?norm;
                            dc:format       spc:NRRD;
                            spp:headscan    ?headscan.
            FILTER (?norm = spc:NormalizedBrainMask ||
                    ?norm = spc:RobustNormalizedBrainMask).
            ?regMask        rdf:type        spc:RegisteredBrainMask;
                            rdfs:member     ?reg_bag;
                            dc:format       spc:NRRD;
                            spp:normalizedMask
                                            ?brainMask.

                    }"""^^sparql:GraphPattern],
        [ a           msm:Precondition;
          rdf:value   """{

            ?norm_bag       rdf:type        rdf:Bag.
            ?headscan       rdf:type        spc:Headscan;
                            dc:format       ?format.
            FILTER (?format = spc:NRRD || ?format = spc:MHA)
            ?normMask       rdf:type        ?norm;
                            rdfs:member     ?norm_bag;
                            dc:format       spc:NRRD;
                            spp:headscan    ?headscan.
            FILTER (?norm = spc:NormalizedBrainMask ||
                    ?norm = spc:RobustNormalizedBrainMask).
            ?regMask        rdf:type        spc:RegisteredBrainMask;
                            rdfs:member     ?reg_bag;
                            dc:format       spc:NRRD;
                            spp:normalizedMask
                                            ?brainMask.

                    }"""^^sparql:GraphPattern].
```

Listing A.7: Semantic description of semantic map generation expert.

A.1.7 Surgical Phase Recognition

```
exp:MLPR
    rdf:type                sep:SemanticExpert;
    rdfs:label              "Semantic ML-based Phase Recognition Expert";
    spp:contributor         sp:Patrick_Philipp;
    spp:creator             sp:Darko_Katic;
    spp:exampleRequest      spimg:MLPR_example_request.ttl;
    spp:exampleResponse     spimg:MLPR_example_response.ttl;
    spp:sourceCode
        <https://github.com/patrickraoulphilipp/cognitivepipelines>;
    spp:description
```

```
        "Surgical phase recognition based on a ML-based algorithm.";
    owl:sameAs              sp:MLPR_Description;
    sawsdl:modelReference
        [ a           msm:Postcondition;
          rdf:value   """{

            ?phase            rdf:type        spc:Phase;
                              spp:value       ?value.
            ?event            rdf:type        spc:SurgicalEvent;
                              spp:instrument  ?instrument;
                              spp:action      ?action;
                              spp:structure   ?structure.
                              spp:phase       ?phase.

                      }"""^^sparql:GraphPattern],
        [ a           msm:Precondition;
          rdf:value   """{

            ?col              rdf:type        rdf:Bag;
                              rdf:li          ?trainingSample.
            ?trainingSample   rdf:type        spc:Surgery.
            ?ontology         rdf:type        spc:Ontology.
            ?event            rdf:type        spc:SurgicalEvent;
                              spp:instrument  ?instrument;
                              spp:action      ?action;
                              spp:structure   ?structure.
            ?instrument       rdf:type        spc:Instrument.
            ?action           rdf:type        spc:Action.
            ?structure        rdf:type        spc:TreatedStructure.

                      }"""^^sparql:GraphPattern].
```

Listing A.8: Semantic description of ML-based phase recognition expert.

```
exp:RPR
    rdf:type                sep:SemanticExpert;
    rdfs:label              "Semantic rule-based Phase Recognition Expert";
    spp:contributor         sp:Patrick_Philipp;
    spp:creator             sp:Darko_Katic;
    spp:exampleRequest      spimg:RPR_example_request.ttl;
    spp:exampleResponse     spimg:RPR_example_response.ttl;
    spp:sourceCode
        <https://github.com/patrickraoulphilipp/cognitivepipelines>;
    spp:description
        "Surgical phase recognition based on a rule-based algorithm.";
    owl:sameAs              sp:RPR_Description;
    sawsdl:modelReference
        [ a           msm:Postcondition;
          rdf:value   """{

            ?phase            rdf:type        spc:Phase;
                              spp:value       ?value.
            ?event            rdf:type        spc:SurgicalEvent;
                              spp:instrument  ?instrument;
                              spp:action      ?action;
```

```
                              spp:structure   ?structure.
                              spp:phase       ?phase.

                        }"""^^sparql:GraphPattern],
        [ a           msm:Precondition;
          rdf:value   """{

            ?ontology         rdf:type        spc:Ontology.
            ?event            rdf:type        spc:SurgicalEvent;
                              spp:instrument  ?instrument;
                              spp:action      ?action;
                              spp:structure   ?structure.
            ?instrument       rdf:type        spc:Instrument.
            ?action           rdf:type        spc:Action.
            ?structure        rdf:type        spc:TreatedStructure.

                        }"""^^sparql:GraphPattern].
```

Listing A.9: Semantic description of rule-based phase recognition expert.

A.1.8 Named Entity Recognition

Note that parameter configurations have been removed from pre- and postconditions for better presentation.

```
exp:GNER
    rdf:type                sep:SemanticExpert;
    rdfs:label              "Semantic Named Entity Recognition Expert";
    sepnlp:contributor      sepkb:Patrick_Philipp;
    sepnlp:creator          sepkb:Generic;
    sepnlp:exampleRequest   sepkb:GNER_example_request.ttl;
    sepnlp:exampleResponse  sepkb:GNER_example_response.ttl;
    sepnlp:sourceCode
        <https://github.com/patrickraoulphilipp/cognitivepipelines>;
    sepnlp:description
        "Named entity recognition algorithm for raw text.";
    owl:sameAs              sepkb:GNER_Description;
    sawsdl:modelReference
        [ a           msm:Postcondition;
          rdf:value   """{

            ?col              rdf:type             rdf:Bag;
                              rdf:li               ?annotation1, ?annotation2.
            ?annotation1      sepnlp:disjunct      ?annotation2.
            ?annotation1      rdf:type             sepnlp:Annotation;
                              nif:isString         ?text1;
                              sepnlp:token         ?token1;
                              sepnlp:isEntity      "true"^^xsd:boolean.
            ?token1           rdf:type             sepnlp:Token
                              nif:anchorOf         ?mention1;
                              nif:beginindex       ?start1;
                              nif:endIndex         ?end1;
                              nif:referenceContext ?text.
            ?annotation2      rdf:type             sepnlp:Annotation;
                              nif:isString         ?text2;
```

```
                                sepnlp:token          ?token2;
                                sepnlp:isEntity       "true"^^xsd:boolean.
            ?token2             rdf:type              sepnlp:Token
                                nif:anchorOf          ?mention2;
                                nif:beginindex        ?start2;
                                nif:endIndex          ?end2;
                                nif:referenceContext  ?text.
            ?text               rdf:type              ?type.
            FILTER (?type = nif:Sentence || ?textType = nif:Paragraph).

                      }"""^^sparql:GraphPattern],
        [ a           msm:Precondition;
          rdf:value   """{

            ?text               rdf:type              ?type.
            FILTER (?type = nif:Sentence || ?textType = nif:Paragraph).

                      }"""^^sparql:GraphPattern].
```

Listing A.10: Generic semantic description of NER expert.

A.1.9 Named Entity Disambiguation

Note that parameter configurations have been removed from pre- and postconditions for better presentation.

```
exp:GNED
    rdf:type                sep:SemanticExpert;
    rdfs:label              "Semantic Named Entity Disambiguation Expert";
    sepnlp:contributor      sepkb:Patrick_Philipp;
    sepnlp:creator          sepkb:Generic;
    sepnlp:exampleRequest   sepkb:GNED_example_request.ttl;
    sepnlp:exampleResponse  sepkb:GNED_example_response.ttl;
    sepnlp:sourceCode
        <https://github.com/patrickraoulphilipp/cognitivepipelines>;
    sepnlp:description
        "Named entity disambiguation algorithm for raw text.";
    owl:sameAs              sepkb:GNED_Description;
    sawsdl:modelReference
        [ a           msm:Postcondition;
          rdf:value   """{

            ?col              rdf:type            rdf:Bag;
                              rdf:li              ?annotation1.

            ?annotation1      rdf:type            sepnlp:Annotation;
                              nif:isString        ?text;
                              sepnlp:token        ?token1.
                              sepnlp:isEntity     "true"^^xsd:boolean;
                              sepnlp:resource     ?resource.

            ?token1           rdf:type            sepnlp:Token
                              nif:anchorOf        ?mention;
                              nif:beginindex      ?start;
```

```
                              nif:endIndex         ?end;
                              nif:referenceContext ?text.

                        }"""^^sparql:GraphPattern],
          [ a           msm:Precondition;
            rdf:value   """{

              ?col              rdf:type             rdf:Bag;
                                rdf:li               ?annotation1, ?annotation2.
              ?annotation1      sepnlp:disjunct      ?annotation2.
              ?annotation1      rdf:type             sepnlp:Annotation;
                                nif:isString         ?text1;
                                sepnlp:token         ?token1;
                                sepnlp:isEntity      "true"^^xsd:boolean.
              ?token1           rdf:type             sepnlp:Token
                                nif:anchorOf         ?mention1;
                                nif:beginindex       ?start1;
                                nif:endIndex         ?end1;
                                nif:referenceContext ?text.
              ?annotation2      rdf:type             sepnlp:Annotation;
                                nif:isString         ?text2;
                                sepnlp:token         ?token2;
                                sepnlp:isEntity      "true"^^xsd:boolean.
              ?token2           rdf:type             sepnlp:Token
                                nif:anchorOf         ?mention2;
                                nif:beginindex       ?start2;
                                nif:endIndex         ?end2;
                                nif:referenceContext ?text.
              ?text             rdf:type             ?type.
              FILTER (?type = nif:Sentence || ?textType = nif:Paragraph).

                        }"""^^sparql:GraphPattern].
```

Listing A.11: Generic semantic description of NED expert.

A.2 Semantic Agent

We first give a full example on how to execute semantic experts based on the step of brain mask generation in TPMs. We then present a condition for an instantiated task and respective exemplary outputs for abstract and grounded planners.

```
@prefix:          <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/sepkb/id/>.
@prefix rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs:     <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl:      <http://www.w3.org/2002/07/owl#>.
@prefix httpm:    <http://www.w3.org/2011/http-methods#>.
@prefix http:     <http://www.w3.org/2011/http#>.
@prefix sawsdl:   <http://www.w3.org/ns/sawsdl#>.
@prefix sparql:   <http://www.w3.org/TR/rdf-sparql-query/#>.
@prefix qrl:      <http://www.aifb.kit.edu/project/ld-retriever/qrl#>.
@prefix vocxnat:  <http://aifb-ls3-vm2.aifb.kit.edu:8080/xnatwrapper/xnat#>.
@prefix xnat:     <http://aifb-ls3-vm2.aifb.kit.edu:8080/xnatwrapper/id/>.
```

```
@prefix sp:       <http://surgipedia.sfb125.de/wiki/Special:URIResolver/>.
@prefix spc:      <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Category-3A>.
@prefix spp:      <http://surgipedia.sfb125.de/wiki/Special:URIResolver/Property-3A>.
@prefix sep:      <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/sep#>.
@prefix exp:      <http://aifb-ls3-vm2.aifb.kit.edu:8080/experts/id/>.
@prefix dc:       <http://purl.org/dc/elements/1.1/>.
```

Listing A.12: Prefixes for linked programs.

A.2.1 Semantic Expert Execution

```
{
:hbag           rdf:type        rdf:Bag.
:headscan1      rdf:type        spc:Headscan;
                dc:format       spc:NRRD.
:headscan2      rdf:type        spc:Headscan;
                dc:format       spc:NRRD.
:brainMask1     rdf:type        spc:BrainMask;
                dc:format       spc:NRRD;
                spp:headscan    xnat:headscan1.
:brainMask2     rdf:type        spc:BrainMask;
                dc:format       spc:NRRD;
                spp:headscan    xnat:headscan2.
:normMask1      rdf:type        spc:NormalizedBrainMask;
                rdfs:member     xnat:hbag;
                dc:format       spc:NRRD;
                spp:headscan    xnat:headscan1;
                spp:brainMask   xnat:brainMask1.
:normMask2      rdf:type        spc:NormalizedBrainMask;
                rdfs:member     xnat:hbag;
                dc:format       spc:NRRD;
                spp:headscan    xnat:headscan2;
                spp:brainMask   xnat:brainMask2.
} => {
_:a http:mthd httpm:POST;
    http:requestURI exp:BMG;
    http:body
    {
    :hbag           rdf:type        rdf:Bag.
    :headscan1      rdf:type        spc:Headscan;
                    dc:format       spc:NRRD.
    :headscan2      rdf:type        spc:Headscan;
                    dc:format       spc:NRRD.
    :brainMask1     rdf:type        spc:BrainMask;
                    dc:format       spc:NRRD;
                    spp:headscan    xnat:headscan1.
    :brainMask2     rdf:type        spc:BrainMask;
                    dc:format       spc:NRRD;
                    spp:headscan    xnat:headscan2.
    :normMask1      rdf:type        spc:NormalizedBrainMask;
                    rdfs:member     xnat:hbag;
                    dc:format       spc:NRRD;
                    spp:headscan        xnat:headscan1;
```

```
                    spp:brainMask    xnat:brainMask1.
    :normMask2      rdf:type         spc:NormalizedBrainMask;
                    rdfs:member      xnat:hbag;
                    dc:format        spc:NRRD;
                    spp:headscan     xnat:headscan2;
                    spp:brainMask    xnat:brainMask2.
    }.
}.
```

Listing A.13: Linked program for executing semantic brain registration expert for two processed patient headscans.

A.2.2 Semantic Planning

```
:ebag         rdf:type      rdf:Bag.
exp:BMG       rdf:type      sep:SemanticExpert, spc:BrainMaskGeneration;
              rdfs:member   _:ebag.
exp:BN        rdf:type      sep:SemanticExpert, spc:BrainNormalization;
              rdfs:member   _:ebag.
exp:RBN       rdf:type      sep:SemanticExpert, spc:RobustBrainNormalization;
              rdfs:member   _:ebag.
exp:BR        rdf:type      sep:SemanticExpert, spc:BrainRegistration;
              rdfs:member   _:ebag.
exp:TS        rdf:type      sep:SemanticExpert, spc:TumorSegmentation;
              rdfs:member   _:ebag.
exp:MG        rdf:type      sep:SemanticExpert, spc:MapGeneration;
              rdfs:member   _:ebag.
exp:RPR       rdf:type      sep:SemanticExpert, spc:RuleBasedPhaseRecognition;
              rdfs:member   _:ebag.
exp:MLPR      rdf:type      sep:SemanticExpert, spc:MLBasedPhaseRecognition;
              rdfs:member   _:ebag.

xnat:start1   rdf:type      sep:GroundedState, sep:StartState.
              sawsdl:modelReference
        [ a           sep:GoundedCondition;
          rdf:value   """{
            xnat:headscan1     rdf:type       spc:Headscan;
                               dc:format      spc:NRRD;
                               spp:manualSegmentation
                                              xnat:seg1.
                               spp:patient    xnat:patient1.
            xnat:headscan2     rdf:type       spc:Headscan;
                               dc:format      spc:NRRD;
                               spp:manualSegmentation
                                              xnat:seg2.
                               spp:patient    xnat:patient1.
            xnat:patient1      rdf:type       spc:Patient.
            xnat:brainAtlasImg rdf:type       spc:BrainAtlasImage;
                               dc:format      spc:NRRD.
            xnat:brainAtlasMsk rdf:type       spc:BrainAtlasMask;
                               dc:format      spc:NRRD.
            xnat:seg1          rdf:type       spc:ManualSegmentation;
                               dc:format      spc:NRRD;
                               spp:patient    xnat:patient1.
            xnat:seg2          rdf:type       spc:ManualSegmentation;
```

```
                                dc:format      spc:NRRD;
                                spp:patient    xnat:patient2.
                        }"""^^sparql:GraphPattern].

xnat:goal1    rdf:type      sep:AbstractState, sep:GoalState;
              sawsdl:modelReference
        [ a           sep:AbstractPostcondition;
          rdf:value   """{
            ?tpm1           rdf:type        spc:TumorProgressionMap;
                            spp:mapContent  ?tpm_seq.
            ?reg_bag        rdf:type        rdf:Bag.
            ?tpm_seq        rdf:type        rdf:Seq;
                            rdf:_1          ?reg_mask1;
                            rdf:_2          ?reg_mask2.
            ?reg_mask1      rdf:type        spc:RegisteredBrainMask;
                            dc:format       spc:NRRD;
                            spp:headscan    xnat:headscan1;
                            rdfs:member     ?reg_bag;
                            spp:normalizedMask
                                            ?brainMask1.
            ?reg_mask2      rdf:type        spc:RegisteredBrainMask;
                            dc:format       spc:NRRD;
                            spp:headscan    xnat:headscan2;
                            rdfs:member     ?reg_bag;
                            spp:normalizedMask
                                            ?brainMask2.
            ?normMask1      rdf:type        ?norm;
                            dc:format       spc:NRRD;
                            spp:headscan    xnat:headscan1.
            ?normMask2      rdf:type        ?norm;
                            dc:format       spc:NRRD;
                            spp:headscan    xnat:headscan2.
            FILTER (?norm = spc:NormalizedBrainMask ||
                    ?norm = spc:RobustNormalizedBrainMask).
                      }"""^^sparql:GraphPattern].
            OPTIONAL {
               ?normMask1   spp:manualSegmentation
                                            xnat:seg1.
               ?normMask2   spp:manualSegmentation
                                            xnat:seg2.
            }
```

Listing A.14: Grounded precondition for executing meta components for a TPM example.

Based on the condition, we now illustrate the generated outputs of the respective semantic meta components.

```
_:pbag        rdf:type      rdf:Bag, sep:PolicyBag.

_:pol1        rdf:type      rdf:Seq;
              rdfs:member   _:pbag;
              rdf:_1        exp:BMG;
              rdf:_2        exp:BN;
              rdf:_3        exp:BR;
              rdf:_4        exp:TS;
```

```
                rdf:_5        exp:TPM.

_:pol2          rdf:type      rdf:Seq;
                rdfs:member   _:pbag;
                rdf:_1        exp:BMG;
                rdf:_2        exp:RBN;
                rdf:_3        exp:BR;
                rdf:_4        exp:TS;
                rdf:_5        exp:TPM.

exp:BMG       rdf:type      sep:SemanticExpert, spc:BrainMaskGeneration.
exp:BN        rdf:type      sep:SemanticExpert, spc:BrainNormalization.
exp:RBN       rdf:type      sep:SemanticExpert, spc:RobustBrainNormalization.
exp:BR        rdf:type      sep:SemanticExpert, spc:BrainRegistration.
exp:TS        rdf:type      sep:SemanticExpert, spc:TumorSegmentation.
exp:MG        rdf:type      sep:SemanticExpert, spc:MapGeneration.

xnat:goal1    rdf:type      sep:State, sep:GoalState, sep:AbstractState.
xnat:start1   rdf:type      sep:State, sep:StartState, sep:GroundedState.
```

Listing A.15: Example output of the semantic abstract planner for the example condition.

```
_:ebag          rdf:type      rdf:Bag.
exp:BMG         rdf:type      sep:SemanticExpert, spc:BrainMaskGeneration;
                rdfs:member   _:ebag.
xnat:grounding1
                rdf:type      spc:SemanticExpertExecution;
                sep:expert    exp:BMG;
                sawsdl:modelReference
          [ a           sep:GoundedCondition;
            rdf:value   """{
              xnat:headscan1     rdf:type       spc:Headscan;
                                 dc:format      spc:NRRD.
                                 spp:patient    xnat:patient1.
              xnat:patient1      rdf:type       spc:Patient.
              xnat:brainAtlasImg rdf:type       spc:BrainAtlasImage;
                                 dc:format      spc:NRRD.
              xnat:brainAtlasMsk rdf:type       spc:BrainAtlasMask;
                                 dc:format      spc:NRRD.
                        }"""^^sparql:GraphPattern].
xnat:grounding2
                rdf:type      spc:SemanticExpertExecution;
                sep:expert    exp:BMG;
                sawsdl:modelReference
          [ a           sep:GoundedCondition;
            rdf:value   """{
              xnat:headscan2     rdf:type       spc:Headscan;
                                 dc:format      spc:NRRD.
                                 spp:patient    xnat:patient1.
              xnat:patient1      rdf:type       spc:Patient.
              xnat:brainAtlasImg rdf:type       spc:BrainAtlasImage;
                                 dc:format      spc:NRRD.
              xnat:brainAtlasMsk rdf:type       spc:BrainAtlasMask;
                                 dc:format      spc:NRRD.
                        }"""^^sparql:GraphPattern].
```

```
34 ||
35 || xnat:goal1    rdf:type      sep:State, sep:GoalState, sep:AbstractState.
36 || xnat:start1   rdf:type      sep:State, sep:StartState, sep:GroundedState.
```

Listing A.16: Example output of the semantic grounded planner based on the abstract plan for the example condition.