Saarland University

Department of Language Science & Technology

Bachelor Thesis

Identification of Processing Steps and their Arguments in German Recipes

Author:
Theresa Schmidt
Matriculation: 2565903

Supervisors:
Prof. Dr. Alexander Koller
Dr. Arne Köhn

Date of submission: 10th January 2020

Declaration of Authorship

I hereby declare that I am the sole author of this thesis. I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise are fully acknowledged in accordance with the standard referencing practices.
I declare that this is a true copy of my thesis, including any final revisions, and that the digital and printed versions are identical.

This paper was not previously presented to another examination board and has not been published.

Saarbrücken,
10.01.2020

City, date

Signature

Abstract

As the interest in home cooking rises, so, too, does the need to explore the recipe domain and find ways to make recipes machine-readable. In this thesis, I look at key characteristics of the German recipe domain and develop a novel detailed graph representation for recipes. I also explore ways to parse recipes into these structures. I annotate a small corpus of German recipes with sequences and dependency relations relevant for modelling recipes. Using a BiLSTM and CRF model, I outline a sequence tagging task which scores an F1 Measure of 80.2. An off-the-shelf dependency parser displays an LAS of 78.3. More steps are needed to build a complete and applicable recipe parser but this work serves as a solid foundation for many future lines of research and implementations that can utilise standardised recipe graphs and associated parsing methods.

Acknowledgements

I thank my supervisor Prof. Alexander Koller for his methodological and substantial advice. Likewise, I thank Matthias Lindemann, for help with the implementation and for providing me with a basic understanding of some of the concepts used.

Contents

Chapter 1

Introduction

Food is, and always has been, a central human interest. Evidence for cooking can be found in excavations of Neanderthal sites. Even thousands of years ago, in Mesopotamia, people were already writing down recipes (Kaufman, 2006). Since then, cooking has evolved under different influences such as agricultural conditions, religious believes and technological innovations. In times of *#foodporn*, food and cooking are as popular as ever. New technologies provide us with new possibilities to browse for recipes and share our experiences. With the internet, recipes have become much more accessible, as we can easily search huge databases for our favorite dishes. Nowadays, we cannot only read recipes, but listen to cooking podcasts or watch instructions on YouTube. Nevertheless, whichever medium we choose, recipes are still mostly presented as linear text, and search engines let us search for recipe titles or a choice of ingredients, at most. This is not comfortable and it is outdated, too. In this high-tech age, users want to be able to ask a cookbook situational questions like *"What should I do next"* or *"What does sauté mean?"*. As dialogue assistants are integrated into homes more and more often, giving them the skill to process recipes is required. As of yet, however, there are no sufficient solutions to this problem.

This research takes the first steps towards making recipes machine-readable, such that dialogue systems, as well as various other applications, can be implemented for the recipe domain. Concretely, I investigate German recipes with regard to linguistic characteristics and their underlying structure. I conclude that recipes describe a cooking process as a system of intermediate products which are transformed into one another by certain actions involving ingredients and tools. I find the best representation for recipes to be graphs like the one in figure 1, which are reminiscent of data flow diagrams. These graphs depict the work flow and product flow for instruction steps, ingredients and tools, which I identify as the key elements of a recipe. Additionally, further details and explanations are included to ensure that a cooking process can be reproduced from the graph as successfully as from the original recipe. The edges in such a graph are labelled so as to make the relations between its nodes clear. To show how such graphs can be obtained from recipe text, I annotate a small corpus of German recipes and lay out a concept for parsing recipe text into structured graphs.

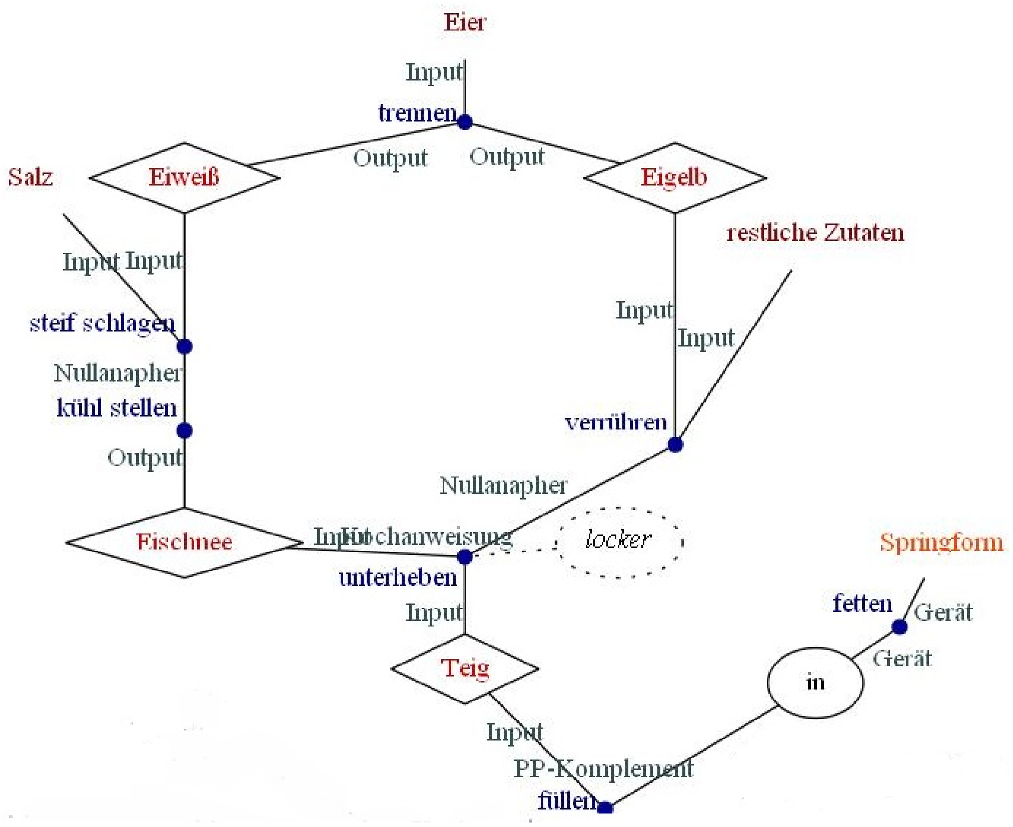


Figure 1. Recipe in graph representation. See details for the construction of this graph in Chapter 4.

In the long run, recipe graphs can be used for various purposes in research and commercial applications. A recipe parser can also come in handy for adding new recipes into the database of script-based systems. Some kitchen devices like Thermomix or microwave ovens use recipes in a specific format to execute recipes. With a suitably configured recipe parser, the transformation of users' own recipes into the required format can become very easy. More applications for recipe processing tools will arise in autonomic robotic cooking. This field still is still developing and not yet available to households.

Moreover, standardised recipe representations can make recipe comparison easier, as Chang et al. (2018) have shown. However, it is not only the comparison between recipes that is relevant - finding overlaps between similar recipes and combining them can yield very fine-grained recipe databases. Apart from that, databases will allow for more search options like food recommendations, for home cooking and eating out, or like searching for specific cooking steps performed in a specific way or on specific ingredients (Trattner and Elsweiler, 2017). For example, such a query could be *"Find a recipe with sliced onions instead of chopped onions"* or *"Are there recipes for warm dessert made from apples?"*. Comparing recipes can also tell us which steps in the cooking process need to be mentioned explicitly and which can be left out, e.g. in recipes aimed at more experienced cooks. In terms of applications, this means that recipes can be adjusted to match individual levels of expertise, depending on the user.

Finally, graph representations can aid cross-lingual and translation tasks. A tool that expands a recipe graph back into natural (sounding) language can be combined with a translation tool on the graph itself.

This research is very domain-specific. Nevertheless, I expect instructions and manuals from various domains to be quite similar to recipes. The development of a recipe parser can be the basis for the development of parsers for other domains.

This thesis is structured as follows: In Chapter 2, I discuss related work in the recipe domain, which focuses primarily on English. Chapter 3 analyses characteristics of the recipe domain. The above mentioned graph representations are introduced in Chapter 4, where I also discuss details about the annotated corpus. I describe and discuss identifying labelled sequences with Neural Networks in Chapter 5 and outline how dependency parsing can be utilised to find relations in recipe data. In Chapter 6, I illustrate further steps in making recipe parsing applicable and provide an overview of possible research based upon recipe graphs.

Chapter 2

Related Work

The domain of recipe processing offers many opportunities. Nevertheless, it has not yet been explored well. There are some weakly linked approaches to partial problems of recipe processings, mostly for English recipes, whose ideas I connect and develop in this thesis.

Ingredients A set of ingredients can already give some idea about the nature of a dish and its preparation. Hence, ingredient parsing receives much attention in the field of processing recipes. In recipe text, ingredients often are specified very accurately. Ingredient phrases very often contain indications of quantity or comments like *fresh* or *sliced.* Greene (2015) designed a CRF parser which breaks down ingredient phrases like (1) in English recipes into their functional components, which are annotated with either the tag *NAME, UNIT, QUANTITY, COMMENT* or *OTHER*, according to the components' functionality.

(1)	1	Tbsp	granulated	sugar
	QUANTITY	UNIT	COMMENT	NAME

In this manner, ingredient phrases can be converted into a standardised format. This is one of several parsing steps in recipe processing.

Silva et al. (2019) adopt the ingredient phrase tagger and build a model, also for English recipes, that associates ingredients from the list of ingredients given in the recipe with their respective recipe steps. They discriminate between *preparation techniques* (purely physical, e.g. *cutting*) and *cooking methods* (qualities of ingredient(s) are changed e.g. *frying*) to allow detailed insights into what happens to the individual ingredients. For the preparation techniques, they achieve 0.93 F1 Measure, and for the cooking methods, 0.73. They also point out that parsing for dependencies in a pre-processing step can improve the performance of their English language model.

Tree Structures. A different take on associating actions with ingredients is to structure them together in dependency trees. Jermsurawong and Habash (2015) do this for English recipes with the goal of offering new approaches to browsing recipe databases. They do not concentrate on the individual cooking steps as much as they aim to connect the steps to each other. Their model generates trees of sentences with the ingredients from the ingredients list as leaves. They do not perform any further parsing on the recipe text. The parser relies upon a complex rule-based heuristic; edges are generated based on over 1000 features. A Support Vector Machine is used to rank the edges and choose the best one for every component. From their results, Jermsurawong and Habash (2015) conclude (among other things) that while stemming the input might facilitate the computation of edges, it also leads to new mistakes in the output.

Tasse and Smith (2008) address the question of how recipes can be formalised into a simplified formal language. They introduce *MILK* (Minimal Instruction Language for the Kitchen), where recipes are interpreted as a series of world states. These states consist of ingredients, tools and additional description and their relations. Twelve actions are defined which can change states. All actions that occur in recipes are abstracted into these twelve classes. The initial state is empty. Ingredients and tools can be added and combined into intermediary products, which then become ingredients themselves. In the final state, all tools and ingredients are consumed except for one ingredient, which is the dish described by the recipe. The *MILK* language is not a graph language per se, but relating these states according to dependency will result in a graph structure. What makes Tasse and Smith (2008)'s work unique is that they give names to the outputs of every single action, even if these names are not mentioned in the recipe. As previously mentioned, world knowledge about the denomination of outputs can be learned automatically from comparing graphs of different recipes. *MILK* is language independent, in principle. Tasse and Smith (2008) report an approach to parsing English recipes with a Naive Bayes Classifier but admit to rather weak performance.

Chang et al. (2018) also introduce tree representations for recipes. Their main goal is to capture the overall structure of a recipe rather than its detailed information. Their trees for English language recipes consist of a linear arrangement of actions which can have additive edges towards input ingredients and are computed by a CRF. Chang et al. (2018)'s recipe trees do not include tools and detailed instructions. All in all, they are highly simplified representation of recipes which are not comprehensible on their own.

Applications. Chang et al. (2018) not only present these tree structures for recipes - their main focus lies in the concrete application *RecipeScape*, a graphical interface where many different recipes for one dish can be compared at the same time. As recipes are processed in tree form, *RecipeScape* is basically language independent. The platform is aimed at experienced and professional cooks. It relates recipes to each other based on their structure and the used ingredients. For one dish, about 500 recipes are displayed as a recipe cloud. They are divided into six clusters according to pairwise tree similarities. Users can easily identify the most common and also the most unusual approaches in the database for a class of dishes. For the most frequent actions and ingredients, *RecipeScape* computes statistics of when they occur on a timeline in each cluster. Additionally, individual recipes can be selected to show their structure and text. *RecipeScape* also shows overlaps between two recipes at a time regarding their structure and set of ingredients.

Researching recipe text does not necessarily yield universally applicable results for different languages. Research into parsing German recipes is very sparse. There is one approach for a neural model[1] to recognize ingredients in recipe text. Apart from that, there are two unrelated implemented approaches to what a dialogue assistant might look like in the kitchen domain. Both systems, Kochbot (Schäfer et al., 2013) and Paschta[2], limit parsing to the ingredient list. These systems can thus only answer a limited set of questions and are only contingently useful as actual kitchen tools. On the other hand, their existence *and* their shortcomings show general interest in the cooking domain and applications therein, as well as the need for computational linguistic research and developments in the recipe domain.

[1] See `https://www.depends-on-the-definition.com/identify-ingredients-with-neural-networks/`

[2] See https://github.com/dialogos-saar18/Paschta/

Chapter 3

Characteristics of the Recipe Domain

In this chapter, I introduce some aspects that make recipes a special and interesting domain. Note again that, while some of the presented features do hold for recipes in other languages, I base the following statements on recipes written in German.

3.1 Temporal Order

Aligning a recipe's individual processing steps into the best order of actual execution can pose computational challenges. Models need to capture domain-specific characteristics on the levels of input processing, graph construction and output for a user application.

Input. Recipe instructions cannot be assumed to be given in exactly the order in which they should be performed (Malmaud et al., 2014). There are two types of instructions in German recipe text that regularly disobey the temporal ordering of food preparation steps: participles and the cluster of advice, hints and suggestions. Such remarks are typically provided at the very end of a recipe and can refer to any action in the whole cooking process, thus causing long distance dependencies and unclear classifications. Many instances of additional information are only ornamental and cannot be considered necessary information for the actual execution of a recipe. Not all participles contain cooking steps relevant to the recipe. For example *frozen peas* or *minced meat* are fixed ingredients that are usually bought with the required qualities. Also, participle function do not constitute actions when they have adverbial meaning such as in (2).

(2) Die Zwiebeln und die Kartoffeln *getrennt* schneiden.
'Cut onions and potatoes *separately.*'

However, participles can and do contain cooking actions that have to be included in the recipe graph. Many participles (e.g. *boiling water* or *cubed tomatoes*) contain cooking actions which should be performed in the preparation phase of the cooking. Compound nouns can sometimes show similar effects: *'onion rings'* implies *'use the onion rings you have prepared in an earlier step'*. A recipe parser will have to distinguish between participles that are cooking actions and those that aren't.

(3) a. Die Eier trennen,
'Separate the eggs,'
b. das Eiweiß mit Salz steif schlagen,
'beat the egg whites with salt until stiff'
c. und kühl stellen.
'and put in a cool place.'
d. Das Eigelb mit den restlichen Zutaten verrühren,
'Mix the egg yolks with the remaining ingredients,'
e. den Eischnee locker unterheben.
'gently fold in the beaten egg whites.'
f. Den Teig in eine gefettete Springform füllen [...]
'Pour the batter into a greased (springform) baking pan [...]'

Graph Construction. In general, a cooking process is a series of stages or intermediate products that occurs by applying various actions (Jermsurawong and Habash, 2015) to some set of ingredients and tools (with the output of the last action being the finished dish). Although the order of these actions appears linearly in the recipe text, closer inspection reveals that hierarchical structures are better suited to describe the order of these actions. Jermsurawong and Habash (2015) define this hierarchical order from the angle of a dependent action (i.e. an action that is performed after another action). An action A depends on another action B if B is a necessary condition for the execution of A. Contrary to Jermsurawong and Habash (2015)'s position, I argue in this thesis that an action cannot only depend on various previous actions but may also have more than one succeeding action that depend on it directly. For example, in (3-a)[1], separating eggs creates two outputs: egg whites and egg yolks. Consequently, (3-b) and (3-d) depend directly on (3-a). Independent action sequences should be represented as non-connected branches in the recipe graph as opposed to a linear sequence implied by the original text. For (3) that means, (3-d) should not depend on the precedingly mentioned (3-c), which it is independent of. Instead, it depends only on (3-a). The independent threads of what happens to the egg whites and the egg yolks, respectively, are only joined in (3-e), which depends on both, (3-c) and (3-d) (see also figure 2, where the recipe (3) is depicted as a graph).

Text Generation. In an application environment (e.g. a dialogue assistant presenting instructions to the user), the graph representation will have to be re-linearized. With parallel

[1] (3) is an excerpt from the recipe "Käsekuchen ohne Boden" (URL: `https://www.chefkoch.de/rezepte/1748391284126423`).

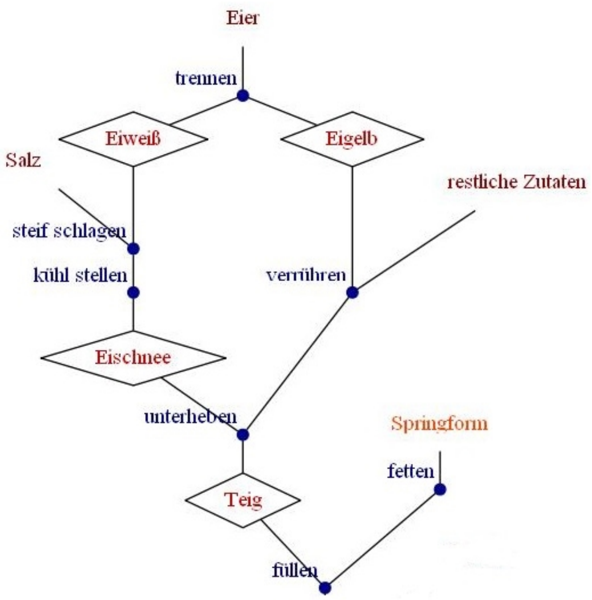


Figure 2. Simplified recipe graph for the recipe in (3). Actions are depicted as blue nodes, ingredients as leaves with red font, tool as leaves with orange font, and intermediate products as diamond shaped nodes.

threads such as in (3) and figure 2, this is not a trivial task. An intuitive approach would be to decide on a random ordering of the parallel actions, e.g. from left to right, then from top to bottom. This approach would be fine for the example in figure 2. However, (3-c) implies that the cooling will take some time. Therefore, it is better to first process the egg whites before addressing the egg yolks, as is also suggested by the order in the original text (3). This random ordering heuristic can sometimes lead to mistakes, too. For example, some actions must be performed at the same time. Usually, this concerns steps that take some time to finish but do not take much attention. For example when making a dish with roast meat, the roast should be put into the oven before the sides are prepared so that everything is finished approximately the same time. On the other hand, some steps have to be performed last in a set of parallel actions when it is important that the next action is performed immediately after this step (e.g. a batter that has to be put into the oven as soon as it is finished).

3.2 Language

Recipes constitute a very specific domain of language and thus present limited linguistic variety, while showing characteristic key features that the define the domain and are less common in standard German. These feature are worth analyzing for this project.

Style. Typically, the grammatical mood in recipes is the infinitive (with imperative meaning) just as in examples (4) and (5). In the corpus of 100 recipes created as part of this project, only 3 recipes do not use infinitive forms, instead using primarily passive voice. Additionally, although there are some declarative sentences scattered in the data, almost all of these do not contain cooking instructions. Rather, they contain specifications that are included in the corresponding graphs as action-specific instructions, if they are included at all. Most of these sentences contain modal restrictions like *can*, *should*, *If you like*, *until*, or *if*. In even fewer recipes, the occasional passive construction can be observed.

(4) a. Trockene Zutaten vermischen.
'Mix dry ingredients.'
b. Butter, Eier und Milch hinzufügen.
'Add butter, eggs and milk.'
c. Gründlich verrühren.
'Stir thoroughly.'
d. Den Teig in eine Backform geben.
'Put the batter in a baking pan.'

(5) a. Alle Zutaten mit dem Schneebesen zu einem Teig vermengen.
'Combine all ingredients into a batter with a whisk.'
b. Eine Stunde ruhen lassen.
'Let rest for one hour.'

Verb Ellipses. In some sentences, verbs are elided. This never happens in more than two consecutive sentences. Still, this phenomenon makes graph construction harder for such sentences as there is no lexical anchor for the implied action.

(6) Auf einem Teller anrichten, Petersilie darüber, fertig.
'Dress on a plate, parsley on top, done.'

(7) Die Nudeln abgießen und zurück in den Topf.
'Drain the pasta and back into the pot.'

If they are not elided, cook actions can be found in sentence predicates and in past participles like *'chopped onions'* or *'salted meat'*. However, not all past participle forms correspond to recipe actions. This is true for tools (e.g. *rounded knife*) and ingredients (e.g. *frozen spinach*).

Articles in Noun Phrases. Noun phrases in recipes often do not demand definite articles as strongly as they usually do in other domains. In fact, the choice between definite and zero articles seems quite arbitrary. In (4), the zero articles could be exchanged for definite articles and vice versa in all noun phrases without causing the text to become less characteristic of the domain. Indefinite articles usually can't be replaced by the zero article. Apart from that, many noun phrases follow the scheme numerical-unit-noun (Greene, 2015). Both aspects are not very typical of other domains. Therefore, pre-trained models imported from other domains might struggle with noun phrases in recipes.

3.3 Referencing Inputs

A verb's grammatical arguments usually are also the arguments or outputs of the actions corresponding to the respective verb. This observation does not hold in opposite direction. Indeed, many action inputs, and sometimes also tools, are not made explicit in recipe text. Also, an action's output might be mentioned in a different section of a recipe than the action in which the output is produced as can be observed with *"Eischnee"* (*'beaten egg whites'*), which is the output of (3-c) but is first mentioned explicitly in (3-e). References like this can stretch much longer than in this example, causing hard to resolve long distance relations.

Anonymous Objects. Intermediate products are not consequently mentioned in recipe texts. This leads to two related phenomena connected to referencing issues. The first phenemona, anonymous objects, refers to objects that are not make explicit in the recipe when they first appear. Often, these objects are named in another part of the recipe, when they are referenced. For example, (4) shows a prototypical example, where *"[der] Teig"* in (4-d) is an anonymous object referring to the result of the action performed in (4-c).

Anonymous objects can also emerge at the interface between recipe text and the list of ingredients. Ingredients can be referred to by collective expressions (Jermsurawong and Habash, 2015). There are collections that can be resolved from analyzing the list of ingredients and the context such as '*the ingredients for the dressing*' if there is a separated section '*For the dressing*' in the ingredients list, or something like *"alle Zutaten" ('all ingredients')* in (5-a). Other collections need further world knowledge to identify the referenced ingredients. For example, the concept of *"dry ingredients"* (*"trockene Zutaten"* in (4-a)) cannot be resolved from the context alone. Similarly, single ingredients can be referred to by hypernyms. For example in a recipe for Fish&Chips, the ingredients list might specify *'cod'*, while the instructions refer to the same ingredient as *'fish'* in one or more places. As investigating the list of ingredients is not in the scope of this thesis, only the basic type of anonymous objects are covered here in the annotation and parsing tasks.

Zero anaphoras. Secondly in recipe texts, zero or null anaphoras (Malmaud et al., 2014) are used much more often than in most other domains. In fact, they are very common. For example in the training split of the corpus introduced in this thesis, there are roughly 800 sentences and 729 zero anaphoras (see tables 4 and 6). This special type of ellipses omits the direct object of a verb. The object position is filled implicitly by the immediate context. In the recipe domain, the object of one action is usually either the same as in its preceding action or it refers to the result of this preceding action. This results in the movement of a verb's arguments outside of sentence boundaries. In this manner, the continuant character of the cooking process is reflected in the syntactic structure of its textual representation. The remote referent can be explicitly named like *"Teig"* (*'batter'*) in (5-a), which is the referent and object of *"ruhen lassen"* (*'let rest'*) in (5-b) or stay anonymous like the result of *"hinzufügen"* (*'add'*) in (4-b) which is referred to by *"verrühren"* (*'verrühren'*) in (4-c).

Chapter 4

Graph Structures for Recipes

A recipe can be interpreted as a (textual) description of a cooking process. Although recipes can account for many characteristic traits, there is still considerable variation in style and form within the domain. In order to make recipes machine-readable, a different, more restricted representation than common recipe text is needed. As described in Chapter 2, previous approaches introduced different takes on the representation problem. There are representations that focus more on structure (Jermsurawong and Habash, 2015; Tasse and Smith, 2008) while others concentrate on formalization (Chang et al., 2018; Silva et al., 2019; Greene, 2015; Tasse and Smith, 2008). Each representations scheme is designed to address a specific application. All of them are designed with specific applicational aspects in mind.

In this chapter, I present a new formal representation of recipes which unifies and expands these previous suggestions. The goal is to develop a recipe representation that is universally applicable in different fields of research and applications. To achieve this goal, I propose a methodology and annotation scheme whereby recipes are transformed into graphs that balance abstract and necessary information. My proposal also keeps costs for annotation and parsing relatively low.

4.1 Recipe Representation

While many recipes have an underlying tree structure, I observe that trees are not perfectly suited to capture phenomena such as ambivalent sequences or actions with multiple products (remember figure 2). Instead, I propose a new graph representation for recipes.

Integrating ideas from previous work, I develop a unified structure of cooking actions and the ingredients they use. The individual actions can be arranged into dependency structures as shown in figure 3. Parallel to these ingredient combination graphs, I identify a second flow of information in the preliminary outputs (action products) in each recipe (figure 4) (Tasse and Smith, 2008). This development of ingredients and products is not as obvious as that of ingredients and actions. While actions are almost always mentioned explicitly in the recipe,

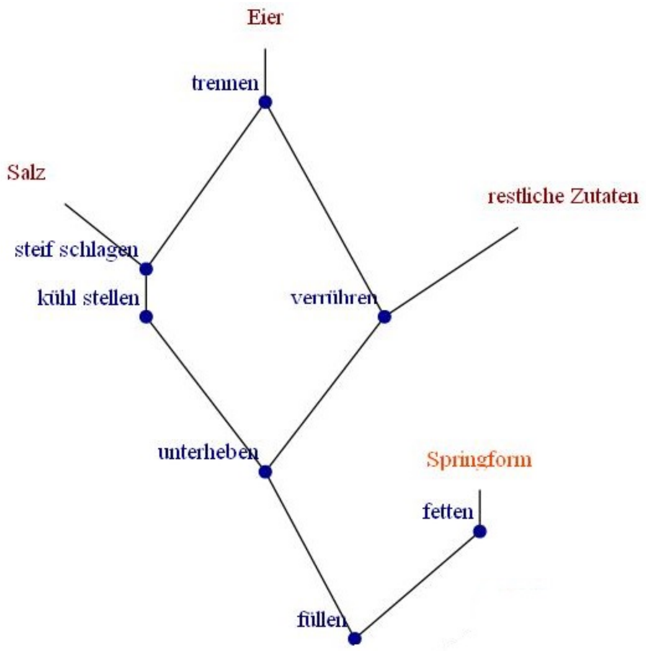


Figure 3. Action graph for the example recipe in (3).
This graph is motivated from bottom to top, connecting actions (blue nodes) to the actions, tools (orange leaves) and ingredients (red leaves) they depend on. The bottommost action (*"füllen"*) depends on all other actions and on all tools and ingredients, whereas tools and ingredients are depicted as leaves because they do not depend on anything.

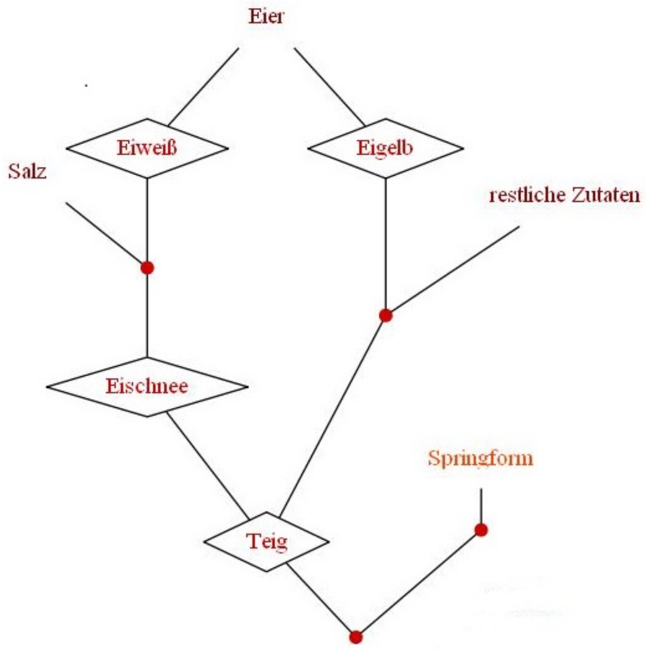


Figure 4. Product graph for example recipe.
This graph shows the states of ingredients and tools over the course of the cooking process. It can be read from top to bottom as inputs defining consecutive states as output products. Where products are named explicitly, they are depicted as diamond shaped nodes, while anonymous products are represented by dots. To reach a new (farther down) state, inputs can be combined, split (eggs to egg whites and yolks) or transformed.

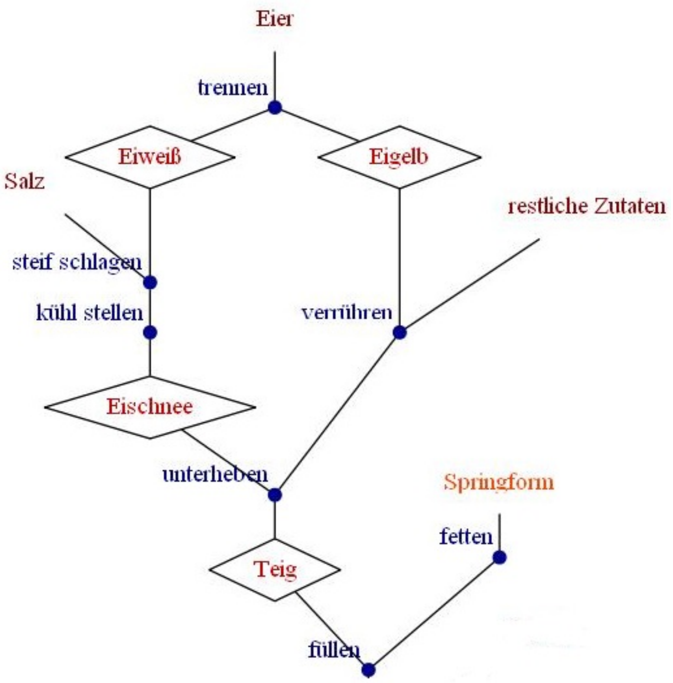


Figure 5. Merged action and product graph.
Anonymous product states are not depicted; named ones are inserted between the action they are resultative of and the action that depends on them (e.g. separating eggs has egg whites as output which are the input of beating egg whites.

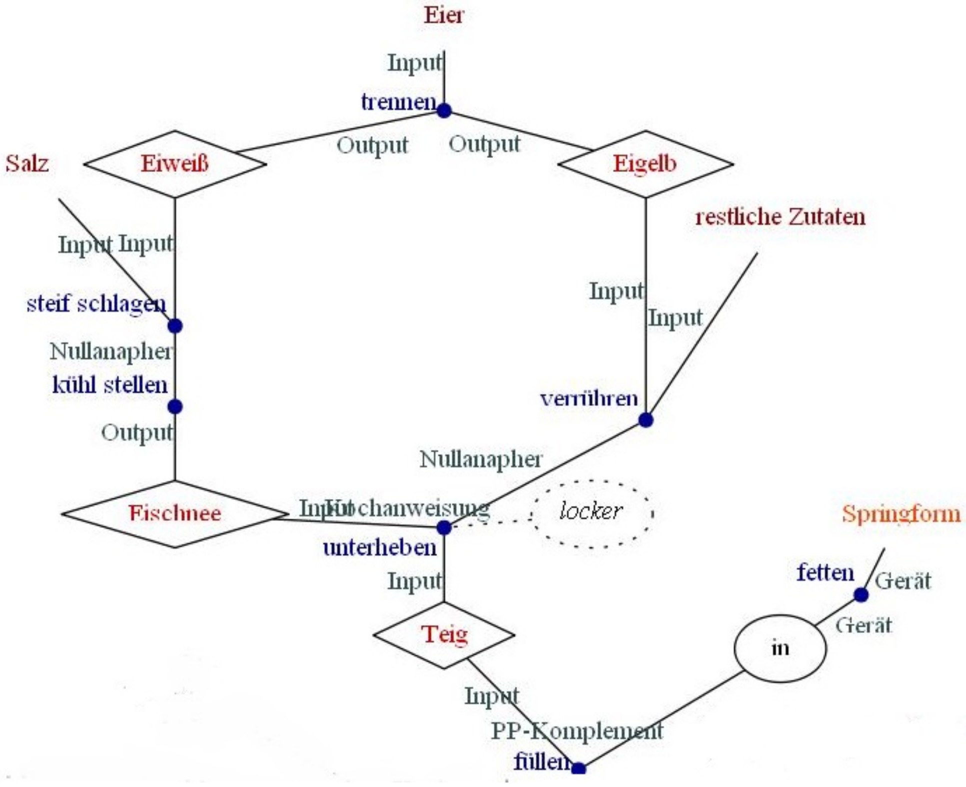


Figure 6. Full representation.
Here, the relations between all nodes are labelled. Also, further types of nodes add necessary details from the original recipe: specifications (dotted black) and prepositions (black ellipses).

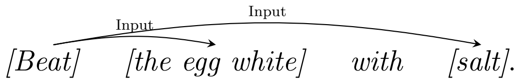
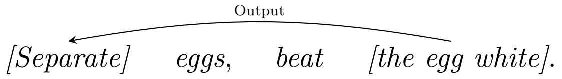
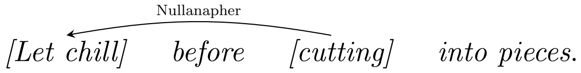
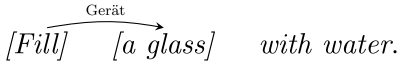
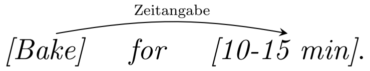
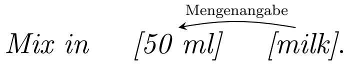
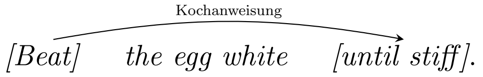
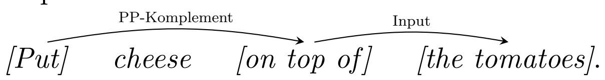
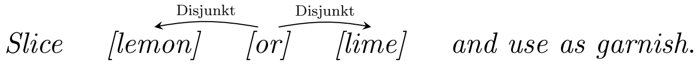
Dependency Relation	English	Explanation & Example
Input	Input	Connects an action or preposition to its input.  *[Beat] [the egg white] with [salt].*
Output	Output	Connects products to the action they are an output of.  *[Separate] eggs, beat [the egg white].*
Nullanapher	Zero Anaphora	Like *Input* but between two action nodes. Applies when input is not mentioned explicitly.  *[Let chill] before [cutting] into pieces.*
Gerät	Tool	Like *Input* but for tools.  *[Fill] [a glass] with water.*
Zeitangabe	Time Span	Relates an action and adverbials with temporal constraints.  *[Bake] for [10-15 min].*
Mengenangabe	Quantity	Relates an ingredient or tool to where its quantity is specified.  *Mix in [50 ml] [milk].*
Kochanweisung	Cooking Instruction	Relates adverbial phrases (that are not temporal) to actions.  *[Beat] the egg white [until stiff].*
PP-Komplement	Prepositional Complement	Points from an action to the preposition of its complement.  *[Put] cheese [on top of] [the tomatoes].*
Disjunkt	Disjunct	Marks the disjuncts in a disjunction.  *Slice [lemon] [or] [lime] and use as garnish.*

Table 1. Dependency relations used in the annotation task.

Label	English	Explanation & Example
Zutat	Ingredient	First mention of an ingredient. *Pour lemon juice over the fried fish.*
Zwischenprodukt	(Preliminary) Product	Anything that is the result of an action. *Pour the batter into a greased baking pan.*
Gerät	Tool	First mention of a tool. *Drape a damp cloth over the bowl.*
Kochschritt	(Cooking) Action	Any expression describing an action. *Pour the batter into a greased baking pan.*
Bedingung	Specification	Measures of quantity, time spans, adverbial remarks. *Mix in 100 g flower and immediately bake for roughly 20 min at 170° C.*
Präposition	Preposition	Prepositional phrases in prepositional complements to action verbs. *Put cheese on top of the tomatoes.*
Disjunktion	Disjunction	Disjunctives. *Slice lemon or lime and use as garnish.*

Table 2. Sequence labels used in the annotation task.

action products are often referred to much later in the recipe than when they are produced, if at all[1]. The final product is usually introduced in the recipe's title. Apart from titles, recipes are not obliged to name the products of any action. The times that products *are* mentioned, is when the context is ambiguous or inconclusive about the input of a dependent action. Myproposed graph representation merges the actions graph with the output graph: Where an output is specified, it is included as node in the graph. Anonymous products are left out of the graph; for merged graph see figure 5.

Tools are often not included in abstract recipe representations although they are mentioned in recipe text frequently. The tools required for preparing a dish can be an integral part of a recipe. For example, inexperienced cooks might not know that there are special muffin tin to bake muffins in and that muffins will not be shaped nicely if they are baked without this tool. Another example is heating chocolate. If chocolate is heated to much, it will clot. To prevent this it should only be heated in a water bath. Without adequate information about which tools to use, cooking certain dishes successfully can be difficult. Hence, I consider tools to be crucial information in reproducing a cooking process (Tasse and Smith, 2008). I observed additional scenarios, where information about tools can also be relevant to users, in experiments with the *Paschta*[2] system, where test users frequently wanted to know *'How many dishes will I have to wash?'* or asked questions in the direction of *'I have only one pot. Will that be enough to cook this recipe?'*. Information about tools is most relevant for users when they are brwosing for recipes or planning a cooking process. This is especially true when they are unfamiliar with

[1] The implications arising from absent objects are discussed in Chapter 3.3

[2] `https://github.com/dialogos-saar18/Paschta/`

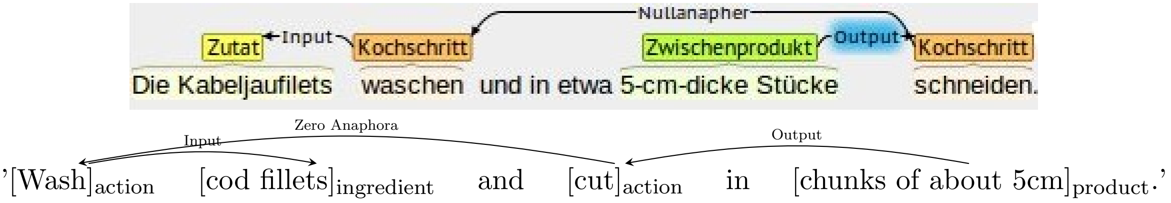

Nullanapher
Zutat
Input
Kochschritt
Zwischenprodukt
Output
Kochschritt
Die Kabeljaufilets waschen und in etwa 5-cm-dicke Stücke schneiden.
Zero Anaphora
Input
Output
'[Wash]action [cod fillets]ingredient and [cut]action in [chunks of about 5cm]product.'

Figure 7. Typical example for a *Zwischenprodukt* sequence as output of a *Kochschritt* sequence.

a recipe. The question of which tools are needed is particularly relevant for more elaborate recipes that require special tools not typically found in the inventory of a regular kitchen, or in circumstances where access to tools is restricted (e.g. on a camping trip). For these reasons tools ought to be included in a thorough recipe representation.

To complete the information displayed in my graphs, I add measures of quantity to ingredient, product and tool phrases. Additionally, I specify actions by adding adverbial phrases and time spans. For action verbs with prepositional complements, the preposition are included in the graph to mark the precise relation between the verbs and objects. Disjunctions are also accounted for although they don't occur too often. Dependencies are labelled, as well, even though they are often obvious from the types of the linked nodes. I deliberately include many details which can later be cut or simplified as needed for individual use-cases.

In conclusion, the proposed graph representation merges action dependencies with product development and adds details like quantities, time spans and modal specifications. The exact meaning of labels and relations is explained in tables 2 and 1. An example annotation with the complete graph representation is depicted in figure 6 for the shortened recipe (3) and in Appendix A for the full recipe.

4.2 Corpus Annotation

As part of this thesis, I present a pilot corpus of 100 recipes, annotated by a single annotator (myself). The recipe texts are written in German by non-professionals and were selected randomly from the *Chefkoch*[3] database. I consider only the recipe texts and discard titles and ingredient lists. The text is tokenized but no stemming or lemmatization is performed.

The text is tokenized and sequence tags are annotated in the *BIOUL* format [4], where each token is annotated with a tag that combines the label of the whole sequence that the token is part of as well as the position of the token in this sequence (Konkol and Konopík, 2015; Ratinov and Roth, 2009). In *BIOUL*, *B* stands for **b**eginning of a sequence, *I* for **i**nside, *O* for **o**utside, *U* for **u**nit and *L* for **l**ast. *I* tags may only appear between *B* and *L* tags. A sequence that starts with a *B* tag has to end with an *L* tag; single token sequences are annotated with *U*. The *O* tag is used for all tokens that do not belong into any sequence. For example, *B-Zutat* is the

[3] www.chefkoch.de

[4] The *BIOUL* annotation scheme is sometimes alternatively called *BILOU, BMEWO or BIOES*.

Die	B-Zutat
Kabeljaufilets	L-Zutat
waschen	U-Kochschritt
und	O
in	O
etwa	O
5-cm-dicke	B-Zwischenprodukt
Stücke	L-Zwischenprodukt
schneiden	U-Kochschritt
.	O

Table 3. Sequence annotation in *BIOUL* format for the sentence in figure 7, where the same sequences are indicated by coloured boxes.

	Recipes	Sentences	Tokens	Sequences	Labelled Tokens
Training	80	808	9494	3750	6381
Development	10	114	1234	473	811
Evaluation	10	122	1310	541	1016

Table 4. Sizes of the data set splits. The number of sentences is not precise because of shortcomings in the tokenization.

tag for the beginning of a *Zutat* sequence. Thus, the format captures sequence borders even for two consecutive sequences with the same label. See table 3 for an example annotation.

The corpus is split into training set, development set and evaluation set in a classical 80:10:10 random split (for statistics see tables 4, 5 and 6). It is available[5] in the popular CoNLL format (Tjong Kim Sang and De Meulder, 2003) with the annotated domain-specific tags in the XPOS column and dependencies in the HEAD, DEPREL and DEPS columns.

4.3 Interesting Substructures

The sequence type *Zwischenprodukt* comprises every object that has been transformed by an action. A *Zwischenprodukt* can be a mix of some ingredients, a different state of a single ingredient or a tool that has been modified. Moreover, it can be the combination of a tool and one or more ingredients. For example, a pot of soup may be referred to as *soup* or as *pot* in instructions like (8-a) or (8-b), respectively.

(8) a. 'Take the soup off the stove.'
b. 'Take the pot off the stove.'

[5] Download at `https://github.com/TheresaSchmidt/Recipe-Parser/blob/master/Corpus/pilotCorpus.7z`

Label	Training	Development	Evaluation
Zutat	774 | 1.5	119 | 1.4	118 | 1.7
Zwischenprodukt	469 | 1.8	74 | 1.9	92 | 1.9
Gerät	255 | 1.9	29 | 1.7	40 | 2.0
Kochschritt	1225 | 1.1	157 | 1.1	195 | 1.1
Bedingung	493 | 4.4	85 | 2.7	91 | 3.2
Präposition	305 | 1.0	43 | 1.0	49 | 1.0
Disjunktion	29 | 1.0	2 | 1.0	8 | 1.2

Table 5. Number of labelled sequences (left) and average sequence length (right) for each label with respect to the data set splits

Relation Type	Training	Development	Evaluation
Input	1313	203	221
Output	475	75	99
Nullanapher	729	88	112
Gerät	318	34	48
Zeitangabe	184	22	21
Mengenangabe	138	19	13
Kochanweisung	402	48	58
PP-Komplement	304	43	49
Disjunkt	32	4	12

Table 6. Number of occurrences of the individual relation types with respect to the data set splits

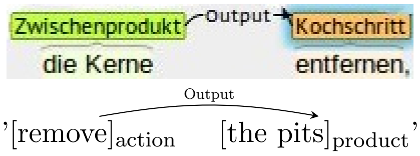


Figure 8. Product as output of a removing action

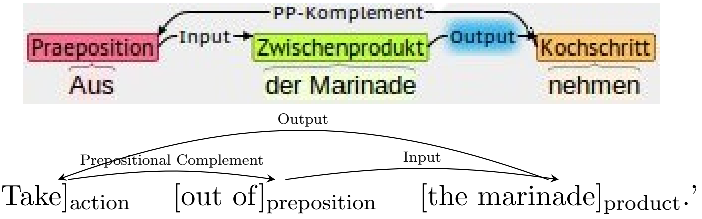


Figure 9. (Apparent) cycle.

This phenomenon can explained as pars pro toto, a stylistic device, expressing the logistic truth that moving the soup will entail moving the pot and, vice versa, moving the pot will result in moving the soup, too.

Although the label *Zutat* is used only for the first mention of an ingredient, multiple *Zutat* sequences can occur in one recipe for the same ingredient. This happens especially often with the ingredients flour and butter (Jermsurawong and Habash, 2015), and with spices, too, which are used as main ingredient in one place and also in small quantities for a different purpose like thickening a sauce or adding flavor to another component of the overall dish.

Verb phrases including an adverb are used in uniform manner in most but not all cases. There are some that are very unambiguously to be labelled as *Bedingung* and *Kochschritt* with a *Kochanweisung relation* (e.g. "$[grün]_{Bedingung}[einfärben]_{Kochschritt}$"). Others are very strong collocations and often alternatively written as one word (e.g. *"kleinschneiden"* and *"klein schneiden"*). I resolve to treating strong collocations as single sequences, and therefore label them as *Kochschritt* (e.g. $[klein\ schneiden]_{Kochschritt}$). There are also borderline cases like *glasig dünsten* (*'braise until translucent'*). This is the standard phrasing that appears in many recipes. Hence, it seems like a strong collocation. However, similar terms like *glasig anschwitzen* (*'sauté until translucent'*) or *weich andünsten* (*'braise until soft'*) show that it actually isn't that strong. I resolve to treating *glasig* as cooking instruction for *dünsten*: $[glasig]_{Bedingung}[dünsten]_{Kochschritt}$.

Products are not necessarily introduced as objects of the verb phrase describing the action they are produced by. If they are, it is often in the manner shown in figure 7. If parts of an ingredient are removed by an action, the removed part will also be a *Zwischenprodukt* (see figure 8). Products can also be introduced much later than they are produced in the cooking process; they are often mentioned first when they are used as input for another action (see table 1, Output).

In some cases, cycles occur in the corpus when ingredients are separated in the action and the input is identified by the same term as one of the outputs. In the example in figure 9, the expression *"die Marinade"* refers to two distinct states. In the first, something is being marinated. Hence it is placed in the marinade, most likely in a container. Here, the expression *"die Marinade"* refers to the whole complex of the object being marinated, the container, and the marinade itself, similar as in (8). This first state is the input to the action via the preposition, which is necessary to specify in what manner the input is related to the action. The second state is the output state, the marinade in the presumed container, which has probably been changed by the marinating process. This output marinade can be used later in the recipe in a dependent action. Although this phenomenon appears as a cycle in the corpus, the annotation will become more precise by splitting the sequence *"die Marinade"* into the two distinct nodes which it actually stands for in post-processing.

Chapter 5

Parsing Task

To make the proposed recipe representations available to actual applications, it is necessary to generate graphs from recipe text. To this end, I use the previously described corpus to develop a parsing methodology. As the corpus is annotated with labelled sequences and with labelled dependency relations, I identify two sub-tasks of parsing recipes: sequence labelling and dependency resolution. This thesis focuses mainly on sequence labelling but provides an outline for the dependency parsing, as well as pre-processing and post-processing.

5.1 Sequence Labelling

Sequence Labelling is used to distinguish between chunks of tokens which refer to entities of different functionality classes that cannot simply be classified by POS tagging. The task entails finding relevant sequences in the recipe text as well as classifying them into the classes that were introduced by the annotation. For the task at hand, the main objective of the sequence labelling model is to distinguish between ingredients, (preliminary) products, and tools, all of which can appear as noun phrases, along with recognizing instructional action terms, which mostly constitute verb phrases. These four classes are sufficient to model the general structure of a recipe. Any further tags are used for additional information needed to understand the recipe well enough to actually cook with it.

5.2 Description of the Implementation

The sequence tagger is implemented as a neural network (NN) with a BiLSTM encoder generating predictions in a CRF output layer. I compare similar models with two different pre-trained token embedders, ELMo and BERT.

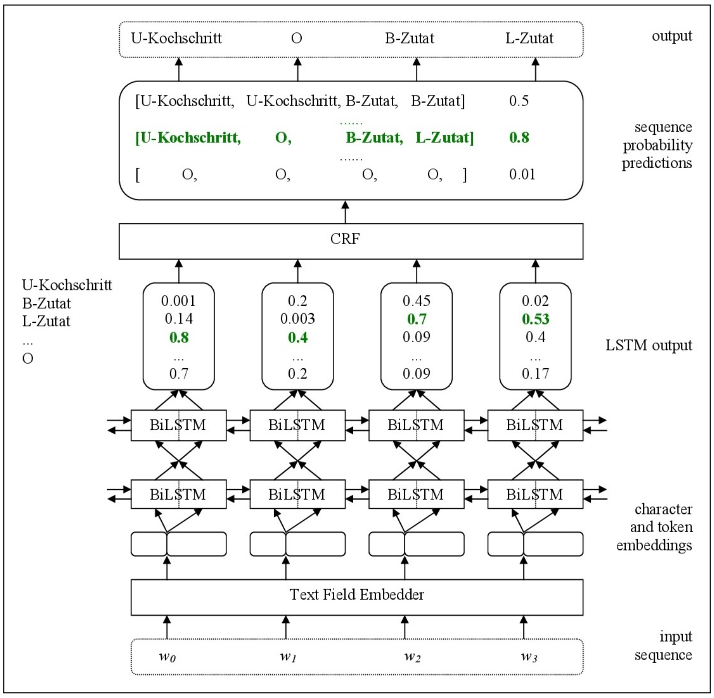


Figure 10. The full architecture of the tagger.
Character and token embeddings are generated for each input token and then concatenated. These embeddings are the input to both directions of the first LSTM layer. The output sequence predicted by the LSTM layer is [L-Zutat, L-Zutat,B-Zutat,B-Zutat]. It is overruled by the more likely prediction [U-Kochschritt, O, B-Zutat, L-Zutat] by the CRF in the output layer.

5.2.1 Text Field Embeddings

At the input level of the tagging model, embedding vectors are generated for each token in the recipe text. This model uses concatenated token and character embeddings. The character embeddings are generated by a Convolutional Neural Network. This is a type of NN especially well suited for recognising patterns and therefore perfect for the generation of embeddings of token characters (Vosoughi et al., 2016; Goodfellow et al., 2016). Using character embeddings in a NN is important for dealing with unknown or misspelt tokens in the input.

As token embedders heavily depend on huge amounts of training data, I used pre-trained token embeddings in the proposed model. This is possible because token embedders are rather general language models and thus neither very domain-specific nor very task-specific. Pretraining embeddings makes the training of the down stream model more time efficient while providing embeddings that have been trained on huge data sets. In this thesis, I compare pre-trained, deep contextualised ELMo (Peters et al., 2018) and BERT Devlin et al. (2018) embeddings, which are the most popular token embeddings in recent NLP tasks. BERT embeddings usually lead to better performance than ELMo embeddings for two reasons. Firstly, it is better at encoding context, and secondly, the pre-trained ELMo features are not changed in the training of the downstream model, whereas the BERT parameters will be fine-tuned to domain-specific input. It follows thatthe BERT model is more portable and computes more fine-grained embeddings than the ELMo model generally leading to better performance. On the other hand, ELMo is more efficient computationally.

5.2.2 Encoder

With the concatenated token and character embeddings as input, the encoder computes preliminary predictions for the task at hand. In a sequence to sequence task such as the presented tagging task, there is one output vector per input vector, i.e. one output vector per token. These output vectors indicate probabilities for the individual tags.

The encoder used for the tagging task is a bidirectional Long Short-Term Memory (BiLSTM) (Hochreiter and Schmidhuber, 1997; Schuster and Paliwal, 1997). Because of their recurrent design, (Bi-)LSTMs consider a token's context for the computation of its prediction with shared weights and biases for all time-steps - activations are passed on sequentially: They are more successful in learning long-distance dependencies than basic RNNs by means of using different gates (i.e. regulation functions) to evaluate the importance of the context information and the relevance of the current activations for the predictions in the next time step (Cai et al., 2019; Goodfellow et al., 2016). This thesis employs input, forget and output gates without peephole connections. The BiLSTM consist of two discrete LSTM layers executed on the input in opposite time directions (forward and backward). Their outputs weighted and added, and then passed on as input of the next layer (Schuster and Paliwal, 1997). Since the LSTM is two-layered and bidirectional, it will perform the following actions:

$$h_t^{(1)} = LSTM^{(1,f)}(x_t^0, c_{t-1}^{(1,f)}) \oplus LSTM^{(1,b)}(x_t^0, c_{t-1}^{(1,b)})$$

$$h_t^{(2)} = LSTM^{(2,f)}(\delta_t^{(2,f)} h_t^1, c_{t-1}^{(2,f)}) \oplus LSTM^{(2,b)}(\delta_t^{(2,b)} h_t^1, c_{t-1}^{(2,b)})$$

Where $h_t^{(i)}$ is the output of the $i - th$ LSTM layer at time step t (i.e. for the $t - th$ input token), $c_t^{(i)}$ is the corresponding cell state, $x_t^{(0)}$ is the output of the text field embedder and the input to the LSTM at time step t, $\oplus$ is the operator for vector concatenation, and $\delta_t^{(2,b)}$ is the dropout random variable applied to the inputs of the backward LSTM in the $2nd$ layer. $h_t^{(2)}$ is the output of the BiLSTM. It is passed on as input to the next layer of the model, the CRF layer.

5.2.3 Prediction Model

After the data has been processed by the encoder, label predictions could already be inferred from the output. For each token, the label corresponding to the highest value in the output vector would be chosen.

However in most cases, it is better to add an output layer to the NN to refine these independent predictions (Huang et al., 2015; Lample et al., 2016). The sequence tagger in this research implements a Conditional Random Field (CRF) (Lafferty et al., 2001) as output layer. Conditional Random Fields apply constraints for valid prediction sequences. For example in the BIOUL label encoding, a sequence may not start with an *L-X* or an *I-X* label. Apart from

Hyper-parameter	ELMo	BERT
Hidden size BiLSTM	50	200
Layers BiLSTM	2	2
Dropout BiLSTM	0.5	0.5
Dropout CRF	0.5	0.5
Learning rate	0.0075	0.001
Gradient norm	10.0	10.0
Training epochs	50	50

Table 7. Hyper-parameters for both models after fine-tuning.

these constraints, the model also learns transition probabilities between tags and includes them in the CRF. In the discussed implementation, start and end transitions are not included. The output sequence of the full model is predicted by the CRF layer as the most likely sequence of labels, given the preliminary probabilities computed in previous steps for individual tokens. Often, and in the discussed model as well, the Viterbi algorithm is used for this step (Ma and Hovy, 2016). The logged prediction probabilities $p(tags|inputsequence)$ are directly used as negative loss for the training.

The complete architecture of the NN model for the sequence labelling task is depicted in figure 10 (Li, 2017). I base this architecture on the implementation of an NER model[1] by the Allen Institute for Artificial Intelligence (Gardner et al., 2017).

After some, but not extensive, fine-tuning, I could raise the performance to 80.2 (ELMo) and 76.9 (BERT) with the hyper-parameters provided in table 7; see tables 8 and 9 for a more detailed breakdown of the performance measures.

5.3 Discussion of the Results

Contrary to common expectation, the model with ELMo word embeddings performs better than the model based on BERT. With most hyper-parameter configurations, the difference between both models fluctuates closely around 3 points. I hypothesize that this is due to the small size of the corpus in combination with the design of ELMo and BERT. More precisely, the fact that BERT embeddings encode better representations of the context than Elmo embeddings. It follows that BERT is more exposed to problems with sparse data. As the corpus is limited in size, it seems plausible for BERT not to perform at its best. I expect that with more data to train on, the parsing model will perform best with BERT embeddings, after all.

An alternative or complementary explanation for why ELMo performs better than BERT is that the BERT embeddings are pre-trained and then fine-tuned during the training of the downstream model. While this is generally a positive influence on portability because the model

[1] Described at: `https://github.com/allenai/allennlp/blob/v0.8.3/tutorials/getting_started/walk_through_allennlp/creating_a_model.md`

Model	F1 Measure	Precision	Recall	Accuracy
ElMo (development data)	82.3	81.6	83.0	83.9
ELMo (evaluation data)	80.2	78.9	81.6	78.7
BERT (development data)	79.3	79.3	79.2	80.1
BERT (evaluation data)	76.9	77.0	76.8	77.5

Table 8. Performance of fine-tuned models for ELMo and BERT embeddings. Best of four runs.

Label	Precision	Recall	F1 Measure
Zutat	70.6	81.4	75.6
Zwischenprodukt	69.5	60.0	64.4
Gerät	60.8	77.5	68.1
Kochschritt	96.3	98.6	97.4
Bedingung	71.9	70.3	71.1
Präposition	76.4	85.7	80.8
Disjunktion	83.3	58.3	68.6

Table 9. Detailed performance metrics on the evaluation data for the model with ELMo embeddings.

is not fitted to any specific task (as opposed to the feature-based approach), BERT embeddings are dependent on the fine-tuning the parameters to the domain. ELMo embeddings are feature-based and completely pre-trained. Because of sparse data, the fine-tuning cannot be fully accomplished which gives the model poorer performance.

Overall, the results of the downstream model with ELMo embeddings seem rather good, considering that both models were trained on a rather small corpus. The performance can be expected to increase further with more training data and exhaustive fine-tuning. Then, performance measures might become comparable to recent achievements in German sequence labelling tasks. Akbik et al. (2018) report somewhat better results than this research (F1 Measure: 88.3) as the current state-of-the-art for German NER on CoNLL-2003 data (Moon et al., 2019). The task and data are only related to sequence labelling on recipe text to a certain degree. In the CoNLL-2003 task, there were only 4 types (*LOC, PER, ORG* and *MISC*), compared to seven types in the recipe domain. On the other hand, with 553 articles 12,705 sentences and 206,931 tokens, the associated corpus is several times bigger than the corpus in this research (Tjong Kim Sang and De Meulder, 2003). Both differences make the sequence labelling in this study harder to learn for a neural network.

5.4 Label-Specific Performance

The label-specific performance measures of the ELMo based model (see table 9) look very much as expected. For instance, very good performance is achieved for cooking actions. Cooking actions are very short sequences - only 13% are longer than one token and 1.6% of *Kochschritt* sequences are at least three tokens long - which positively influences both, precision and recall. Also, they are verb forms most of the time. This will be reflected in their embedding representations and make them easily distinguishable from other sequence types, consequently bringing about a high precision. The high recall for actions is due to the reverse observation, namely, that almost all verb forms are also cooking actions.

Disjunctions can account for an acceptable precision, while their recall is relatively bad. This seems to be due to their sparsity in the training data. The model more or less learns a list of candidates for disjunctions. Therefore, many sequences labelled as disjunctions actually belong to this type, while tokens unseen in training cannot be recognised as hits. This hypothesis cannot be supported from the model's output for the evaluation data as the few instances of disjunctions are not enough for an error analysis. However, one interesting error is worth mentioning.

(9) a. [mit]$_{\text{Präposition}}$ [einem Zestenreißer bzw. Zitronenschaber]$_{\text{Gerät}}$
'[with]$_{\text{preposition}}$ [a zester$_1$ or zester$_2$]$_{\text{tool}}$'
b. [mit]$_{\text{Präposition}}$ [einem Zestenreißer]$_{\text{Gerät}}$ [bzw.]$_{\text{Disjunktion}}$ [Zitronenschaber]$_{\text{Gerät}}$
'[with]$_{\text{preposition}}$ [a zester$_1$]$_{\text{tool}}$ [or]$_{\text{disjunction}}$ [zester$_2$]$_{\text{tool}}$'

In (9-a) the disjunction and its disjuncts were parsed as one sequence and labelled as *Gerät* instead of the expected (9-b) where there are two tool sequences with a disjunction in between. I had considered annotating disjunctions like this with the prospect of further decomposing them in a post-processing step. However, I refrained from doing so as some disjunct phrases can become very complex and stretch over long segments of text.

The performace for *Zutat*, *Zwischenprodukt* and *Gerät* sequences is not very high. This can be ascribed to the big overlap between the three categories. Instances of *Zwischenprodukt* are ingredients and/or tools which have already been modified by previous actions. Oftentimes, they are referred to by the same expressions. In (8), *'pot'* doesn't refer to the tool itself but to the complex product created by performing certain steps with the goal of cooking soup. Hence, mentions of products can be confused for ingredients or tool resulting in a lowered recall for products and a lowered precision for ingredients and tools. The model is biased towards labelling sequences as ingredient or tool rather than as product which is reflected in the reasonable recalls for ingredient and tool sequences, as well as the precision for products being higher than the corresponding recall. Ingredients in general perform better than tools simply because they occur in the corpus much more frequently.

The *Bedingung* / specification type shows moderate but surprising success considering it is a sort of wildcard category for any kind of remark that is important for the recipe but cannot be fit into any other category. I suspect - and the data seems to confirm this - sequences like time spans and measurements of quantity, which always follow the same pattern (Greene, 2015), raise the overall performance of the class considerably.

Finally, the relatively high recall for prepositions will stem from the linguistic class of prepositions being a closed class. Even more than action terms, prepositions are almost never longer than one token (1 of 305 instances in the corpus). Additionally, prepositions can hardly be confused with other sequence types. I expect a very high recall for this label as soon as the training data has been upsized and the total number of preposition instances in the training has been increased. The considerably lower precision can likely be explained by the circumstance that not all prepositions are assigned to a sequence label. A good fraction are not considered to be contributing necessary information in the original recipe and are consequently not annotated with any label at all.

5.5 Dependeny Parsing

So far, I have only shown part of the parsing that will generate the proposed graphs from continuous recipe text. Unfortunately, the scope of this thesis does not allow elaborate discussion of the steps necessary to produce graphs from sequence labels. However, I want to at least show that an off-shelf dependency parser can learn to identify the domain specific relations from this thesis with reasonable performance.

I test the *Biaffine Dependency Parser* from AllenNLP.[2] The model is conceptualised for Universal Dependency parsing and consists of a biaffine classifier upon a three-layered BiLSTM with multilingual BERT embeddings. The model is based on based on Dozat and Manning (2016). The input to the model are the sequence label and token columns from the sequence labelling task. However, some edges were randomly omitted to transform the input into dependency trees (147 edges were deleted in the training split, 27 in the validation split and 40 in the evaluation split). The output consists of one head per token as well as the corresponding relation type. Full-length recipe files are used as instances in training and prediction in order to include the many dependencies that cross sentence boundaries. The parser achieves an Unlabeled Attachment Score (UAS) of 80.0 and a Labeled Attachment Score (LAS) of 78.3 on the evaluation set. Note that these metrics are computed on a reduced tree representation of the data where surplus edges from the graph representation are completely ignored at training time and in the evaluation.

If the approach to analyse sequence relations with a similar parser as in this experiment is to be pursued, a method to insert all previously neglected edges will have to be developed.

Training the parser on tokens and POS tags, rather than tokens and the domain-specific labels, yields a UAS of 45.4 and an LAS of 42.2, showing that the previously described sequence tagging task is essential to finding the correct dependency relations between the individual sequences. Comparing these results to training only on the tokens (UAS: 39.4 and LAS: 35.2), a weak correlation between syntactic properties and relations from the cooking domain can be inferred. This is plausible as many cooking relations are variations of verb-adverb or verb-object relations (e.g. the inputs and outputs of a *Kochschritt* very often are the objects of the corresponding

[2] `https://allenai.github.io/allennlp-docs/api/allennlp.models.biaffine_dependency_parser.html`

verb). The results suggest that the cooking dependency model could profit from pre-training on syntactic dependencies of a bigger corpus before fine-tuning it to the domain specific sequence labels and relations.

In their approach to parsing recipes into trees, Jermsurawong and Habash (2015) reach an UAS of 93.5%. However, lower performance than this was to be expected from the above model as the parsing task in this thesis is much more fine-grained. Where their model relates sentences as smallest (and only type of) entities, this model has to distinguish seven different types of sequences in the input. Also, this model is trained on a smaller corpus and is not fine-tuned at all. On the other hand, Jermsurawong and Habash (2015)'s approach is solely rule-based while neural approaches like my own are generally expected to outperform rule-based ones. Note, that only the UAS can be compared as Jermsurawong and Habash (2015) do not have different types of relations.

Chapter 6

Future Work

In this thesis, I have introduced a formalization for recipe texts that makes recipes machine-readable. I have also outlined the first steps to building a parser on recipes. I this regard, I have introduced an annotation scheme and a first approach to sequence tagging with a neural model, with the tagging model showing promising performance. Also, I have marked an outline for dependency parsing on the sequences. Here, I discuss further steps needed to improve the performance of both models. Additionally, I illustrate potential future research projects that expand upon the work presented in this thesis.

6.1 General Ways to Improve the Parsing Results

Both parsing models presented in this thesis, the sequence tagger and the dependency parser, show promising results.

An obvious choice to improve the performance of these parsers is to provide more training data. Neural Networks are known to depend heavily on big amounts of data which we hope to provide in an additional round of annotations, soon. Additional to annotating more data, data can be fed into the models by fine-tuning the token embedder on unannotated domain-specific data. Unannotated data can be acquired quite easily in very big quantities. Furthermore, I expect the models to improve from using POS tags as supplementary input features. They can be added to the annotated data in a pre-processing step.

I have already performed some fine-tuning on the tagging model and seen that the performance can be increased. It stands to reason that more exhaustive fine-tuning can lead to even better performance. For example, learning rate schedulers and momentum schedulers can be investigated. Also, more token embeddings like the fine-grained BERT, for example, could be compared to ELMo and the smaller BERT that I have already explored. Admittedly, with the current results, it seems likely that a more fine-grained BERT model will only be able to increase the overall performance if it has access to enough training data.

An additional approach to improving performance could reduce the set of sequence types and dependency relations. On the one hand, I intend recipe representations to be very detailed to assure compatibility with a wide range of applications. On the other hand, some types and relations are assigned with very narrow constraints and could easily be acquired in rule-based post-processing. For example, prepositional phrases might be changed into *ingredient*, *product* or *tool* sequences, respectively, from the current pattern, *preposition* plus *ingredient / product / tool*. Also, the *zero anaphora* relation could adopt the *input* label as the distinction between both relations exclusively lies in the types of the involved sequences. In this way, the core model might achieve better performance without adding too much to the overall computational effort.

Finally, it might be interesting to compare the neural setup from this thesis with rule-based approaches.

6.2 Deeper Insight into the Tagging Task

The CRF model assumes all tags in the tagset to be independent of each other. We, on the other hand, consider tags with the same sequence label to be more similar to each other than to tags with different sequence labels. This should be reflected in the computation of the loss function. Currently, the loss for training the model is derived from the probabilities produced in the CRF. To weigh sequence labels higher than positions, a new loss function has to be developed.

I observe that the corpus is so small that it doesn't contain all tags that are possible to generate from the set of sequence labels in the *BIOUL* format. Some other tags appear very infrequently. On the one hand, it might be feasible to insert the missing tags into the model. Then, a pre-trained CRF tagger could assign these tags without ever having seen them in the domain-specific training data. In this case, the tags in question are assumed to be occurring so infrequently in the data as a result of random chance and coincidence because the data set is quite small. However, it seems more likely that they will also appear very seldomly in bigger data sets because they simply are very uncommon. In that case, it might be more feasible to simplify the model by discarding the infrequent tags and find a different solution for the few cases where they do appear. Which of the two options is actually better could be assessed better once more annotated data is available.

Looking at this issue from a different angle, the *BIOUL* annotation scheme might not even be the ideal choice to represent the given segmentation. *BIOUL* is the most fine-grained annotation scheme used in sequence labelling. Contrary to other annotation schemes, it can also represent nested sequences. As nested sequence do not occur in the annotated data, different tagging option can be explored without losing precision. Which tagging scheme is the best is influenced by the language, the tag set, the model's architecture and, most likely the domain, as well. So without testing different segment annotations against each other, we cannot form an opinion as to which will be the best for the described setting (Konkol and Konopík, 2015).

6.3 Approaches to Dependency Parsing

In this thesis, I have presented one design for parsing dependency relations. This model has the major shortcoming of being a tree parser whereas the structures in the data are actually graphs that slightly extend trees. Consequently, either the missing dependencies have to be added in post-processing or the model has to be modified in a way that it can model more general graphs than trees.

In the tested dependency model, predictions are computed for whole recipes. This approach bears the advantage of including all long distance dependencies. Parsing recipes sentence by sentence is another promising setup. A sentence parser can be trained more easily than a parser for whole recipe texts simply because of the shorter inputs and outputs. Also, the number of training samples is increased indirectly as each recipe is split up into multiple sentences. On the other hand, this approach cannot model dependencies stretching beyond sentence boundaries which will have to be added in a subsequent processing step. Both, this or the current model could benefit from pre-training a parser for syntactic dependencies before fitting the resulting model to the domain and exchanging the output layer for another one with domain specific relation types.

The performance of the dependency parser is quite good. Still, the performance can be further improved with some rule-based feature engineering. The dependency relations were annotated under rather strict type constraints. As mentioned before, many relations types are redundant in the sense that they can be deduced only from the sequence labels and the underlying constraints. These constraints should be implemented in the output layer of the parser. In fact, it might be interesting to see how well a model based solely on the constraints will perform.

6.4 Further Research Based on Recipe Graphs

The recipe representations from this thesis are very detailed with the objective to simplify them to different extents as the concrete use case demands it. In some cases it can be enough to consider only the general structure of the recipe, e.g. the task described by Chang et al. (2018). For such comparisons, a reduction to a limited set of actions can be imagined. However, actions should not be abstracted in general, as representations are required to be precise in other application examples (Tasse and Smith, 2008). For example, a dialogue assistant should be able to specify the manner in which components of a dish are combined. It makes a big difference whether something is mixed in, folded in or put on top of something. Good abstraction levels for different fields of application will have to be investigated.

As already mentioned in Chapter 3, some applications like dialogue assistants or translation tasks may require recipes to be expanded into linear text. One important aspect will be that the text generated from the structural representation not only has to sound natural, it also has to exhibit the characteristics of typical recipe text as it is described in Chapter 3. Apart from the obvious task of expanding a structural representation into natural sounding text, an important sub-task will be finding a good temporal ordering. I propose beginning with the definition of a basic temporal hierarchy, such that an action stems from all previous actions necessary to perform the action itself (Jermsurawong and Habash, 2015). Also, actions should be annotated with how long before their subsequent action they should be begun. These temporal measures constitute an estimate that can be used as a heuristic in putting independent actions in order. This approach will add a nuance to the recipe representation that makes sure processes that take longer but do not need much of the cook's attention (e.g. pre-heating the oven) are initiated *before* actions like making a cake batter which have to be performed immediately before the succeeding action (in this example, putting the cake into the oven).

So far, I have only discussed processing recipe text. However, the titles and ingredient list were disregarded only for the purpose of the presented research. Both are important components of recipes, nevertheless. When a recipe has been parsed into a graph, the next step is to connect it to its title and ingredient list. The title usually is the name of the dish described in the recipe and therefore it is the output of the very last action. Connecting the graph to the entries in the list of ingredients will be more difficult. Trivial assignments, i.e. when the recipe mentions single ingredients by the exact term that is also used in the list of ingredients, are rather common. But there are also less obvious cases in considerable frequency. As mentioned before (Chapter 3.3), inputs to actions can be referred to by collective entities. Some of these are named products (e.g. *"everything"* or *"the chopped vegetables"*), which may be resolved with dependency parsing. I expect the performance of the model to have lower recall than other sub-tasks as it is likely that not all referenced components will always be recognised.

Collective entities referring to ingredients (e.g. *"the dry ingredients"* or *"the remaining ingredients"*) might not be as easy to resolve. They are practically impossible to resolve only from context inside the recipe text. The list of ingredients offers a limited range of possibilities but for many collections, the actual assignment involves world knowledge exceeding anything mentioned in the recipe itself. Similar issues arise with the problem of resolving hypernyms. I hope to find or construct a suitable ontology that can identify properties and related terms for ingredients and tools. To some extend, features of ingredients may also be learned from aligning and comparing similar recipes. For this to be actually useful, big amounts of recipes will have to be used.

In general, recipe graphs can facilitate various different aspects of machine learning from aligning graph representations of multiple recipes for the same or for similar dishes. With the knowledge gathered from finding parallels in different recipes, individual recipes can be extended to be more detailed. This is relevant to applications, for example when it comes to customising recipe presentation on a user interface according to the users' respective levels of expertise. For example, if a recipe states *"poach the eggs one after the other"*, and the user is not sure what *poaching* is, they will be offered the possibility to expand the action and get the more detailed explanation that poaching an egg means boiling it without the shell. This explanation can be expanded even more to a precise description of all steps entailed in poaching an egg from boiling water to how long the egg should stay in the water. I expect especially good advances in learning which actions and ingredients are entailed in preliminary products like *broth* or *dough*. Other products like *mixture* are rather arbitrary and apply to the output of basically any action including and succeeding an ingredient combination action. One goal of learning from alignments will be to enrich recipe graphs with names for the outputs of all actions and, in reverse, learn the entailments of products. On a side note, an investigation whether products are typically named under certain conditions, will be interesting and can possibly improve the aforementioned task of making textual output at a user interface sound more domain-specific.

Another promising aspect to be learned will be the use of tools. Tools are mentioned only occasionally in typical recipe texts. Completing recipes by adding all necessary tools to their graphs, will make automatic comparison more successful and will also provide the data for functionalities like judging whether a recipe can be cooked with the available equipment and many more (see also Chapter 4.1). Detailed information on the tools to use is also very important in fitting recipes to the needs of an inexperienced user. Such an inexperienced user can also be a robotic application to which the use of tools might not be obvious.

Continued research into the cooking domain may eventually involve experiments as to whether neural networks can learn enough about recipes to generate new recipes themselves. The main evaluation objectives will be completeness, connectedness and taste. With completeness I mean that all necessary steps are specified such that the dish can be cooked solely by following the recipe. For example, if a mixture is mentioned, it has to be the output of some previous action. It's especially important that all ingredients which are needed are actually mentioned. Also, completeness requires one final product that can be considered a dish. Recipes with several final products or with an uncooked dough as final product would not likely be considered complete. The corresponding recipe graph should also be fully connected. In particular, all ingredients, tools and products that are introduced should also be used in a cooking action. While these conditions are sufficient from a theoretical point of view, and while a model satisfying them will be stunning, keeping actual use in mind, *taste* will be the most important objective for evaluating the success of the generated recipes. Even more, a recipe generator will only be useful if it can produce interesting recipes in the sense that they are novel and taste at least very well. The generation task can be relevant for filling gaps in the recipe database. My vision is a model with the following feature: If an exemplary search in the database for *turkey with apricots* does not yield any results, a new recipe will be generated from existing recipes for turkey and the knowledge that apricots are fruit.

Lastly, it will be interesting to see how well recipe graphs and the corresponding parsing methods will generalize to to other languages and domains. As the language style used in recipes is very characteristic and specific (at least in German), other languages might have completely different challenges and features to offer. I hypothesize that, when it comes to portability to slightly different domains like user manuals, recipe graphs can make for a good basis in developing structural representations for the described processes. Similar applications can be imagined in related domains like user manual. There, too, dialogue assistants, script learning or translation of text into robotic scripts seem viable.

Chapter 7

Conclusion

This thesis constitutes an introduction to the recipe domain in NLP with the motivation of finding a detailed machine-readable representation for recipes as well as a methodology for parsing recipes into these representations, extracting and preserving all their relevant content and the precise structure of the cooking process underlying a recipe.

Upon closer inspection of the domain, I have found German recipes to be written in very characteristic language. While most recipes conform to a uniform style, thus facilitating recipe processing, other typical features like ellipses, deviations from temporal ordering and non-explicit references pose challenges. The two key aspects of information encoded in recipes are dependency relations between individual actions on the one hand, and the states that ingredients and tools go through, generating products, on the other hand. With this information, the structure of recipes can best be organised in graphs. In this thesis, I used more detailed graphs, which additionally include modal and prepositional specifications, as well as labelled relations between sequences.

I have annotated a collection 100 German recipes and developed a neural approach to sequence labelling on this corpus. The proposed neural network, consisting of a combined embedder with character embeddings and pre-trained ELMo token embeddings, a two-layered bidirectional LSTM as encoder and a CRF in the output layer, achieved an F1 Measure of 80.2 after moderate fine-tuning. A comparison of ELMo and BERT token embeddings showed poorer performance for BERT than for ELMo, which I trace back mainly to the small size of my corpus. Furthermore, I tested an off-the-shelf architecture for dependency parsers on the relations in recipe graphs and can account for an LAS of 78.3. I also demonstrated that this task depends heavily on previous sequence labelling.

In this thesis, I have demonstrated an approach to parsing recipes into graph structures. Once the parsing task has been completed with necessary improvements like providing more training data or refining the take on dependency parsing, among others, recipe graphs can be the foundation for more research and a broad field of applications in the kitchen domain.

References

Akbik, A., Blythe, D., and Vollgraf, R. (2018). Contextual string embeddings for sequence labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, pages 1638–1649.

Cai, L., Zhou, S., Yan, X., and Yuan, R. (2019). A stacked BiLSTM neural network based on coattention mechanism for question answering. *Computational intelligence and neuroscience*, 2019.

Chang, M., Guillain, L. V., Jung, H., Hare, V. M., Kim, J., and Agrawala, M. (2018). RecipeScape: An interactive tool for analyzing cooking instructions at scale. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 451. ACM.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dozat, T. and Manning, C. D. (2016). Deep biaffine attention for neural dependency parsing. *CoRR*, abs/1611.01734.

Gardner, M., Grus, J., Neumann, M., Tafjord, O., Dasigi, P., Liu, N. F., Peters, M., Schmitz, M., and Zettlemoyer, L. S. (2017). AllenNLP: A deep semantic natural language processing platform.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Greene, E. (2015). Extracting structured data from recipes using conditional random fields. URL: `https://open.blogs.nytimes.com/2015/04/09/extracting-structured-data-from-recipes-using-conditional-random-fields/`. Last visited: September 18, 2019.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Huang, Z., Xu, W., and Yu, K. (2015). Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991.

Jermsurawong, J. and Habash, N. (2015). Predicting the structure of cooking recipes. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 781–786.

Kaufman, C. K. (2006). *Cooking in acient civilizations.* Westport, Conn. : Greenwood Press.

Konkol, M. and Konopík, M. (2015). Segment representations in named entity recognition. In *International Conference on Text, Speech, and Dialogue*, pages 61–70. Springer.

Lafferty, J., McCallum, A., and Pereira, F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. *CoRR*, abs/1603.01360.

Li, M. (2017). CRF Layer on the top of BiLSTM (BiLSTM-CRF). URL: `https://createmomo.github.io/2017/09/12/CRF_Layer_on_the_Top_of_BiLSTM_1/`. Last visited: January 9, 2020.

Ma, X. and Hovy, E. H. (2016). End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. *CoRR*, abs/1603.01354.

Malmaud, J., Wagner, E., Chang, N., and Murphy, K. (2014). Cooking with semantics. In *Proceedings of the ACL 2014 Workshop on Semantic Parsing*, pages 33–38.

Moon, T., Aswathy, P., Ni, J., and Florian, R. (2019). Towards lingua franca named entity recognition with bert. *arXiv preprint arXiv:1912.01389.*

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365.*

Ratinov, L. and Roth, D. (2009). Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, CoNLL '09, page 147–155, USA. Association for Computational Linguistics.

Schäfer, U., Arnold, F., Ostermann, S., and Reifers, S. (2013). Ingredients and recipe for a robust mobile speech-enabled cooking assistant for german. In *Annual Conference on Artificial Intelligence*, pages 212–223. Springer.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Silva, N., Ribeiro, D., and Ferreira, L. (2019). Information extraction from unstructured recipe data. In *Proceedings of the 2019 5th International Conference on Computer and Technology Applications*, pages 165–168. ACM.

Tasse, D. and Smith, N. A. (2008). Sour cream: Toward semantic processing of recipes. *Carnegie Mellon University, Pittsburgh, Tech. Rep. CMU-LTI-08-005.*

Tjong Kim Sang, E. F. and De Meulder, F. (2003). Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Trattner, C. and Elsweiler, D. (2017). Food recommender systems: Important contributions, challenges and future research directions. *CoRR*, abs/1711.02760.

Vosoughi, S., Vijayaraghavan, P., and Roy, D. (2016). Tweet2vec: Learning tweet embeddings using character-level CNN-LSTM encoder-decoder. *CoRR*, abs/1607.07514.

Appendix A

Graph for a Whole Recipe

(10) a. Die Eier trennen,
'Separate the eggs,'

b. das Eiweiß mit Salz steif schlagen,
'beat the egg whites with salt until stiff'

c. und kühl stellen.
'and put in a cool place.'

d. Das Eigelb mit den restlichen Zutaten verrühren,
'Mix the egg yolks with the remaining ingredients,'

e. den Eischnee locker unterheben.
'gently fold in the beaten egg whites.'

f. Den Teig in eine gefettete Springform füllen [...]
'Pour the batter into a greased (springform) baking pan [...]'

g. und evtl. mit Früchten belegen.
'and optionally garnish with fruit.'

h. Bei 170° C ca. 1 Std. backen.
'Bake for roughly 1 h at 170° C.'

i. Die Oberfläche sollte goldgelb sein.
'The crust should be golden yellow.'

j. Abgekühlt mit Puderzucker bestäuben
'Powder chilled [cake] with icing sugar'

k. und in Stücke schneiden.
'and cut into pieces.'

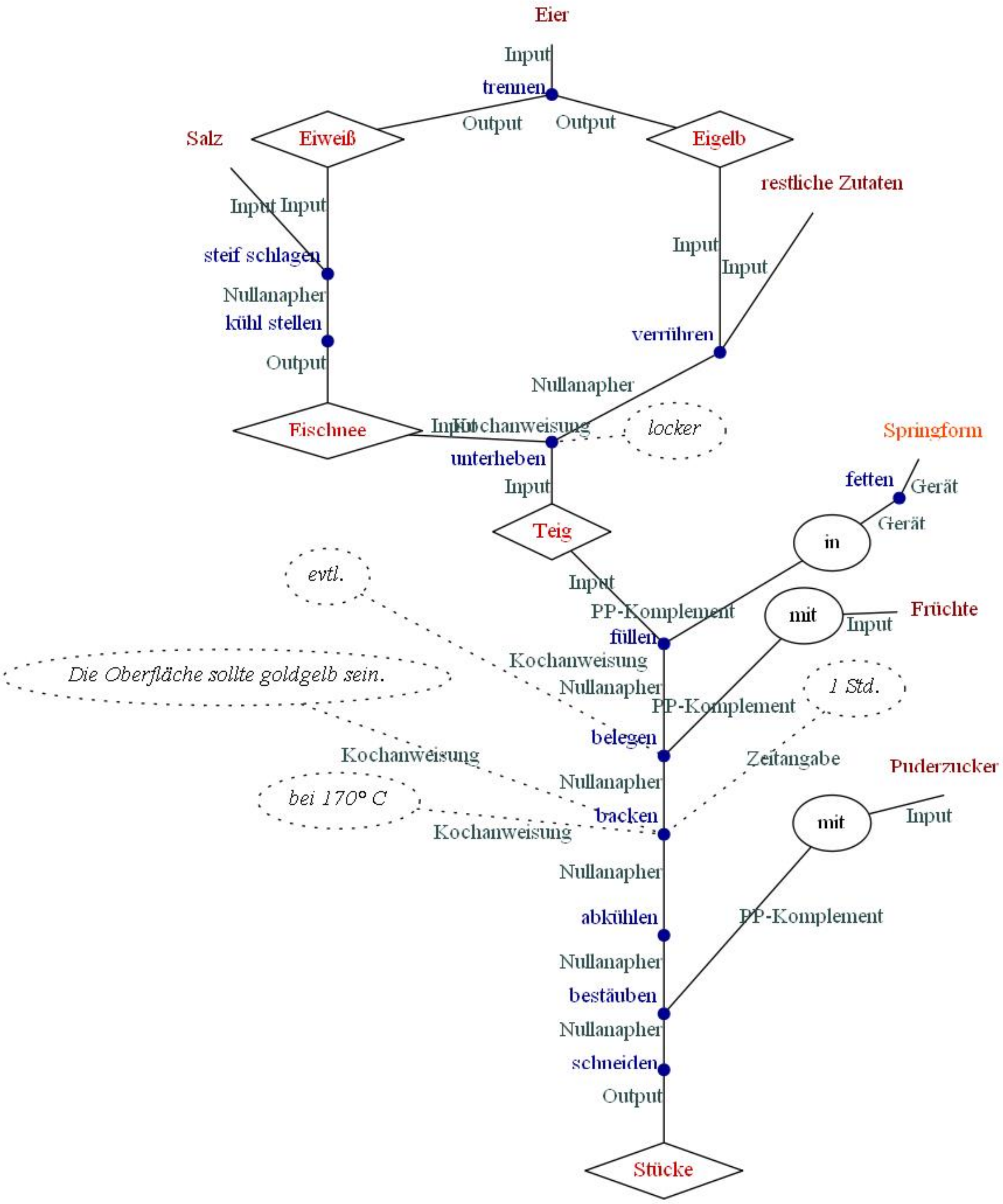


Figure 11. Complete recipe graph for the recipe in (10).