Karlsruhe Institute of Technology

Carnegie Mellon University

A Study on Semantic Parsing of Cooking Recipes

Bachelor's Thesis of

Florian Pfisterer

at the Department of Informatics
Institute for Anthropomatics and Robotics

Reviewer:	Prof. Alexander Waibel
Second reviewer:	Prof. Tamim Asfour
Advisor:	M.Sc. Stefan Constantin
Second advisor:	Prof. Eduard Hovy
Third advisor:	M.Sc. Naoki Otani

July 28, 2019 – November 27, 2019

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Pittsburgh, PA, United States, November 27, 2019

. . . . . .

(Florian Pfisterer)

Abstract

To be useful in interactions with humans, computers need to thoroughly understand natural language. For this purpose, a deep semantic analysis of text is required. Due to the ambiguity and sparseness of natural language however, a syntax-level understanding of sentences by themselves is not enough. Rather, deep semantic parsing requires inter-sentence context and commonsense background knowledge.

In order to come closer to open-domain commonsense knowledge extraction algorithms and thus open-domain semantic parsers, our approach is to focus on a domain with procedural semantics. By a deep semantic analysis of instructional text, we can spot gaps in between instructions, which we can interpret as implicit semantic assumptions and extract as commonsense knowledge. Cooking recipes are widely available, have a limited vocabulary and thus offer an attractive initial starting point for such work.

In this thesis, we study English cooking recipes, identify challenges for semantically parsing them, and evaluate existing rule-based, syntax-based and neural end-to-end semantic parsers on a small dataset of 50 English recipes we create. We compare their performance on trigger word identification (which words evoke actions or entities), action classification (which type of action is evoked) and input identification (which entities participate in each action).

We find that the syntactic parse of the recipe is a very useful input for a semantic parser, as syntax-based pipelines in our comparison outperform neural methods by 0.48 and more absolute F1-score difference in input identification. Additionally, using the Gold syntactic labels instead of a syntactic parse obtained from a state-of-the-art dependency parser improves the input identification F1-score by 0.06. This also suggests current syntactic parsers have a relatively low accuracy in the cooking recipe domain, which we examine both quantitatively and qualitatively.

Based on our investigation, we suggest various directions for future work. We believe that research on inter-sentence context, additional inputs for the parser and the combination of multiple models will bring the best performance improvements.

Zusammenfassung

Um in Interaktionen mit Menschen nützlich zu sein, müssen Computer ein tiefes Verständnis natürlicher Sprache haben. Hierzu ist eine tiefgreifende semantische Analyse von Texten erforderlich. Da natürliche Sprache allerdings oft mehrdeutig ist, reicht ein oberflächliches Verständnis der Syntax einzelner Sätze nicht aus. Vielmehr erfordert tiefgreifendes semantisches Parsen satzübergreifenden Kontext und Commonsense Hintergrundwissen.

Um dem Ziel von allgemein einsetzbaren Commonsense Extraktionsalgorithmen und damit allgemeinen semantischen Parsern näher zu kommen, ist es unser Ansatz, uns auf eine Domäne mit prozeduraler Semantik zu fokussieren. Mithilfe einer tiefgreifenden semantischen Analyse von instruktionalen Texten können wir Lücken zwischen Instruktionen feststellen, diese als implizite semantische Annahmen interpretieren und als Commonsense Wissen extrahieren. Kochrezepte sind breit verfügbar, nutzen ein begrenztes Vokabular und bieten damit einen attraktiven Startpunkt für solch eine Forschungsarbeit.

In dieser Abschlussarbeit studieren wir englische Kochrezepte, erkennen Herausforderungen, diese semantisch zu parsen und evaluieren bestehende regelbasierte, syntaxbasierte und neuronale End-zu-End semantische Parser auf einem Datensatz von 50 englischen Rezepten, den wir selbst kreieren. Wir vergleichen deren Genauigkeit bezüglich Auslöser-Identifizierung (welche Wörter implizieren Aktionen oder Entitäten), Aktions-Klassizifierung (welcher Typ von Aktion wird impliziert) und Input-Identifizierung (welcher Aktion sind Entitäten zugeordnet).

Wir stellen fest, dass der syntaktische Parse eines Rezepts eine sehr nützliche Eingabe für einen semantischen Parser ist, denn syntaxbasierte Pipelines in unserem Vergleich übertreffen neuronale Methoden um 0.48 und mehr in absoluter F1 Differenz. Zudem verbessert die Eingabe der syntaktischen Gold-Labels anstatt des von einem state-of-the-art syntaktischen Parser generierten Parses den F1-Score bezüglich Input-Identifizierung um 0.06. Dies suggeriert ebenfalls, dass aktuelle syntaktische Parser eine vergleichsweise geringe Genauigkeit bei Kochrezepten haben, was wir sowohl quantitativ als auch qualitativ untersuchen.

Basierend auf unserer Untersuchung schlagen wir diverse Richtungen für zukünftige Forschungsarbeiten vor. Wir denken, dass Forschung in den Bereichen satzübergreifender Kontext, zusätzliche Eingaben für den Parser sowie die Kombination mehrerer Modelle zu den besten Leistungssteigerungen führen wird.

Contents

List of Figures

List of Tables

1. Introduction

1.1. Motivation

Language is not only how we communicate but also makes up for a lot of how we think. Thus, for computers to be useful in interactions with humans, they need to deeply understand natural language.

Understanding natural language in the way a human does requires the computer to have much more than a surface understanding like the syntactic structure of each sentence can provide. Deep semantic analysis of texts requires inter-sentence context and often knowledge about not explicitly mentioned facts. Such commonsense knowledge is shared by humans and used to interpret texts, without them being consciously aware of.

Our goal is to get closer to developing an open-domain deep semantic parsing framework. Our approach to acquire the commonsense knowledge required to do so is to detect implicit semantic assumptions in texts by performing a semantic analysis of them. Texts with procedural semantics that list instructions are well-suited for this task, because by semantically parsing them, we can spot gaps and extrapolate to omitted pieces of information, which we can interpret as commonsense knowledge.

We choose to use cooking recipes as one representative of this category of text. Because cooking is such a common task for humans, in cooking recipes, "obvious" information is frequently left out. We investigate methods to build a semantic parser for cooking recipes that can be used to infer such gaps. Such a parser needs to be able to accomodate the specific characteristics of cooking recipes which we study as part of this work.

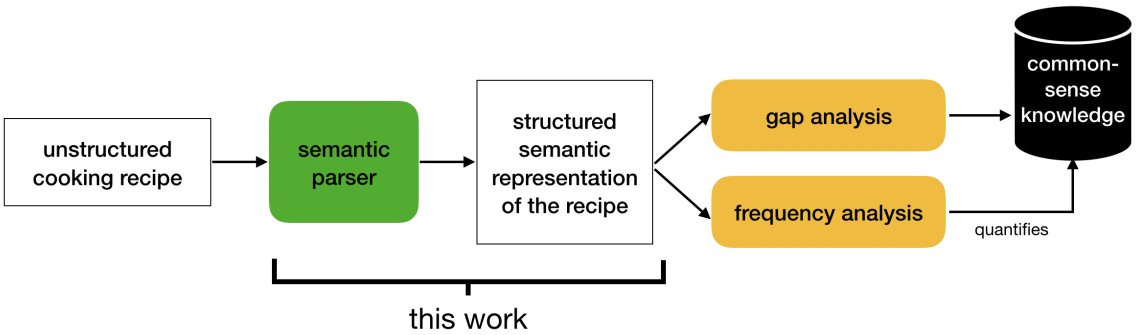


Figure 1.1.: High-level overview of the commonsense knowledge extraction pipeline and the scope of this work.

Figure 1.1 gives a visual overview of the overall framework and which parts this thesis is focusing on. Components of the pipeline which are not part of this work will be briefly explained to aid coherence and comprehensibility where appropriate.

1.1.1. Semantic Parsing

What does it actually mean for a computer to *understand* a piece of natural language?

What does it actually mean for a *human* to understand a piece of natural language?

One definition of understanding a text is that the computer can correctly answer questions about it. What does the computer actually "acquire" by "reading" a piece of natural language that allows it to reason and answer questions about the text?

In this thesis we take the approach that at the core of automatic natural language understanding with computers lies a *representation* that allows the computer to structure the unstructed natural language text it has read in order to reason about texts or even unmentioned facts. Thus, fundamentally, natural language understanding becomes the transformation of natural language into such a representation.

This is the definition of semantic parsing, which has various applications [21]. One is to understand and execute commands for robots or conversational agents like Siri, Google Assistant or Alexa. Another is data exploration by parsing natural language queries to formal database queries.

In this work we study study semantic parsing with the downstream task of commonsense knowledge acquisition in mind. Acquiring large amounts of commonsense knowledge — knowledge that humans unconsciously share, often without being aware of it — will help computers understand humans and the world they live in better and thus make them more useful.

1.1.2. Cooking Recipes

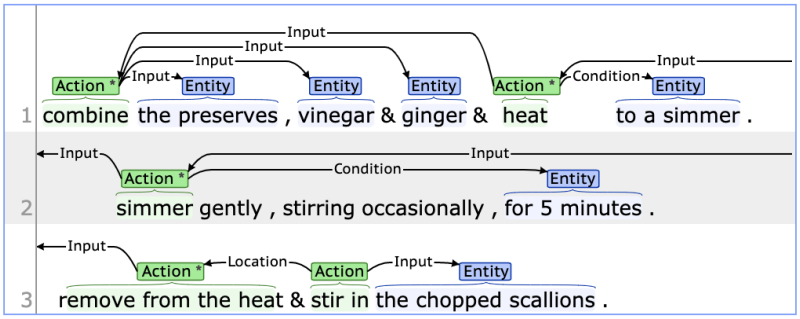


Figure 1.2.: An example cooking recipe, semantically annotated in brat [42].

Why study cooking recipes and not another category of text? Semantic parsing of open-domain texts is a hard problem and is far from solved [21]. It includes not only building a parsing algorithm, but also designing a semantic representation that captures enough depth for the specific use case. Figure 1.2 show an example recipe that is semantically annotated. The goal is that a semantic parser can produce such and deeper annotations (i.e. a structured representation) not only for cooking recipes, but for open-domain texts. In order to come closer to such a general-purpose semantic parsing framework, we focus on a narrower domain. Cooking recipes are attractive for several reasons.

Firstly, cooking recipes generally use a limited vocabulary. This reduces sparsity, a problem commonly encountered in natural language processing [5]. It also allows the

development of rule-based systems, because the rules can be manually designed if the scope is limited. ocra was first built solely for omelet recipes, allowing us to only focus on language commonly encountered in describing how to make an omelet.

Secondly, a lot of (unlabeled) data is available, which is especially important for machine learning approaches. There are several publicly available corpora of cooking recipes [22, 7, 43].

Thirdly, in contrast to other types of procedural texts such as lab protocols [25] or scientific processes [35], to understand cooking recipes, one requires little to no domain knowledge. Thus, annotators of a corpus do not need to have a professional background in the respective area, which makes the crowd-sourced annotation of training samples cheaper. Annotated training data — as was confirmed during this study — is very important not only for machine learning models to learn patterns in the data but also for researchers to better understand the problem itself.

Fourthly, semantically parsing cooking recipes allows interesting applications beyond the downstream task of commonsense knowledge acquisition, as in our project. One could imagine personalized instructions for some dish based on the user's cooking expertise, combining multiple recipes on the same dish based on the user's preference or the generation of new recipes [7].

Finally, as this study demonstrates, parsing cooking recipes comes with many complex, unsolved challenges.

1.2. Contributions of this Work

The main contributions of this thesis are fourfold:

1. We design a semantic annotation schema tailored both for the downstream task of commonsense knowledge acquisition and the representation challenges arising in cooking recipes.

2. We annotate a dataset of 50 English cooking recipes both syntactically and semantically with this annotation schema. Based on the annotation effort and statistics gathered from the dataset, we study the syntactic characteristics commonly encountered in this domain. We identify challenges that arise from these characteristics and draw conclusions as to what is important in a semantic parser for cooking recipes.

3. We evaluate four different pipelines on this dataset, both quantitatively and qualitatively. We study the accuracy measured in precision, recall and F1-score of these pipelines on three core metrics we define, and also analyze the parsers' performance with respect to the cooking recipe challenges identified earlier.

4. Based on the results of this evaluation and the accompanying case study, we propose several directions for future work.

1.3. Structure

Chapter 2 briefly reviews some background concepts about natural language processing, syntax and semantic parsing built on later in the thesis. In chapter 4, we describe our annotation effort for the 50-recipes dataset. We further study characteristics and challenges found in cooking recipes, evaluate different semantic parsers on the dataset and analyze them with respect to the challenges identified earlier. Chapter 5 places this thesis within related work in the two fields it touches: the category of text (procedural text) and the task (semantic parsing). Based on our investigation, chapter 6 proposes several directions for future work. It draws both on work in this thesis as well as on related work, and suggests ways to combine different methods which might yield considerable performance improvements. Finally, chapter 7 concludes the thesis, reviewing the core findings and contributions.

2. Background

2.1. Natural Language Processing

Natural language processing (NLP) is a field at the intersection of computer science and linguistics concerned with how computers process, understand and generate natural language [30]. Basic concepts from NLP needed for the following sections will be briefly reviewed in this section.

2.1.1. Syntax

Syntax is concerned with the structure of a sentence.

Parts of speech Parts of speech (POS) are categories of words with similar grammatical and semantic properties. The most common are noun ("carrot"), verb ("stir"), adjective ("liquid"), adverb ("often"), preposition ("in"), coordinating conjunction ("and"), determiner ("the") and pronoun ("it") [30]. A POS-tagger reads a sentence and assigns a POS-tag (noun, verb, etc.) to each word.

Syntactic parsing There are two paradigms for syntactic parsing: constituency parsing and dependency parsing. Constituency parsing utilizes a context-free grammar to separate a sentence into a tree of its syntactic constituents (such as noun phrase, verb phrase, etc.). Dependency parsing on the other hand builds a tree where each word except the root has exactly one *head* word (parent node), which is the semantic center of the relation and provides features for surface forms such as inflections. The child node of a head word is called the *dependent* of the head word [30]. The relation between head and dependent is called a *dependency*.

Dependency labels Dependency labels describe the relation between a head and a dependent. As an example, a verb can have an object and a subject (and other relations to words in the sentence). In this case, the verb would be the head and the object and subject would be dependents of the verb. Different languages have different sets of dependencies. Work has been done on establishing a universal standard for dependency relations. Universal Dependencies [38] is the product of one of such projects, and we utilize its schema in our work.

2.2. Semantic Parsing

This section formally defines semantic parsing and what it entails, relates it to sub-problems and discusses different approaches. Finally, it introduces the type of semantic parsing we examine in this thesis.

2.2.1. Task Description

Semantic parsing is the transformation of a piece of natural language into a formal, machine-understandable representation of its meaning. Depending on the depth of this representation, semantic parsing entails a number of sub-problems which are explained in the following.

2.2.1.1. Intent Classification

Each piece of text, usually a sentence, has a primary intent or event that is being described. The task of intent classification is a special type of sequence classification. Given a text sequence, the task is to determine the main intent expressed in the text out of a predefined set of possible intents.

For example, the sentence "Book a flight from San Diego to Düsseldorf on December 5" primarily describes a "booking" of something. Mostly, when the piece of text is a single sentence, the intent is triggered by a verb. In the following, we will often write a set of tokens (words) *evokes* a frame. The frame (cf. 2.2.2.2) can be understood as a template for the meaning of the sentence, given by the primary intent, and which has slots for arguments and links to other frames to be filled in.

2.2.1.2. Semantic Role Labeling

Also called shallow semantic parsing, semantic role labeling is the task of identifying entities in a sentence and assigning semantic roles to these entities [20]. An entity might be a person, an object or an abstract concept - any group of words that semantically describes one concept. A semantic role is interpreted with respect to the primary intent of the sentence. The slots of the frame are filled in by the entities, which have semantic roles defined with respect to the frame. The essence of semantic role labeling is, given a sentence, to answer the question "who did what to whom".

Formally, given a sequence of n tokens $X =< x_0, \ldots, x_{n-1} >$, the task of semantic role labeling is to predict a sequence of labels of the same length: $Y =< y_0, \ldots, y_{n-1} >$ [20]. Each label y_i determines if token x_i belongs to an entity in X, and if so, gives the role of this entity. A common way to handle multi-token entities is the IOB format [40], in which the label can be one of

- **B**-`role`: beginning of an entity of role `role`
- **I**-`role`: inside of an entity token sequence of role `role`.
- **O**: outside, not part of an entity

For example, the cooking recipe sentence "Mix in herbs, salt and pepper." (input sequence $X =<$ 'Mix', 'in', 'herbs', ',', 'salt', 'and', 'pepper', '.' $>$) may be labeled with the label sequence $Y =<$ O, O, O, O, B-Input, O, B-Input, O $>$. Note that the first two tokens ("Mix in") are the trigger for the primary intent of the sentence and might evoke a "Merge" frame.

2.2.1.3. Coreference Resolution

Semantic Role Labeling is commonly framed as a per-sentence problem, because semantic roles are defined with respect to the main intent of a sentence [20]. However, when one interprets a document (a sequence of sentences) as a whole, one often encounters different mentions referring to the same concept (e.g., using pronouns). One such domain is the cooking recipe domain which we explore in this thesis. In cooking recipes, which are sequences of instruction sentences, frequently the input to an action described in sentence n comes from sentence $n-1$. Sometimes this relation is made explicit using a pronoun. However, cooking recipes often have anaphora as well: the relation is implicit and needs to be inferred from the semantics (cf. Figure 4.3).

In such cases, an explicit or implicit coreference resolution is needed. A semantic parser must be able to correctly handle cases where one entity is referred to in multiple places, often with different words. Formally, coreference resolution is the task of finding all references to the same entity in a text. It can be framed as a binary classification task to determine whether two references from a text corefer to the same entity or not [20].

2.2.2. The Problem of Representation Design

A core question in semantic parsing is about the representation into which the piece of natural language is transformed. There exist different types of semantic representations for different use cases. The most common are examined in the following [21].

Once one has decided which class of representation is best suited for the task, one must also consider the fine-grained design of the representation. The possible design choices depend on the class of representation and are described in our case in section 4.1.2.

2.2.2.1. Logic-based Formalisms

One meaning representation is to use first-order logic (FOL) with quantified variables and predicates. An example is the sentence "All primes greater than 2 are odd", which can be expressed in first order logic as $\forall x.\text{prime}(x) \wedge \text{greater}(x, 2) \Rightarrow \text{odd}(x)$ [27]. While vanilla first order logic has limitations in what it can represent, FOL augmented with lambda calculus (LC) has been used for querying databases and giving robots instructions [21]. Recent work has introduced a more compact representation of LC [26], which has been applied to answering compositional questions on semi-structured Wikipedia tables [24] among other work [21].

Another type of logic-based formalism for representing semantics is description logic languages, which are a bit less expressive than FOL. A famous representative of this line of work is the Web Ontology Language (OWL) [33], used as a language for ontologies.

2.2.2.2. Graph-based Formalisms

Another approach to representing meaning is graphs. Nodes represent entities or events and edges represent semantic relations between those entities or events [21]. One common way to understand such a graph is to represent each piece of text that evokes some intent as a frame. The basic idea of frame semantics, originally introduced in [13], is that one cannot understand the meaning of a single word without the other knowledge which relates to the word. Thus, one or more words evoke a predefined frame, which relates to the concept the words refer to. A frame consists of slots which link the core intent or event to its arguments. These arguments can be a set of tokens or other frames.

A set of frames is thus a graph, with frames and their arguments being the nodes and frame-valued slots being the (labeled) edges.

As an example, one cannot understand the word "booking" without knowing about what is booked, who is booking something, the date of the booking, etc. Thus, "booking" might evoke a `Booking` frame which has arguments for the the person who books something, the thing that is booked (e.g., a flight) and when the booking takes place (e.g., December 5).

Slots of frames can be generic and the same for all frames (generalized semantic roles, used in PropBank [39]) or they can be specific for each frame (deep roles, used in FrameNet [3]). Generalized roles include proto-agent (the subject, the volitional causer of an event) and proto-patient (the object, the thing being acted on or caused) [20]. In the booking example above, deep roles could be "booker", the "booked thing", "booking date", etc.

In this work, we use frame semantics as the target meaning representation, as it is best suited for our downstream task of commonsense knowledge acquisition (which is further described in section 3.1).

2.2.2.3. Programming Languages

One can also represent the meaning of a piece of text using a high-level programming language such as Python or Java. Advantages include that developers are already familiar with such a representation and the schema and syntax are relatively simple, which limits the scope [21]. Parsing then becomes a kind of machine translation from a natural language like English to a programming language like Python.

2.2.3. Semantic Parsing Corpora

An integral part of semantic parsing research is the development of corpora. The two most relevant corpora for semantic role labeling are PropBank [39] and FrameNet [3], which are further described in the following.

2.2.3.1. PropBank

The Proposition Bank (PropBank) [39] contains sentences from the Penn TreeBank [32] annotated with semantic roles. Each verb sense has specific roles, which are given by `Arg0`, `Arg1`, `Arg2`, and so on. The commonality between different verbs is that, in general, `Arg0` is the proto-agent and `Arg1` is the proto-patient. There is one set of roles for each sense of each verb.

As an example [20], the **agree.01** PropBank entry specifies its roles as follows:

- `Arg0`: the agreeer (the first entity agreeing)
- `Arg1`: the proposition (what is being agreed upon)
- `Arg2`: the other entity agreeing

The **.01** behind the verb (**agree**) in the entry name indicates that this is the entry for the first sense of the verb "agree". It might have other senses, whose entries would then be called **agree.02**, **agree.03**, and so on. An example sentence annotated using this entry is [$_{\texttt{Arg0}}$Management] agreed with [$_{\texttt{Arg2}}$the IT department] about [$_{\texttt{Arg1}}$the new project management system].

In addition to these core arguments `Arg0`, `Arg1`, etc., PropBank also has a number of additional attributes called `ArgMs`, which modify or add to the main frame. Examples include `ArgM-TMP` for temporal information (e.g., "tomorrow morning") and `ArgM-LOC` for information about the location of the event being described (e.g., "at CMU") [20].

2.2.3.2. FrameNet

FrameNet [3] is a dataset for semantic role labeling that contains frames and specific roles for each frame, which provide background information for a given concept, such as booking a flight. In addition to the frame structure, FrameNet also includes labeled example sentences. As an example, the **change_position_on_a_scale** frame is defined as follows [20]: This frame consists of words that indicate the change of an Items position on a scale (the `Attribute`) from a starting point (`Initial_value`) to an end point (`Final_value`). An example sentence annotated using the slots of this frame is [$_{\texttt{Attribute}}$Apple's shares] rose to [$_{\texttt{Final_value}}$USD 245].

2.2.4. The Problem of Transformation Algorithms

Once the desired semantic representation is defined, the next task building a semantic parser entails is to create an algorithm that transform the piece of natural language into this representation. Just like there is a wide variety of representation types, many different parsing system types exist. The most common are described in the following.

2.2.4.1. Rule-based Systems

The first semantic parsing systems were mainly rule-based. Because rules cannot be manually created for all domains, such systems were domain specific [21]. Different methologies on which the system is based on exist:

- pattern matching: predefined patterns are matched against the input text word by word, like in the dialogue system ELIZA [46]
- syntax-based systems: an intermediate syntactical representation given by a syntactical parser is mapped to the semantic representation using manually created rules (ocra, described in section 4.3.1 is based on this idea)

An example from this class of semantic parsers is [15], which introduces a semantic interpreter called "Absity". Semantic interpretation is defined here as "The process of mapping a syntactically analyzed text of natural language to a representation of its meaning." [15].

2.2.4.2. Statistical Techniques

Work on large semantic role labeling corpora like FrameNet [3] has fostered research in statistical techniques for automatic semantic role labeling. In contrast to purely rule-based approaches, these techniques rank the probability of different parses using diverse statistical features. Gildea et al. [14] propose the first statistical model on FrameNet, which utilizes mutliple statistical classifiers to identify both abstract and domain-specific semantic roles in a sentence. Various other statistical and lexical features are combined to train the statistical classifiers. Another line of work has used semantic grammars (e.g., weighted linear combinatory categorical grammars) for parsing PropBank [2], which ranks different parses given by the grammar by their probability.

2.2.4.3. Neural Methods

In recent years, due to the rise of deep learning techniques, many NLP tasks such as machine translation, question answering or syntactic parsing have been approached using end-to-end methods for sequence-to-sequence learning [21]. While traditional methods require hand-crafting features, templates or lexicons, they can better leverage the a-priori logic compositionality than end-to-end methods [21].

One of the first such sequence-to-sequence approaches to semantic parsing uses an encoder-decoder architecture with recurrent neural networks to transduce the input sequence of natural language to the output semantic representation [12]. The decoder is aided in generating the hierarchical structure of the resulting logical expression by using special parenthese-tokens and parent feeding connections.

Recent work on neural semantic parsers includes SLING [41] and Neural Process Networks (NPN) [7], which are examined thoroughly in section 4.3.2 and section 5.1.1, respectively.

2.2.4.4. Syntax-first vs. Semantics-only

While early methods like rule-based systems that use a mapping from a syntactic parse use an intermediate symbolic representation (such as ocra, explained in section 4.3.1), modern neural methods are trained end-to-end, and all intermediate representations are latent. A problem arising with an intermediate symbolic representation is error accumulation. A semantic parser which uses a syntactic parse as the base representation can only be as good as the syntactic parser. In addition, syntax is inherently limited to the sentence level. As this study shows, operating on the sentence level is not enough for semantic parsing of a cooking recipe (cf. section and Figure 4.3). However, modern neural semantic parsers which go directly from text to semantic representation lack the explainability a human-understandable intermediate symbolic representation provides. Latent representations in neural networks can currently not be analyzed as well as a syntactic parse, for example.

3. Commonsense Knowledge

3.1. Project Context

Our long-term goal is to be able to define commonsense knowledge quantitatively and to automatically acquire it from different types of text. Commonsense knowledge acquisition is a longstanding task in artificial intelligence research, but it is not considered a solved problem [48]. For the purpose of this thesis, we use an abbreviated version of the definition of commonsense knowledge given in [48]:

> Knowledge of default assumptions about the world, which seems so fundamental and obvious that it usually does not explicitly appear in people's communications

Our approach is to focus on the domain of instructional text, as this genre often omits "obvious" parts, which we can find by a deep semantic analysis of the text. Initially, our work is concerned with semantically parsing cooking recipes to be able to spot implicit semantic assumptions (in what we call *gap analysis*) and align multiple recipes to quantify the commonsense-ness of a piece of knowledge (in what we call *frequency analysis*). Both of these methologies will be explained in the following.

3.2. Commonsense Knowledge Extraction

Disclaimer: the author of this thesis has not worked on the commonsense knowledge extraction part of this project. His sole focus was on the semantic parsing of cooking recipes with the downstream goal of gap analysis and frequency analysis in mind. To better understand the considerations employed in the semantic parsing part however, this section briefly outlines our approach to commonsense knowledge extraction once the semantic parse of a recipe is available.

3.2.1. Gap Analysis

Our assumption, which we have empirically validated (cf. section 4.2), is that cooking recipes often leave out information that is obvious to a human reader, which however we can automatically discover by a deep semantic parse and literal interpretation of the recipe. The way we discover such left-out information, which we call a *gap*, is by semantically parsing each instruction of a recipe and inferring pre- and post-conditions for each of the participating entities (i.e., ingredients and utensils). We can then compare the post-conditions of an entity from the last instruction it participated in (say in sentence n) with

the pre-conditions of another instruction in sentence $m > n$. If the conditions do not match, we can infer that some information has been left out and that this information is probably commonsense knowledge, as it would have been explicitly given in the recipe otherwise.

As an example, consider this excerpt from a recipe below:

1. Put a skillet over high heat.
2. Melt butter.

One pre-condition of the "melt"-action in the second sentence is that the input (the butter, in this case) needs to be solid and in a location that is hot. We know that the butter is solid by default. However, there is no post-condition of a previous instruction in the recipe that specifies the location of the butter. Clearly, the information where the butter is located has been left out in the recipe. We call this a gap. When a semantic parser detects this inconsistency between post- and pre-conditions, it can try to fill in the gap by looking at adjacent instructions. In this example, it might infer that the hot location the butter must be located in is the skillet from the previous sentence (just like a human does automatically).

Repeating this process for a large number of recipes (including recipes of the same dish) will allow us to build a knowledge base that contains such pieces of commonsense knowledge like the fact that butter is usually heated in a skillet.

3.2.2. Frequency Analysis

We want to be able to quantify the commonsense-ness of a piece of knowledge (in a cooking recipe). Our strategy to do so is to semantically parse a large number of recipes for the same dish (e.g., an omelet), semantically align the instructions of each recipe and then analyze which pieces of knowledge occur in almost every recipe and which do not. We call this last step *frequency analysis*.

For example, we can assume that all omelet recipes will in some form instruct the cook to heat eggs. Thus, this piece of knowledge can be considered not being commonsense (otherwise it would be an obvious step and at least some recipes would leave it out). The information that the eggs need to be cracked before heating them on the other hand might occur only in some omelet recipes. Others might just contain the instruction "Heat eggs in a pan until set" and assume the reader understands that the eggs shells must be removed beforehand. Thus, we can consider this information as being commonsense.

By aligning a large number of recipes for the same dish that way, we can build a *master recipe* that allows us to analyze the frequency of each individual instruction or action-ingredient pair. It also allows us to infer necessary pre- and post-conditions between steps in the master recipe, which we can use to build a lexicon that in turn improves parsing accuracy and hence helps with the gap analysis (described above).

4. A Case Study: Semantic Parsing of Cooking Recipes

4.1. Semantic Representation

Before one can build a semantic parser or annotate a dataset, the first step is to design an annotation schema that captures enough semantic depth for the task. In our case, we need to be able to understand which cooking instructions take place in which order, and which inputs they have, such that we can later assign pre- and post-conditions to them, extract commonsense knowledge and do frequency analysis (as described in section 3.2).

4.1.1. Semantic Characteristics in Cooking Recipes

Cooking recipes offer several challenges for a semantic representation. In particular, they often have complex control structure. Some of the challenges in the following are addressed by our annotation, others are left to future work (see section 6.2).

4.1.1.1. Representing conjunctions among actions or ingredients

In some recipes, multiple actions are included in one sentence and have an "and"- or an "or"-relation between them. If it's an "and"-relation, there most likely is a temporal relation between the actions, which is further discussed in the next section. "Or"-relations between actions (i.e., either do action *A* or action *B*, but not both) need to be representable as well. One option would be to create a new "or"-frame and add the action frames as slots. Another is to establish an "or"-slot on existing frames and link action frames that are conjoined in an "or"-relation. An example sentence from a recipe which requires such an or-relation between ingredients is the following: "Serve topped with whipped topping or ice cream".

4.1.1.2. Representing temporal relations between actions

Cooking recipe instructions naturally have an order (by how they are listed in the recipe). However, we also encountered recipes with more complex temporal relations than this "before"-relation. For example, a recipe step might explain that some action *A* needs to be done every 5 minutes during the execution of action *B*. Such and other temporal relations need to be representable, which is left to future work. The way we treat such complex temporal relations in this work is to conjoin the actions with a "before"-relation and include the specific details in the condition slot of one of the action frames.

4.1.1.3. Representing goal states and relations

Often, the temporal scope of actions is specified explicitly with a given duration (e.g., cook for 30 minutes). A more complex case also occurs frequently, however. Some actions need to be executed until some goal condition is met (e.g., "heat butter until bubbling subsides"). One approach is to represent the goal condition ("bubbling subsides") using another frame, and to link it to the original action ("heat butter") using a new "goal-state" slot, which is the approach we take in this work.

A similar analysis of control structure in cooking recipes has been done in [29], which classifies the relations between actions into condition ("do A until B is satisfied"), sequence "do A and then B") or alternatives ("do either A or B").

4.1.2. Annotation Schema

We present the annotation schema we have designed based on the requirements for the downstream task of commonsense knowledge acquisition (section 3.2) and the unique challenges encountered in cooking recipes (section 4.1.1). We later use this schema to annotate our own small dataset of cooking recipes, which is described in section 4.2.

In general, we represent a recipe as a whole (not each single sentence by itself) and annotate on the word level. Each contiguous span of words can be an `Action` (and evoke one or more frames) or an `Entity` or neither. In addition to this word-level annotation, actions and entities can have binary relations between them. The frame types and relation types are described in the following.

4.1.2.1. Frame types

Frames describe actions that the cook in a recipe must perform. The current set of frames has been iteratively devised during the annotation of our small dataset. We started with a small set of given cooking actions inspired by [7] and supplemented them with new frames as we encountered a case that was not covered by the current set of frames. Finally, we generalized some frames such that they cover multiple more specific frames in order to simplify the annotation task.

Some of the most important frames are `Merge` (multiple inputs are merged together, e.g., "Mix eggs, salt and pepper."), `LocationChange` (something is moved to a different place, e.g., "Slide the omelet onto a plate.") and `TemperatureChange` (the input is either cooled or heated, e.g., "Heat butter."). The full list of frames, along with a definition of each, can be found in Appendix A.

4.1.2.2. Relation types

Each action frame can have the following relations to other frames and entities:

- `Location`: an entity that describes the location where this action takes place or where something is moved
- `Utensil`: a list of entities that describe utensils like a fork, spoon or oven which are utilized in this action

number of recipes	50
number of sentences	404
number of verbs	747
number of unique verbs*	251

avg number of sentences per recipe	8.08
avg number of verbs per sentence	1.85

Table 4.1.: General dataset statistics of the 50 recipes dataset collected and annotated by us as part of this study. *: after lemmatization

- `Input`: a list of entities that describe the entities being acted upon in this action or `Actions` if the output of the other action is an input of this action
- `Condition`: a list of entities that describe conditions of the action, such as temporal conditions (e.g., the duration of the action)

Entities cannot have relations to other entities, only to action frames. An entity can be linked to at most one action and an action can be an `Input` of at most one other action.

4.1.2.3. Pre- and Post-conditions

Given a set of frames with the corresponding input entities, one can assign pre- and post-conditions to the inputs, which later allows finding gaps for gap analysis. Such conditions are usually a set of attributes for ingredients, such as their location, temperature or state of matter. In ocra, pre- and post-conditions are based on a manually created lexicon.

As this thesis is only concerned with the semantic parsing part and the author has not worked on the pre- and post-conditions, we omit further discussion of how to represent and infer such conditions.

4.2. 50 Cooking Recipes Dataset

To deeply investigate the linguistic characteristics of cooking recipes, we manually annotated a corpus consisting of 50 English recipes. These recipes were randomly sampled from the Now You're Cooking (NYC) Dataset [22] and describe different dishes and drinks. The NYC dataset contains recipes collected from mailing lists in the 1990s. As the authors of the recipes are not professional recipe writers, the language is rather informal. Some general statistics about our 50 recipe dataset can be found in Table 4.1.

The annotation was performed by two annotators, both of whom do have a background in NLP, but are non-linguists. The annotation was split up into two phases which built on each other: syntactic and semantic annotation. For both phases, we used brat [42], a web-based visual annotation tool.

4.2.1. Syntactic Annotation

In a first phase, we annotated only the syntactic dependencies and part-of-speech (POS) tags. Specifically, we focused on dependency relations that are relevant to the later semantic annotation. The relevant relations are listed in the following with their Universal

Category	Precision	Recall	F1
verb detection	0.99	0.88	0.93
obj identification	0.92	0.83	0.87
obl identification	0.76	0.90	0.66

Table 4.2.: Performance of the StanfordNLP dependency parser on selected POS-tagging and dependency categories on our 50 recipe dataset.

Dependencies [38] abbreviations, along with the semantic features they typically capture in a recipe sentence.

- **root**: usually the first verb in the sentence, which describes the action (type of instruction) the sentence describes, e.g., "blend in flour."
- **direct object** (*obj*): usually an ingredient or other input to an action, e.g., "blend in flour."
- **oblique nominal** (*obl*): typically a location or condition of the action, e.g., "put flour and cocoa in a baking pan."
- **adverbial clause modifier** (*advcl*): typically a goal condition of an action, e.g., "mix well until smooth."
- **conjunct** (*conj*): usually encountered in enumerations of ingredients or actions (only considered if the root of the conjunction is an object or oblique nominal of the verb), e.g., "parboil potatoes and carrots."
- **compound** (*compound*): nouns or verbs that consist of multiple words, e.g., "mix in herbs." or "egg whites".

As a baseline, the StanfordNLP parser [31] (further described in section 4.3.1.1), which is one of the recent standard NLP tools, was run on all 50 recipes. Its output was presented as the base annotation to the annotators, who then fixed the dependencies and POS-tags in brat.

We measured the inter-annotator agreement using Cohen's kappa on the syntactic annotation task. We only took into account a dependency annotation if the first or second annotator labeled the dependency as one of the relevant relations listed above. In this case, both the label and the head were included in the calculation. Over all 50 recipes, an inter-annotator agreement of 0.891 was reached. This is considered a high agreement in the NLP domain [1].

After going through 10 of the 50 recipes together and discussing ambiguous cases, one annotator adjusted annotations on the remaining 40 recipes. The resulting dependencies and POS-tags were used as the Gold labels in the evaluation and further annotation.

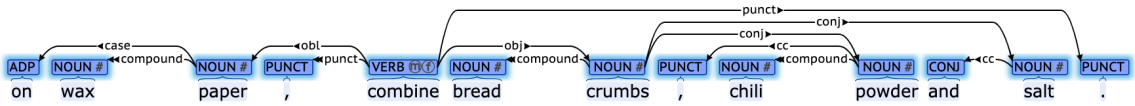
Table 4.2 shows the performance of the StanfordNLP dependency parser on this dataset for selected categories, as compared to the Gold labels. Considering that the current state of the art in syntactic parsing of general text (the Penn Treebank corpus [32]) has an F1-score of 0.96 and an F1-score of 0.97 in POS-tagging as of September 2019 [49], the

results show that syntactic parsing of recipes is very different to syntactically parsing other types of texts. If the syntactic parse is used as an input to a semantic parser, especially the verb detection recall of 0.88 is problematic: This means that 12% of verbs are missed, because they are misclassified as nouns or other parts of speech by the POS-tagger.

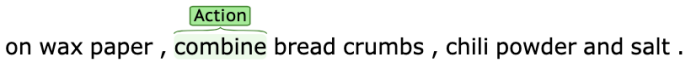
Section 4.2.3 analyzes the syntactic characteristics of cooking recipes that cause a general-purpose syntactic parser trained on newspaper text like the StanfordNLP dependency parser to fail when run on cooking recipes.

4.2.2. Semantic Annotation

In a second phase, we took the Gold dependencies and POS-tags from the previous step and converted them to a simple baseline for the semantic annotation. These conversion steps will be illustrated by applying them to the following sentence from our dataset:

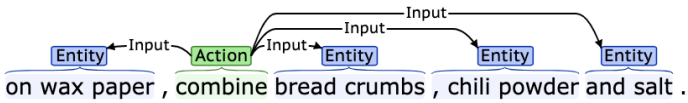


1. Mark words tagged as verbs including all compound dependents as the trigger of an `Action`:

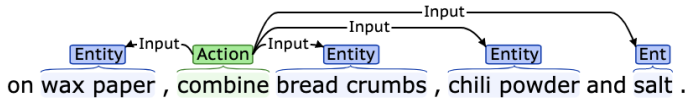


2. Apply a manually created verb-frame mapping to the lemmatized trigger to get the frame. In the example sentence, assign the `Merge` frame to the "combine" action.

3. Attach all oblique and object dependents of the verb including all their dependents (recursively), as `Input` entities to the action (also including conjunctions of objects or oblique dependents):

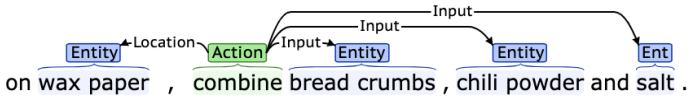


4. Prune away leading or trailing prepositions or coordinating conjunctions of the entities:



This baseline annotation was presented to the two annotators. Both annotators then independently went through all 50 recipes again, expanding and fixing the semantic annotation. In particular, this involved classifying the automatically generated input entity relations into the different cases of location, input, condition or utensil (see section 4.1.2 for the full semantic annotation schema).

For the example sentence used above, the correct semantic annotation is the following (where the first input, "wax paper", is re-classified as a `Location` relation):



4.2.3. Syntactic Characteristics

As shown in Table 4.2, general-purpose syntactic parsers trained on newspaper texts (and other non-recipe genres) do not perform well on cooking recipes. When correcting the dependency annotations proposed as the baseline by the StanfordNLP parser, we noticed a number of mistakes that were commonly made by the parser. The syntactic characteristics of cooking recipes that make parsing them different from parsing other types of text are imperative language, implicit objects, omissions and uncommon part-of-speech tags. Each characteristic is described in the following.

4.2.3.1. Imperative Language

Instructions in cooking recipes are mostly formulated in imperative language. In our dataset, only 50 of the 747 verbs have a subject (in the Gold annotation). This affects the performance of POS-taggers and dependency parsers considerably, as it is a clear difference from other types of texsts such as news articles, which are typical training samples of POS-taggers and dependency parsers (e.g., the Penn Treebank corpus [32]).

In most cases, the implicit subject of the verbs is the cook performing the instructions. However, if the subject is named explicitly, in most cases it is not the cook but an ingredient. The verb in this case usually describes a goal condition of the ingredient. Figure 4.1 presents an example sentence from the dataset.

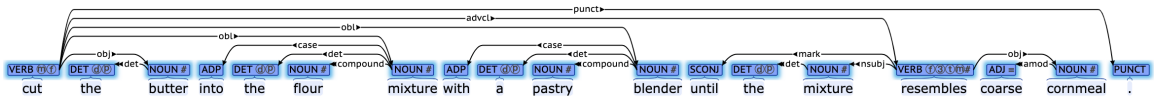


Figure 4.1.: A sentence from the dataset in which the verb "resembles" has a subject ("the mixture") and describes a goal condition.

A possible solution is to insert a subject automatically in every verb clause where the syntactic parser does not detect a subject and then run the parser again. In cooking recipes, one might just insert the pronoun "You" before each verb lacking a subject.

	[min, max]	mean
# of verbs per sentence	[0, 6]	1.85
# of obj per verb	[0, 2]	0.42
# of obl per verb	[0, 4]	0.52

Table 4.3.: Ranges and average number of objects (obj) and oblique nominal dependents (obl) of verbs in our dataset. Based on 747 verbs in 404 sentences and 50 recipes.

4.2.3.2. Implicit Objects

Not only subjects but also objects of intransitive verbs are frequently omitted in cooking recipe instructions. This is known as zero anaphora, which occurs very infrequently in the English language [18], which is one reason why current NLP tools like the StanfordNLP parser do not perform well on cooking recipes.

Semantically, objects or oblique nominal dependents of a verb are usually the input ingredients of the action that the verb implies. As we found later in the semantic annotation, inputs of an action in sentence at position n in the recipe include inputs from previous sentences at positions $\leq n - 1$ in 14% of cases (see Figure 4.3). Table 4.3 presents statistics on how many objects or oblique nominals verbs in our dataset typically have.

Most importantly, for the 747 verbs the average number of objects is 0.42, meaning that verbs with no object predominate. In the Gold labels of our dataset, 432 verbs (57.8%) have no object, 410 (54.9%) have no oblique nominal dependent, and 255 (34.1%) have neither.

Figure 4.2 correlates the normalized step number in the recipe with how often objects were not named explicitly (all recipes were normalized to 10 steps by linearly mapping the sentences to 10 buckets). As one can see, verbs in later steps of the recipe are more likely to miss objects. In the first sentence of recipes, only 36.11% of verbs have no explicitly mentioned direct object. This number is 55.00% for the second sentence (normalized to a 10-step recipe) and increases up to more than 70% in the last steps of the recipe. A possible interpretation is that in later steps of the recipe, the objects that verbs act on are the outputs of previous sentences. This is often assumed implicitly, and thus verbs in later steps are less likely to have a direct object named explicitly in their sentence.

Consider the following two sentences from our dataset:

1. Combine flour, sugar and salt and add to egg whites.

2. Beat until smooth, losing grainy look.

The object of “add” in the first sentence is not explicitly named. Semantically, it is the output of the previous action (“combine”), which in this case is in the same sentence. Similarly, the object of “beat” in the second sentence is omitted. The thing that is beaten is the mixture that has been created in the first sentence.

4.2.3.3. Omissions

In addition to omitted syntactic constituents like objects, other words like prepositions are also frequently left out in cooking recipes. Our dataset’s recipes stem from the NYC

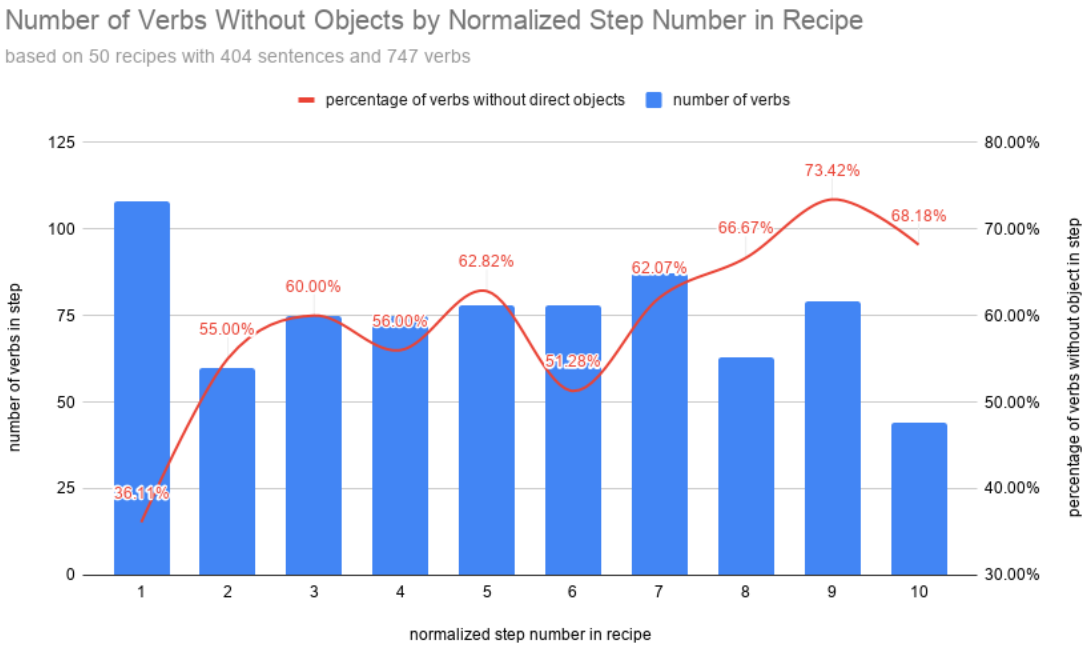


Figure 4.2.: Correlation of the step number in the recipe and how many verbs have no direct object.

dataset, which was originally created from mailing lists in the 1990s [22]. This means that the recipes were not written by professionals, which might explain the informal language. A common example occuring multiple times in the dataset is an omitted preposition in an oblique nominal dependent of a verb that describes a temporal condition, such as the duration of the action. This causes the syntactic parser to misclassify the constituent as an object.

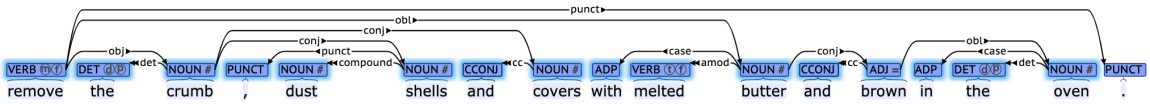
This is illustrated by the following example: “Add the other ingredients and marinade 24 hours.” The preposition “for” of the oblique nominal dependent “24 hours” has been left out. The pre-trained StanfordNLP dependency parser thus misclassified “24 hours” as the object of “marinade”.

If one uses the syntactic parse for the downstream task of semantic parsing, it makes a big difference if one marinades some ingredient for 24 hours or one tries to marinade the duration of 24 hours itself.

4.2.3.4. Uncommon Part-of-Speech Tags

Further analysis of the POS-tagging errors showed that verbs that are commonly used as nouns are often used as verbs in cooking recipes. Examples include “top”, “dust”, “heat”, “press”, “cap”, “frost”, “spray”. Incorrect POS-tags frequently cause the dependency parser to misunderstand the syntactic structure of the sentence. this results in objects of a verb being identified as noun-modifiers instead, which again makes semantic parsing difficult if the syntactic parse is used as an input.

In addition, if verbs are misclassified as nouns, the dependency parser also has trouble distinguishing between enumerations of ingredients and actions. This is illustrated by the following sentence (which is annotated with the (incorrect) StanfordNLP dependency parser's output):



The semantically correct interpretation is that one needs to first remove the crumb and then dust the shells (i.e., "dust" as a verb), which is an enumeration of two actions. However, the syntactic parse suggests otherwise: "dust" is misclassified as a noun, which presumably causes the parser to interpret it as another ingredient of the "remove" action.

Similarly, "brown" is misclassified as an adjective though it is used as a verb here. The syntactic parser attaches it as a conjunct to "butter", which might again cause a semantic parser using the syntactic parse to treat it as an enumeration of ingredients.

4.2.4. Semantic Parser Challenges

During annotation, we also identified some challenges in the cooking recipe domain that concern semantic parsers. When designing the architecture of a semantic parser for cooking recipes, the following should be considered.

Some of the challenges only concern semantic parsers that use a syntactic parse as an input. This is only one possible design choice, as is more throughly examined in section 2.2.4.4. Parsers that go directly from the natural language text to the semantic representation may naturally accomodate these challenges, depending on their architecture.

4.2.4.1. Multiple actions per sentence and verb

A parser must be able to evoke multiple action frames per sentence and per verb. In some contexts, action verbs imply two distinct actions that are performed on the same or different inputs. For example, in the sentence "Cut the tomatoes into the bowl", "cut" implies two distinct actions:

1. Cutting the tomatoes (i.e., a `SizeChange` frame)
2. Moving the cut tomatoes into the bowl (i.e., a `LocationChange` frame)

4.2.4.2. Compound verb triggers

In some cases, actions are not triggered by a single verb, but by phrasal verbs or even the verb phrase. Depending on the compound words, different types of action frames might be semantically correct. For example, "remove from heat" should be treated as one action trigger for a `TemperatureChange` or `LocationChange` frame (or both). A possible solution is to assign action frames not to trigger words, but to a whole sentence or clause. This is the approach we will take in the semantic annotation of a larger recipe dataset, which is planned for the end of the year 2019 but is not part of this thesis.

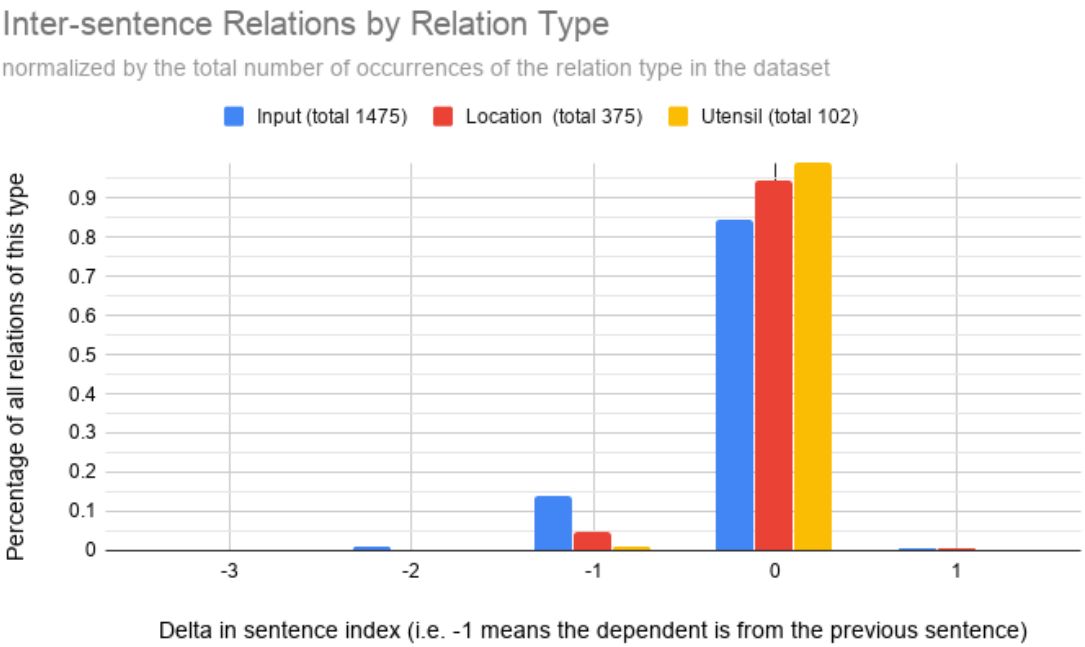


Figure 4.3.: Inter-sentence relations in our 50 recipes dataset (in the gold labels).

4.2.4.3. Inter-sentence relations

249 out of 1952 relations in the Gold labels of the semantic annotation are between frames which are evoked in different sentences. Since objects of action verbs are frequently ommitted in recipes (see 4.2.3.2), often the input to some action is the output of the previous action (207 out of 1475 `Input` relations in our dataset). Figure 4.3 shows the sentence index delta between the head frame of a relation and the dependent frame of this relation, categorized into different relation types. As one can see, for all three relation types (`Input`, `Location` and `Utensil`), the majority of relations are between frames evoked in the same sentence (i.e., sentence index delta 0). In 14% of `Input` relations, the input comes from the previous sentence (i.e., sentence index delta -1). This is only the case for 4.8% of `Location` relations and 1% of `Utensil` relations.

Inter-sentence relations with a sentence index delta of more than one are very rare. In our dataset, 99.28% of all relations are between sentences with a maximum distance of 1. 98.82% of all relations are between either frames evoked in the same sentence or the previous sentence.

To conclude, a semantic parser for cooking recipes must allow such inter-sentence relations. Most semantic parsing studies however focus on sentence-level parsing [21].

4.3. Models

In this case study, we will evaluate two very different semantic parsers on our 50 recipes dataset and specifically examine how each performs with respect to some of the challenges identified during annotation.

The first parser is ocra, a rule-based parser which has been specifically built for our project. It uses a syntactic parse as input and maps it to a semantic parse using manually created rules and lexicons.

The second parser is SLING, a general semantic parsing framework by Ringgaard et al. [41]. It is based on a neural architecture that is trained end-to-end without requiring the manual design of rules or lexicons.

Both parsers have advantages and drawbacks, as will be shown in section 4.4.

4.3.1. ocra: A Rule-based Parser

The cOmmonsense-based Cooking Recipe Analyzer (ocra) is a rule-based semantic parser for cooking recipes. It was developed as part of this project by another person who is not the author of this thesis. ocra has been inspired by ontology-based methods [10] for mapping a syntactic parse to semantic relations. Its rules and lexicons have originally been created specifically for omelet recipes, but with some adjustments (adding a few words to the lexicon) we will use it as one of the parsers evaluated on our dataset (which contains recipes for arbitrary dishes). In contrast to SLING (described in section 4.3.2), ocra uses manually created rules and lexicons to map a syntactic parse obtained from a separate syntactic parser to a semantic representation. The syntactic parser it uses to obtain the syntactic parse is the StanfordNLP dependency parser [31], one of the recent standard NLP tools, which will be briefly described in the following.

4.3.1.1. The StanfordNLP Dependency Parser

Transition-based Parsing Transition-based parsing is a type of parsing algorithm that can be used to parse a sequence of tokens. The basic idea is rooted in the shift-reduce algorithm, originally developed for parsing programming languages. By replacing the formal grammars used in parsing code with a classifier that determines the next transition, the algorithm can be adapted for parsing natural language as well.

Arc-based dependency parsing The arc-based transition system used for dependency parsing [11] is an adjustment of the shift-reduce algorithm to the problem of syntactic parsing. The basic idea is that a classifier predicts one out of a fixed set of possible transitions, based on features describing the current state.

In the beginning, all tokens are in the buffer, including a special `ROOT` token at the top. By predicting the `SHIFT` transition, the parser pushes the next token onto the stack. To create a dependency relation between the two topmost tokens on the stack, a `RIGHT-ARC` or a `LEFT-ARC` transition is predicted (depending on which of the two words is the head and which is the dependent). The dependent is popped from the stack, and the newly created relation is added to the list of dependency relations. Features helping the classifier

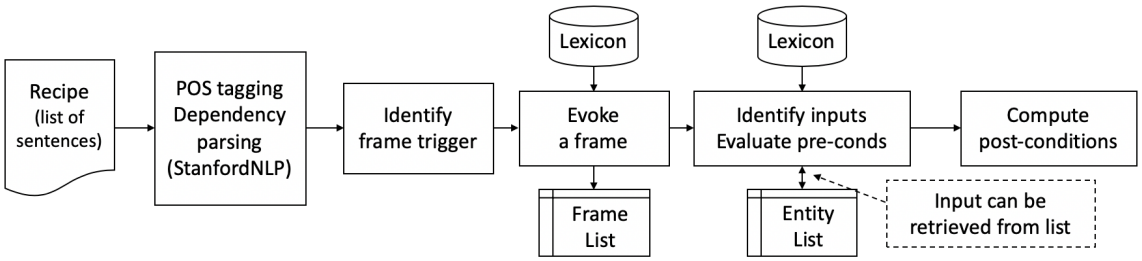


Figure 4.4.: ocra's recipe analyzer pipeline.

to predict the correct transition at each step might be the top words of the buffer and stack, existing dependency relations and POS-tags. If labeled dependency relations are required, `RIGHT-ARC(label)` and `LEFT-ARC(label)` transitions exist for each possible relation `label`.

The StanfordNLP Dependency Parser The Stanford CoreNLP framework provides most of the common natural language processing steps, from tokenization to coreference resolution [31]. For ocra, only the tokenization, part-of-speech (POS) tagging and syntactic parsing steps are required. The POS-tagger uses a log-linear bidirectional dependency network [44] and outputs POS-tags for each token. The syntactic parser uses the input token sequence along with the POS-tags and outputs a labeled dependency tree. The model used is a neural network dependency parser [8] which predicts transition as described above. It has been trained on the English Penn Treebank [32], which is, notably, a general dataset comprised mainly of text from newspapers, a very different style than the style encountered in cooking recipes (cf. 4.2.3).

4.3.1.2. Pipeline

ocra's pipeline from recipe to frames is illustrated in Figure 4.4 [1].

After the StanfordNLP POS-tagger and syntactic parser have been run on all sentences of the recipe (step 1), ocra goes through the sentences one by one. For each sentence, ocra identifies a frame trigger by pre-defined trigger patterns.

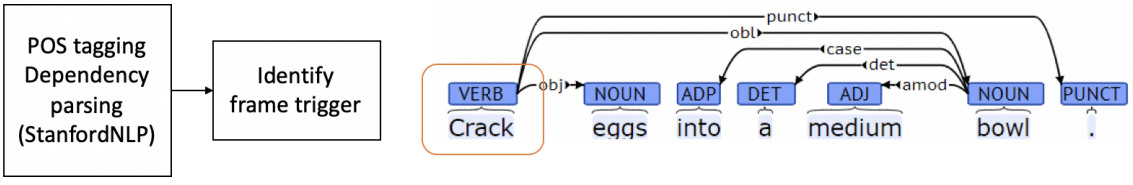


Figure 4.5.: Step 2 of ocra's pipeline. "crack" is identified as a trigger for the `Divide` frame.

In the third step, the frame referenced by the trigger identified in step 2 is evoked. The frame comes from a pre-defined frame list which also contains pre- and post-conditions (which is relevant for the last two steps).

[1] The example illustrations in this section are taken from the ocra documentation with some small adjustments by the author.

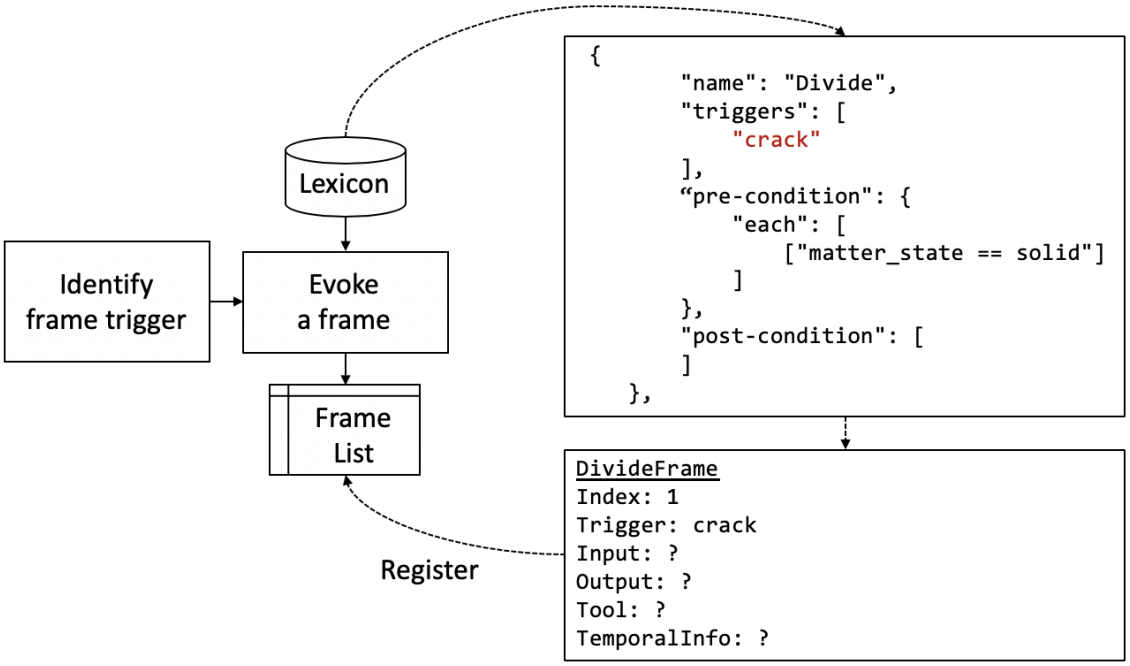


Figure 4.6.: Step 2 of ocra's pipeline. The `Divide` frame is evoked and "crack" is filled in as the trigger.

The slots of the frame evoked in step 3 are filled in step 4. Given the syntactic parse, potential objects or oblique dependents of the trigger verb are added as inputs to the frame, and — depending on which information is given in the lexicon — eventual output entities are filled in as well. If the frame contains pre-conditions applying to its inputs (such as that the matter state of something that is divided must be solid, as in the example), they are checked based on the properties of entities in the Entity List.

In the last step, potential post conditions applying to the output entities given by the evoked frame are evaluated. In the example, the `Divide` frame does not have any post-conditions, so this step is skipped.

4.3.1.3. Pre- and Post-conditions

In addition to evoking frames and filling their slots, ocra also computes and evaluates pre- and post-conditions for each frame and each entity. These are relevant for the downstream task of gap analysis, but are ignored in our evaluation, because this study is concerned only with the semantic parsing part. Also note that SLING, the other parser evaluated in this study, does not have this functionality.

4.3.2. SLING: A Neural Parser

SLING [41] is a framework for semantic parsing. It is based on a neural architecture that is trained end-to-end without any intermediate symbolic representations like dependency parse trees. Its output is a frame graph that represents the meaning of the input token

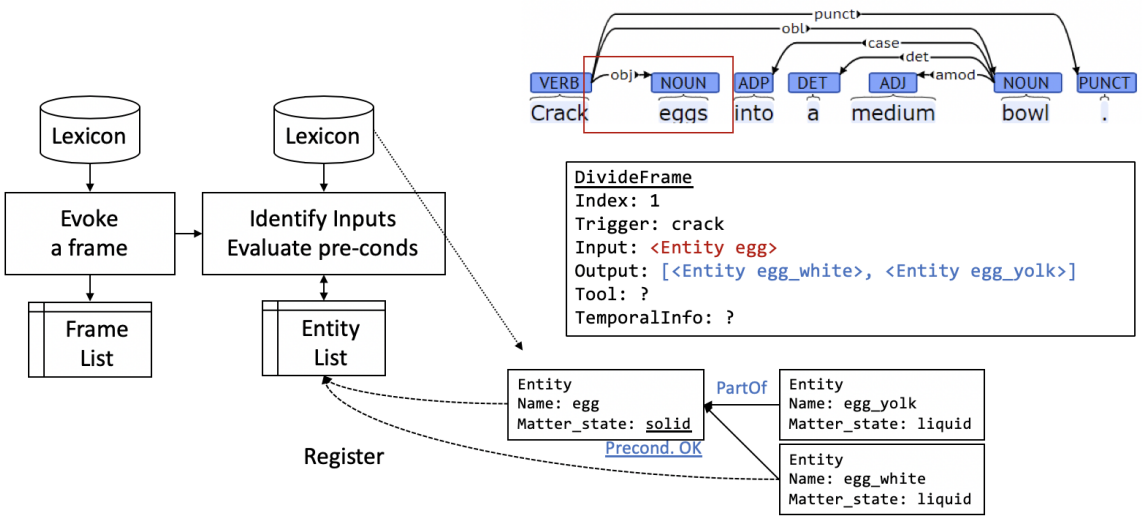


Figure 4.7.: Step 4 of ocra's pipeline. "eggs", as an object of "quote" is added as an input. From the lexicon, ocra knows that egg whites and egg yolks are the parts that eggs consist of. It also knows that the `Divide` frame splits up an entity into its components, and thus fills in the output slot with egg whites and egg yolks.

sequence (see section 2.2.2.2). In the following, SLING's architecture, training process and characteristics will be briefly examined.

4.3.2.1. SLING's transition system

SLING's transition system is based on the same idea as the general transition-based parsing framework described earlier in section 4.3.1.1. The difference is that, while in dependency parsing only a tree connecting tokens is predicted, SLING gradually builds a generic frame graph representing the semantic meaning of the token sequence it receives as an input. It does so by predicting transitions such as "evoke frame X", "link frame X and Y on relation Z", and "set attribute X in frame Y to value Z". SLING's default schema features 8 different *types* of transitions. Some transition types have arguments (such as the name of the frame to evoke), thus the total number of possible transitions depends on the frame lexicon and other factors specific to a given dataset. The number of possible transitions in our dataset is further examined in 4.3.2.3.

The most important transition types SLING uses are:

- `SHIFT`: moves to the next input token
- `STOP`: signals that the parse is completed
- `EVOKE(type, n)`: evokes a frame of type `type` from the next `n` tokens of the input
- `CONNECT(source, role, target)`: adds the `role` slot to the `source` frame with a value of `target`

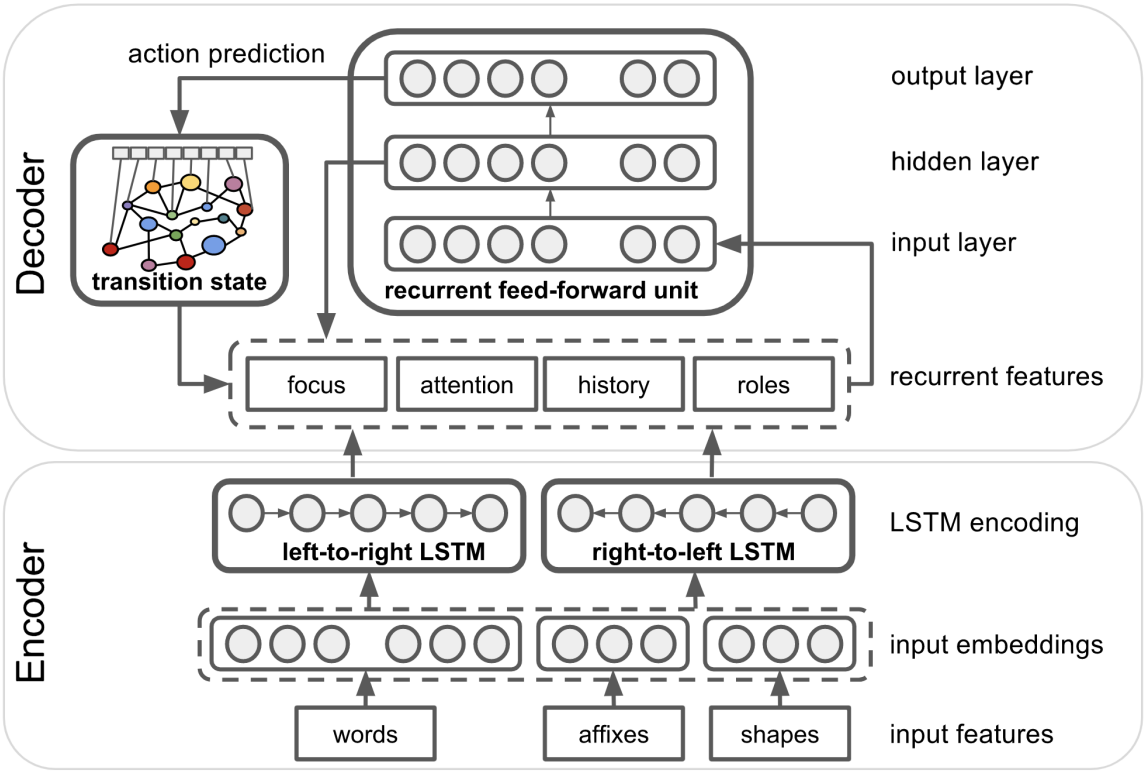


Figure 4.8.: SLING's encoder-decoder architecture. Taken from [41].

4.3.2.2. Architecture

On a high level, the SLING parser has the following architecture (illustrated in Figure 4.8):

1. The input text tokens are mapped to randomly initialized embedding vectors.
2. A bidirectional LSTM (long short-term memory, [16]) encoder network reads the sequence of embedding vectors.
3. The hidden states of the LSTM cells form the basis for the recurrent features, which are used as the input to the recurrent feed-forward unit.
4. The transition based recurrent unit (TBRU) predicts a transition at each step (using a final softmax layer), which is then applied to the frames in the frame store.
5. The frame store is updated and thus the recurrent features (input of the TBRU) for the next step are changed.

The frame store The frame graph is gradually built as the transition system reads the input token sequence. Similar to the list of existing dependencies in the arc-based transition system, SLING uses a frame store to save the partial frame graph. This frame store is a container that keeps recently updated or created frames at the top. A neural representation of the topmost frames in the frame buffer forms a part of the input of the TBRU, which predicts the transitions to execute on the frames at each step based on these features.

This neural representation consists of the hidden layer activations of the TBRU at the transition steps which brought the top-k frames to the top of the frame store. It serves as a continuous representation of the k most recently evoked or referenced frames.

Training Training token sequences are represented as a special type of frame in SLING's frame format. These `Document` frames include a list of all tokens in the input sequence and a list of all frames that are evoked by these tokens. SLING uses a deterministic algorithm to transform a set of frames into a sequence of transitions which the TBRU must predict in order to build the correct frame graph. This transition sequence is used as the Gold sequence and compared to the predicted sequence of the TBRU, using beam search. The loss is the summed cross entropy between the Gold transitions and the predicted probability distribution over all possible transitions at all time steps.

4.3.2.3. Adjustments for our Dataset

Number of possible transitions In the original SLING paper, the parser is trained and evaluated on the OntoNotes corpus [17], which uses PropBank frames [39]. In their experiments, the parser predicts one of 6,968 possible transitions. This large number of possible transitions mainly comes from the many different frame type arguments in the `EVOKE` transition. For our dataset, with only about 15 frames, the number of possible transitions to predict is much smaller (51 to construct the frame graphs for all 50 recipes). In addition to making the final softmax calculation much faster, this also reduces spareseness and makes the learning task simpler.

Inter-sentence relations Since SLING does not support inter-sentence relations (i.e., two frames from different sentences being linked together) but those kinds of relations appear frequently in our data (cf. section 4.2.4 and Figure 4.3), we treated the whole recipe as one sentence. The SLING authors are planning to support inter-sentence relations in a future version of SLING, however [2].

4.4. Evaluation

4.4.1. Training

ocra is a rule-based parser and thus does not need to be trained. We directly ran it on our dataset to obtain its annotations. SLING's neural engine however needs to be trained. The training process and results are described in the following.

4.4.1.1. Training SLING

As our dataset of 50 cooking recipes is very small compared to datasets usually used for training neural networks, this can be seen as more of a proof-of-concept experiment than training a well-performing semantic parser. It shows that training a neural semantic parser

[2] https://github.com/google/sling/issues/129

Split	# of recipes	# of tokens	# of frames
training	35	3,282	1,673
development	15	1,178	564

Table 4.4.: Training and development splits for training SLING. Frames, here, are both frames for actions and also frames for entities (based on the Gold labels).

to predic the annotation schema we designed is generally feasible. If one wants to create a semantic parser based on a neural engine like SLING that works well on a wide range of cooking recipes, considerably more training data is required.

Data preparation For training SLING, we randomly split up our 50 recipes into 35 recipes for training and 15 for development. The dishes of the recipes in both splits are independent. Thus, when evaluating the model on the development set, it needs to annotate different dishes than the ones seen in training. Details can be found in Table 4.4.

Model The SLING model (using default hyperparameters) we trained has the following specification:

- Token vocabulary size (number of embedding vectors): 735
- Number of possible transitions (final softmax output dimension): 48
- Number of parameters: 1.1 million
- Hidden dimension of LSTM cells: 256

Note that the number of possible transitions (final softmax ouput dimension) is around 8,000 in the SLING paper [41], which makes our learning problem much simpler (cf. section 4.3.2.3).

Training setup We use the standard training and evaluation script that is open-sourced as part of SLING [3]. We use a batch size of 8 recipes and the Adam optimizer [23] with a learning rate of 0.0005, $\beta_1 = 0.1$ and $\beta_2 = 0.999$. We clip gradients at 1.0 to avoid the exploding gradients problem [4]. If not specified otherwise, we use the default hyperparameter values as defined in the training script.

Learning constraints The model only has a small number of training examples to train its 1.1 million parameters from (35 recipes). In addition, training and development recipes come from different dishes. Thus, we expected the model to not learn much at all (i.e., the evaluation metrics to stay close to zero). As the results suggest however, even if this restricted setting, the model was able to obtain a considerably high performance when annotating the 15 development recipes. The metrics, results and training progress are described in the following.

[3] See https://github.com/google/sling

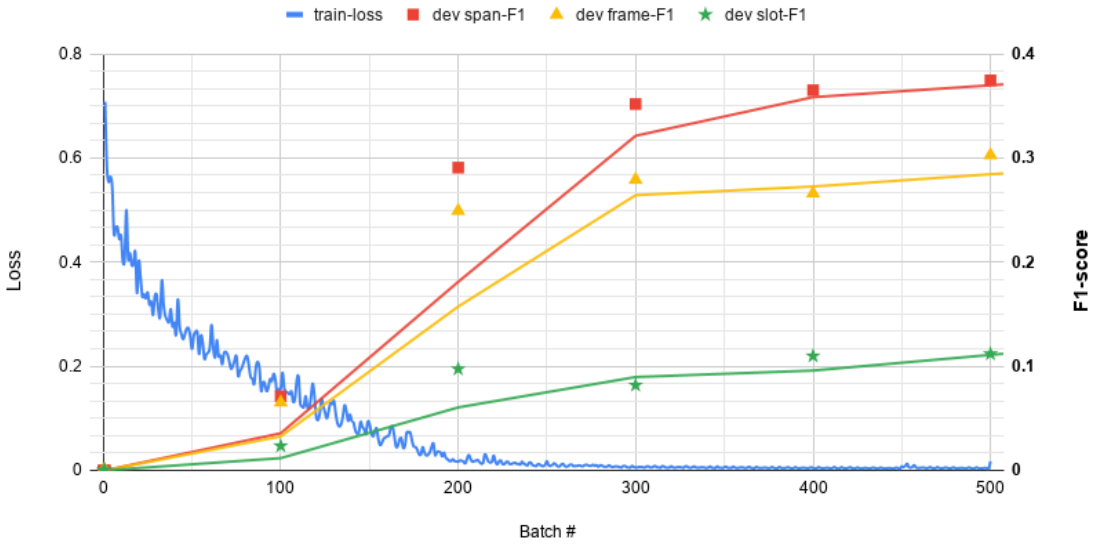


Figure 4.9.: Learning curves for training SLING (train set: 35 recipes, dev set: 15 recipes).

Evaluation metrics SLING's training and evaluation script [4] computes a number of differed evaluation metrics. We focus on the most relevant three. The **span-F1** gives the F1-score of the frame trigger span detection accuracy (i.e., detecting which tokens evoke an action or entity frame). The **frame-F1** gives the F1-score of frame classification accuracy (i.e., which frame is evoked by a token sequence). Finally, the **slot-F1** gives the F1-score of linking different frames together (e.g., linking an input ingredient frame correctly to its action frame). Note that these metrics are based on graph matching algorithms and thus allow no comparison to the metrics we define for comparing the different models in section 4.4.2.1.

Training results Even with only 35 recipes to learn from, the model was able to obtain the following performance metrics after convergence (circa 500 batches, which corresponds to about 114 epochs): **span-F1** = 0.37, **frame-F1** = 0.3, and **slot-F1** = 0.11. The learning curves are shown in Figure 4.9 (plotting the loss on the training set, and the three evaluation metrics on the development set). One can see that as the training loss steadily decreases, the three performance metrics on the development set increase. This means that the model does not just memorize the correct annotations for the 35 training recipes, but learns useful features that help it annotate the other 15 recipes with increasing accuracy.

[4] Training script: https://github.com/google/sling/blob/master/sling/nlp/parser/tools/train.sh

4.4.2. Performance

4.4.2.1. Metrics

We are interested in three performance metrics: trigger identification (which tokens evoke an action frame or entity), action classification (which type of action frame is evoked) and input identification (which action or entity is the input of which action).

Trigger identification In this metric, we only distinguish between entities and actions, not the specific type of actions. We consider the whole token sequence and compare the Gold annotations for actions and entities with the predicted annotations. We report precision, recall and the F1-score on a per-token basis (i.e. rewarding partial matches).

Action classification This metric describes how well actions are classified into the correct type. We treat action classification as a sequence labeling problem, consider the whole token sequence and compare the Gold action annotations with the predicted action annotations. We report precision, recall and the F1-score on a per-token basis (i.e. rewarding partial matches). These values are macro-averaged for all action frame types (i.e., each frame type is weighted equally, regardless of how often the frame occurs). Note that this metric depends on the trigger identification accuracy: if the parser does not predict an action for a given token at all, the action type cannot be correct.

Input classification This metric quantifies how well the inputs of actions (entities or other actions) are identified. We treat this as a binary classification problem: out of all possible action-action (both directions) and action-entity relations, how many are predicted, and how many are correct? In this metric, we do not reward partial matches, but report precision, recall and the F1-score only for perfect matches. Note that this metric depends on trigger identification, because the correct inputs can only be identified if the respective entities and actions have been identified correctly.

4.4.2.2. Results

We report results for the following 4 pipelines on the metrics defined above, when evaluated on the 15 evaluation recipes: Heuristic (the simple heuristic syntax-to-semantic conversion script described in section 4.2.2, using the Gold labels as the syntactic parse), SLING-35 (trained on the 35 training recipes), ocra-Stan (using the StanfordNLP parser for the syntactic parse) and ocra-Gold (using the Gold labels as the syntactic parse).

To acquire the scores of the different pipelines on our dataset, we developed a common format and created converters for each of the output formats. Details about this format can be found in Appendix B.

Discussion In general, as one can see, Heuristic performs best on action classification and input identification by a considerable gap (absolute F1 differences of 0.08 and 0.21, respectively). In trigger identification, ocra-Stan performs about as well as Heuristic, and ocra-Gold beats the two by an absolute F1 difference of about 0.06. If one ignores the

Metric		Heuristic	SLING-35	ocra-Stan	ocra-Gold
Trigger id.	P	0.62	0.45	0.62	**0.70**
	R	0.73	0.50	0.69	**0.77**
	F1	0.62	0.44	0.61	**0.68**
Action cls.	P	**0.63**	0.24	0.47	0.52
	R	**0.62**	0.26	0.47	0.53
	F1	**0.59**	0.24	0.45	0.51
Input id.	P	**1.00**	0.33	0.86	0.85
	R	**0.62**	0.01	0.36	0.41
	F1	**0.77**	0.02	0.50	0.56

Table 4.5.: Precision (P), recall (R) and the F1-score on the 3 metrics on the evaluation dataset.

two pipelines using the Gold syntactic annotations and only compares ocra-Stan against SLING-35, ocra-Stan performs better in all three categories (and all of precision, recall and F1), by F1 differences ranging from 0.17 to 0.48.

Heuristic achieves an input identification precision of 1.0, which means that all input entities predicted by the heuristic rules based on the Gold syntactic parse are actually input entities according to the Gold semantic annotation. Its recall in this category is considerably lower (0.62), which however is still 0.21 better in F1 than all other pipelines. While SLING-35's input identification precision is quite low (0.33), its recall is almost zero (0.01). This means that SLING-35 has difficulty linking input entities to the correct action. Its trigger identification F1 score is 0.44, which although the lowest among the other pipelines still suggests it does correctly identify some action triggers. However, the input identification metric (other than the other two metrics) only rewards perfect matches, which might explain that SLING-35's recall in this category is so low.

The overall results suggest that the Gold syntactic parse is a useful input. When using the Gold parse instead of the StanfordNLP dependency parser's output, ocra's F1 scores improve in all three categories, by absolute differences of 0.06-0.07. In addition, when averaging the F1 scores in the three categories, Heuristic is the best model in the comparison with an F1-score average of 0.66, followed by ocra-Gold with an F1-score of 0.58.

Still, one must keep in mind that Heuristic has been specifically built for our dataset and also served as the baseline for the Gold semantic annotations, while SLING-35, as mentioned before, has only 35 recipes available for training. Thus, this comparison cannot be seen as a fair evalation of the pipelines themselves, only the specific models we built as part of this thesis. Once more training data becomes available, we expect that SLING-35's performance improves considerably and might eventually surpass ocra's. This is however left to future work, as our larger dataset will be available only after the completion of this thesis.

4.4.3. Semantic Parser Challenges

In addition to the accuracy as measured by the three metrics defined in 4.4.2.1, we examine ocra and SLING with respect to some of the challenges in cooking recipes we identified in section 4.1.1, both qualitatively and quantitatively.

4.4.3.1. Multiple Actions per Verb

In our 50-recipe dataset, with 564 total action frames, there are 117 trigger overlaps. This means that 20.7% of action frames are triggered by tokens that already evoke another action frame. This can be both a partial overlap (e.g., in "crack into bowl", "crack" and "crack into" each evoke a frame) and a full overlap (e.g., "crack" evokes two frames) (we only annotated at most two frames per trigger).

ocra While ocra supports and also searches for overlapping triggers, it does not allow the evocation of mutliple frames for the same trigger. This is a design choice that was made based on the empirical result that otherwise, too many action frames would be evoked (i.e., reducing the parsing accuracy).

SLING Given by its transition system, SLING does allow both multiple frame evocations per trigger and overlapping triggers. It can predict two `EVOKE(n, type)` transitions (evoking a frame of type `type` from the next `n` tokens) without a `SHIFT`-transition in between (moving the current position one token further). It must, however, learn to do so. In most cases in the dataset, one token evokes at most one frame, therefore SLING mostly learns to predict `n` `SHIFT`-transitions after each `EVOKE(n, type)` transition.

Evaluation In the Gold labels, 19 of 136 action frame triggers (14%) overlap with another action frame trigger. In contrast, SLING only predicts overlapping triggers in 3 of 119 cases (2.5%), while ocra is close to the Gold labels in this respect, with 15 out of 110 (13.6%). This suggests that SLING might have a bias against overlapping frame triggers, as for most frames it needs to predict `SHIFT`-transitions after each frame evocation.

4.4.3.2. Multiple Actions per Sentence

In 148 of 404 (36.63%) of sentences in our 50-recipe dataset, more than one action frame is evoked. These means that there are two or more distinct actions mentioned from different triggers in the sentence (in contrast to multiple frames per trigger, discussed above).

ocra ocra tries to match as many action triggers on one sentence as it can find. If there are mutliple matches (i.e., multiple words in a sentence have a match in ocra's trigger lexicon), it just evokes action frames for each one and assumes them to be in order of appearance in the sentence. Thus, its accuracy of evoking an action frame does not depend on whether the actions are mentioned in different sentences or the same sentence.

SLING SLING needs to learn to evoke multiple action frames per sentence where needed. As the current version of SLING does not support inter-sentence relations, we are treating whole recipes as one sentence in our training setup. Thus, other than the dot-token after each sentence, SLING has no concept of separate sentences and one might expect it to approximately match the training dataset's frequency of evoking an action frame per recipe.

Evaluation In the 119 sentences in the evaluation dataset, the Gold labels contain more than one action frame in 32 sentences (26.9%). SLING evokes multiple frames per sentence in 30 (25.2%) and ocra in 19 sentences (16%). Interestingly, when ocra is given the Gold syntactic parse (ocra-Gold), it evokes multiple action frames in 33 sentences (27.7%). This suggests that the errors by the StanfordNLP parser hinder ocra of evoking multiple action frames in some sentences. This might be related to a wrong verb-object structure caused by incorrect POS-tags for some action verbs (see section 4.2.3.4).

4.4.3.3. Inter-sentence Relations

Identified as a challenge earlier, inter-sentence relations (i.e. an action-input or action-action relation in which head and dependent come from different sentences) are a common occurrence in cooking recipes. In our dataset, 229 of 1577 relations (14.5%) span multiple sentences.

ocra ocra creates inter-sentence relations with the simple assumption that if a slot in a frame cannot be filled with entities from the same sentence, the most recently created entity (input or location) that hasn't been "consumed" by another frame is used. This entity might come from the previous sentence, or, in some cases even come from earlier sentences in the recipe. ocra keeps track of all entities on its Entity List (see Figure 4.4) and marks them as used when they have been consumed as an input by some frame. Most frames also generate an entity as their output, which is how ocra can set the input to an action the output of another action.

SLING When training and evaluating SLING, we treat the whole recipe as one sentence, so it can predict inter-sentence relations. Thus, it is natural for SLING to link actions and entities from different sentences together, as it does not consider sentence boundaries at all. While its attention buffer may bias SLING in favor of predicting relations between recently evoked frames, in theory it can predict any relation between entites and actions mentioned anywhere in the recipe.

Evaluation Figure 4.10 compares how often each of the four pipelines described in section 4.4.2.2 predicts inter-sentence relations. One can see that the variance in sentence index delta is highest for SLING-35, which can be explained by the fact that each recipe is treated as one sentence.

Heuristic does not support inter-sentence relations (which it also cannot, because it only considers the syntactic parse, which is inherently limited to the sentence boundary). Thus, all its relations are intra-sentence (i.e., a sentence index delta of 0).

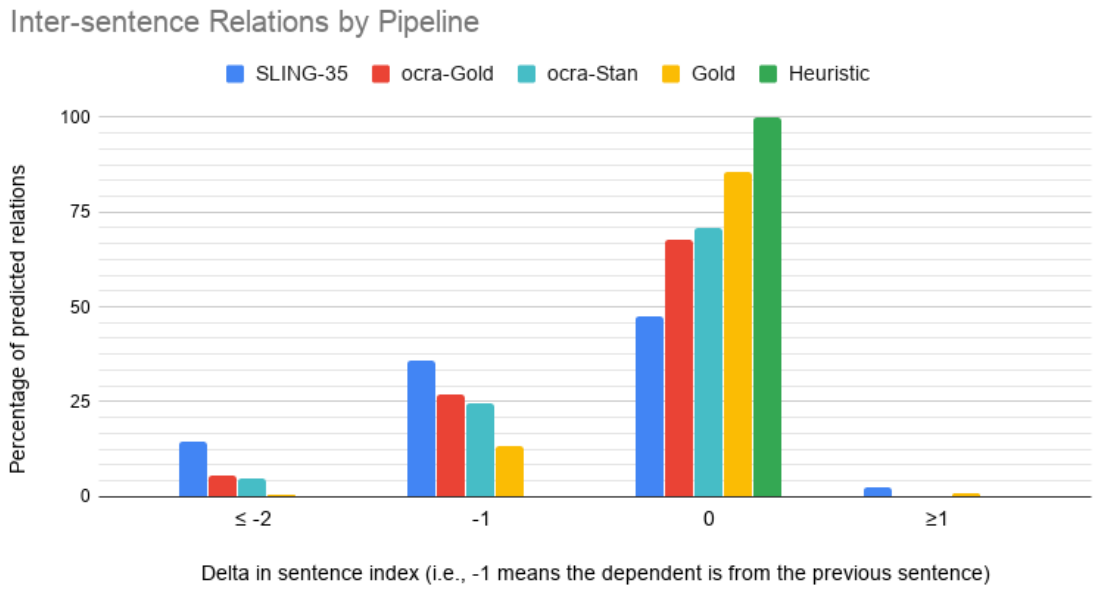


Figure 4.10.: Inter-sentence relations for the four pipelines compared to the Gold annotations in the evaluation dataset.

OCRA-STAN and OCRA-GOLD have a very similar distribution of sentence index deltas. The fact that OCRA-STAN predicts inter-sentence relations a little more frequently than OCRA-GOLD (3 absolute percentage points) might be caused by OCRA-STAN identifying wrong input entities in the same sentence if the syntactic parse is wrong.

4.5. Discussion

To summarize, we have found that — regarding the three metrics we focused on in this case study (trigger identification, action classification and input identification) — the syntax of the sentence seems to be important. The two pipelines that have the Gold syntactic parse available significantly outperform the others. This can be assumed even though we have insufficient training data for SLING and thus the comparison is not a fair one, since ocra's average F1-score on the three metrics improves by 0.06 when using the Gold syntactic parse instead of the StanfordNLP parser's. In addition, one cannot discard the possibility that SLING's latent representations do have some concept of syntax that help it identify actions and entities.

The reason why syntax seems to be so important might be that direct objects of a verb are almost always its input entities (patients) in imperative language such as the one found in cooking recipes. In addition, noun phrases and verb phrases are mostly the triggers of entities and actions in recipes. Thus, the heuristic script annotates actions and inputs correctly in almost all cases. We hypothesize however, that once a certain accuracy is reached by a parser, an emphasis on syntax is not beneficial anymore, because the more

complex relations in a cooking recipe (which the parser may still struggle with) cannot be captured on the sentence-level.

In general, until a considerably larger amount of training data for SLING or other machine learning methods is available, we conjecture that rule-based and syntax-based methods will bring better performance in the short term. While their capabilities might be limited to the simple cases that can be expressed in sentence-level syntax or rules in the lexicon, these are also the common cases with which one correctly parses most recipes.

Another option is alternative forms of supervision, which is discussed as a direction for future work in section 6.3. Transfer-learning, by pretraining a subset of SLING's parameters in an unsupervised fashion for example, is also a path one can explore to circumvent the need for large amounts of (expensive) annotated training samples.

5. Related Work

The work in this thesis touches two different areas: the kind of data (cooking recipes) and the task (semantic parsing). Cooking recipes belong to the larger category of procedural text. Related work both concerning the domain and the computational models will be examined in the following.

5.1. Domain and Problem Formulation

5.1.1. Cooking Recipes

Representing ingredient-instruction structure as trees Jermsurawong et al. [19] propose a shallow representation of cooking recipes. Given a list of all ingredients used in the recipe and the instructions, they predict which ingredients are consumed by which instruction. The resulting representation is a tree that contains ingredients as leaves and instructions as inner nodes. The limitation introduced by this representation is that ingredients can only be merged by an instruction, and not divided or kept independent. In our annotation, we have a `Divide` frame which splits up an ingredient into different parts (e.g., eggs into egg whites and egg yolks). Those kinds of actions cannot be represented with the tree representation proposed in [19].

This work can be seen, however, as a first step towards a deeper semantic representation of recipes, such as the one introduced in this thesis. The high-level flow of ingredients (which ingredients are merged in which instructions) may be used as a separate input to more complex models such as NPN [7], as explored below.

From trees to rooted DAGs Mori et al. [37] expand upon this idea of representing recipes as trees by instead representing them as rooted directed acyclic graphs (DAGs). The root of the graph is a special node that describes the final dish into which all actions lead. Also, they include not only ingredients and instructions as vertices, but also distinguish multiple actions in a sentence and take utensils into account. In contrast to [19], the edges are also labeled. The labels describe how the two nodes are related to each other. For example, an ingredient-to-action edge might be labeled as "d-obj". They annotate 266 Japanese recipes with rooted DAGs describing the flow of entities.

This graph representation can be seen as an extension to our annotation schema. The relations among entities and actions have more detailed types in their work. Also, they include relations such as which tools are equivalent, which tools are parts of each other or which foods are part of each other. While in their work, "the skin of the potato" would contain annotations for "skin" and "potato" and a "food part-of" relations between them, in our work we treat "skin of the potato" as one ingredient. For our downstream goal of

commonsense knowledge acquisition, having such a fine-grained representations as [37] makes the learning problem of semantic parsing more difficult, with little or no benefit for the resulting commonsense knowledge.

Understanding cooking recipes by tracking the world state Malmaud et al. [29] introduce a taxonomy of common linguistic characteristics found in cooking recipes and identify similar challenges as we found in our work. In contrast to our work, their method assumes a given list of all ingredients. They attempt to solve the problem of implicit actions and inter-sentence relations by tracking the world state throughout the execution of the recipe. The world state here is defined by a set of attributes of participating entities such as ingredients or containers. They propose an abstract probabilistic model that predicts symbolic actions which are applied to the current world state by a simple recipe simulator.

Bosselut et al. [7] build on this work and propose a concrete parser that tracks the world state explicitly. As part of their work, they release a hand-annotated dataset of 600 cooking recipes. Their annotation schema is geared towards the task of tracking the world state, as it contains information about a list of attributes like location, cleanliness or temperature for each ingredient after each recipe step. These attributes were an inspiration on the list of frames used in our annotation schema. As the dataset contains no explicit annotation for which trigger words in a sentence evoke the action frames we cannot build on their dataset.

5.1.2. Others

Annotation of lab protocols of biological experiments Kulkarni et al. [25] describe a semantic annotation effort on a corpus of lab protocols of biological experiments. These wet lab protocols are similar to cooking recipes in that they contain instructions on how to combine different substances and modify their state (e.g., heat the reagent). However, the language that is used is more complex and contains many technical terms. The annotation schema described in the paper inspired the annotation scheme for the recipe dataset created as part of this work. It is more complex, as it splits up compounds that describe an ingredient into separate entities such as measurement and reagent or meronyms. These are just treated as n-grams annotated as one `Entity` in our annotation schema. As lab protocols contain much more detailed instructions than cooking recipes and are less likely to leave out information, they represent not an ideal target to study with the goal of commonsense knownledge acquisition from gaps in the recipe. This is presumably the case because the language generally is more formal in lab protocols as it is written by experts with the goal of being able to reproduce an experiment exactly (which is not as important in cooking recipes).

The authors apply baseline action extraction and sequence tagger models to validate the feasibility of machine learning approaches on the task of semantic parsing of procedural texts. Because the task is similar, this makes exploring machine learning models for our task also promising, given enough training data.

5.2. Computational Models

5.2.1. Surface Structure Prediction

Predicting the ingredient-instruction structure of cooking recipes Jermsurawong et al. [19], in addition to the ingredient-instruction tree format, introduce a parser for the task of transforming recipes into the tree format. The problem of generating a tree describing the structure of the recipe is reduced to a binary classification problem. The parser consists of two binary Support Vector Machine classifiers that predict whether a given ingredient-instruction or instruction-instruction edge is valid or not. The features used for the classification consist of various lexical features, as well as n-grams, binary term vectors and the number of ingredients already linked to a given instruction. While the representation the parser outputs is too shallow for our use case, their method achieves an edge prediction accuracy of 93.5%, which makes it interesting to use as an additional input to more sophisticated parsers that output a deeper representation of recipes.

A framework for procedural text understanding Maeta et al. [28] propose a pipeline parser approach for the dataset introduced by Mori et al. [37]. In a first step, the sentences are segmented into words (which is relatively trivial in European languages, but harder for Japanese, which is the language used in this corpus). The second stage is a conditional random field (CRF) that identifies concepts like actions and ingredients and is framed as a named entity recognition (NER) problem. The final step is to parse the full text and predict the rooted DAG representation introduced in [37]. Maeta et al. [28] use a minimal spanning tree (MST) parser that predicts the arc labels. In their experiments, the authors argue that the pipeline approach introduces too many errors at each state. These errors compound and limits the accuracy improvements achievable by optimizing only one component, motivating future work on an integrated solution that goes directly from text to semantic representation. This direction is adressed by the SLING parser [41], evaluated as part of this work, as well as the NPN parser [7], discussed below.

5.2.2. Tracking the World State

In addition to a dataset, Bosselut et al. [7] also introduce a semantic parser trained and evaluated on this dataset. The parser, called a neural process network (NPN), is a recurrent memory network [47] which learns to update entity states using learned action embeddings. First, a sentence is read using a gated recurrent unit (GRU) [9]. The final hidden state is used to predict (a) which actions take place in the sentence and (b) which entities participate in the sentence. Based on the predicted actions, an action embedding for this sentence is constructed and applied to the entities which are stored in the network's memory. A set of attribute extractors predict which value an attribute of a given entity has after each sentence. As annotated training data is expensive, the network is trained with weak supervision using a dataset which was heuristically annotated with a few hand-designed rules. Thus, it lacks annotations for implicitly mentioned ingredients. A big limitation is that the NPN needs to predict all participating entities for each sentence, even if the ingredients have been merged in a previous step. The evaluation is done on the

smaller, hand-annotated 600-recipe dataset, and shows especially low scores for the entity selection task for combined entites (F1 score of 0.21). This specific task is approached more successfully in [19], which is why a combination of these two methods is a compelling direction for future work. This is further discussed in section 6.5.

6. Future Work

Based on our investigation, we propose several directions for future work on semantic parsing for cooking recipes.

6.1. Inter-sentence Relations

As the annotation of our study 50-recipe dataset revealed, cooking recipes can only be understood fully if the parser can take into account not only the current sentence, but maintains a context of the whole recipe. SLING [41], one of the parsers evaluated as part of this work, does not support such inter-sentence relations off the shelf. Thus, we treated one recipe as one sentence when training and evaluating it. The general architecture of SLING, however, would theoretically allow inter-sentence relations. If the frame store (attention buffer) was not cleared after each sentence was read, and the sentences of a recipe were attached to each other and read by the parser in sequence, it could predict relations between frames evoked in different sentences. The authors of SLING have told us that they are working on a new version that allows this.

Another approach to handle the sentence-spanning context in cooking recipes is to explicitly track the world state, as methologies like NPN [7] do.

One area for future work is to develop another (third) approach to handling inter-sentence context. One could imagine, for example, a transformer network [45] encoding the whole recipe at once and then building a frame graph out of the obtained contextualized recipe embedding (instead of the LSTM [16], which is used by SLING), using a similar transition predictor.

6.2. Representation Design

The design of the representation is a core task of semantic parsing. In this work we have identified several challenges concerning representation design, and solved some of them with our own annotation schema.

One of the remaining challenges are temporal relations. Momouchi [36] presents a taxonomy of control structures that one could adapt for the recipe domain and adjust our annotation schema such that temporal control structures can be represented in more detail. It remains questionable, and worthy of future study, however, whether a deeper representation of temporal relations leads to a better bottom line accuracy regarding commonsense knowledge acquisition.

6.2.1. Automatic Representation Design

In this work, we manually designed a representation schema that accomodates the phenomena commonly encountered in cooking recipes (a relatively limited domain). Eventually, our goal is to be able to extract commonsense knowledge from all categories of text. Manually designing a representation for each new domain or broad domains appears to be infeasible. Thus, one direction for future work is to experiment with the automatic construction of frame lexicons, which can then be used to train a semantic parser. This process can be seen, interestingly, as another kind of commonsense knowledge acquisition.

6.3. Training with Weak Supervision

Creating the small study dataset of 50 recipes in this thesis has already been a considerable manual effort, to meaningfully train large machine learning models like SLING [41] or NPN [7] however, much more is required. Such annotation is costly. Thus, one direction for future work is to explore training models with different kinds of supervision signals (other than manual token-by-token annotations).

So-called denotation-based training is a common method to train semantic parsers in general [21]. The idea is not to calculate the error based on the representation the semantic parser produces directly, but to measure the accuracy using a downstream result of that representation (for which the correct result might be easier to acquire than the semantic representation of the parser). If the task is to predict SQL queries from a natural language text, a straightforward measure of accuracy is to compare the result of running the query on a database with the desired result.

Another type of such training is to frame semantic parser as a question answering task. Rather than output a specific semantic representation for an input text (for which labels are expensive to obtain), the parser needs to be able to answer questions about the input [6].

We are not aware of any such work that addresses the cooking recipe domain or uses a similar annotation schema as ours. While if the targeted semantic representation is an SQL query, comparing the result of running the query is a straightforward supervision signal, such types of weakly supervised approaches are not as obvious in the cooking recipe domain. The only option of how to score semantic parses without annotated training samples we could think of is to give parses that use all ingredients given in the ingredient list of the recipe a higher score. Future work in this direction might find enough such signals to be able to train a semantic parser for cooking recipes based solely on weak supervision.

6.3.1. Additional Inputs

A related area for future work is to take additional inputs into account. Especially videos or photos of a person performing the recipe, or of the ingredients in the kitchen might be incorporated as an additional input to the parser. Not all datasets of recipes include such information, but work on the more general domain of how-to texts has datasets that would allow such research [34].

Notably, using a video or photo for each sentence of the recipe, one can solve another challenge we identified in this thesis. The imperative language used in cooking recipes causes the performance of general-purpose syntactic parsers to be considerably lower than on newspaper text (see 4.2.3.1). A well-trained image or video captioning model, applied to the photo or video associated to each sentence can generate a natural language sentence that has a subject and a less informal language in general. Running the syntactic and/or semantic parser on those captions might reduce the difficulty of the parsing task and improve overall accuracy.

6.4. The How-to Domain

Another direction for future work is to expand our research to the broader domain of how-to guides. This domain offers a broader vocabulary, and — supposedly — more frames and more types of slots (not only ingredients, locations, utensils and conditions). Miech et al. [34] present a dataset of how-to videos paired with narrative captions in many different categories, only one of which is cooking. One could build on their work, perhaps, by studying the narrative captions in their dataset, treating the captions for each video as one "recipe". The videos themselves could also be utilized as an additional input for a multimodal how-to guide semantic parser.

6.5. Model Ensemble

Semantic parsing is a complex task that entails multiple subtasks such as trigger identification, action classification, ingredient prediction and/or input identification (depending on the way the problem is framed). Especially given the lack (and expensiveness) of large amounts of manually annotated training samples, end-to-end approaches like SLING [41] or NPN [7] may not be the most fruitful research direction.

Instead, ensembling multiple specialized models, each one optimized for a particular subtask, might be a more promising area for future work. One such combination has already been mentioned in section 5.1.1. It addresses the difficulty NPN has with predicting all participating ingredients for each steps, especially the ones that have been combined in earlier steps. One could try using the SIMMR parser [19], which achieves a very high accuracy in building an ingredient-instruction tree (i.e., which ingredients participate in each step), for this subtask instead. One could then retrain NPN, taking the participating ingredients as an additional input and then only having to predict the correct actions applied to those ingredients instead, which might improve the overall parsing accuracy considerably.

ocra (described in section 4.3.1) is another example of such a combined model. It uses the StanfordNLP dependency parser [31] to get a syntactic parse, which is then further processed by its own rules.

Other combinations of specialized models are conceivable, which is one research direction to further pursue that might mitigate the need for large amounts of manually annotated training samples.

6.6. Commonsense Gaps vs. Parser Errors

Finally, a very general challenge affecting our work with respect to the downstream task of commonsense knowledge acquisition is how one can distinguish between errors of the parser and actual commonsense gaps.

In our work, a gap is defined by a mismatch between a post-condition and a pre-condition for some entity. This gap could be caused by the recipe itself (i.e., some commonsense information has been left out, and this should be recorded) or it could be caused by an error of the parser (e.g., the frame has been identified incorrectly or some slot has been filled with the wrong entity).

In conclusion, this means that only if the parser has a near-perfect accuracy, one can trust the commonsense knowledge it extracts. Another approach, rather than working on improving the parser's accuracy, might be to develop a post-processing algorithm that can distinguish between parser-error gaps and actual commonsense gaps. Frequency analysis to quantify the commonsense-ness of some piece of information, is one feature such an algorithm could utilize.

7. Conclusion

In conclusion, we have investigated the problem of semantic parsing in the cooking recipe domain with the downstream task of commonsense knowledge acquisition in mind.

Our annotation schema is able to represent a cooking recipe in a way that allows us to extract commonsense knowledge and quantify the commonsense-ness of a piece of information.

We have annotated a small dataset of 50 English recipes, which has allowed us to study syntactic characteristics such as implicit objects, imperative sentences and omissions that one commonly encounters in this domain. We found that state of the art dependency parsers and POS-taggers yield considerably more errors in this domain than they do in texts like news articles.

We have defined three metrics — trigger identification, action classification and input identification — that we have used to evaluate four different semantic parsing pipelines on our recipe dataset. We have found that syntax-based methologies outperform other methods considerably, and hypothesized that this is due to the imperative language found in cooking recipes: the object of a verb is the input of the action it represents in most cases.

We have analyzed the four parsing pipelines' performance regarding a central issue in understanding cooking recipes: inter-sentence relations. As such relations where the input of an action comes from an earlier sentence are frequent in cooking recipes, a semantic parser must be able to reliably identify the correct input of an action, even if comes from an earlier sentence and the relation is not stated explicitly.

Based on our investigation, we have proposed several directions for future research in this field, in areas we believe will improve the parsing accuracy, which in turns allows gap analysis and frequency analysis to extract commonsense knowledge more accurately.

When we are able to automatically extract commonsense knowledge from open-domain texts, we can eventually build computers that are able to deeply understand natural language. As language is so central in our lives, this will make computers a lot more useful.

Bibliography

[1] Ron Artstein and Massimo Poesio. "Inter-coder agreement for computational linguistics". In: *Computational Linguistics* 34.4 (2008), pp. 555–596.

[2] Yoav Artzi and Luke Zettlemoyer. "Weakly supervised learning of semantic parsers for mapping instructions to actions". In: *Transactions of the Association for Computational Linguistics* 1 (2013), pp. 49–62.

[3] Collin F Baker, Charles J Fillmore, and John B Lowe. "The berkeley framenet project". In: *Proceedings of the 17th international conference on Computational linguistics-Volume 1*. Association for Computational Linguistics. 1998, pp. 86–90.

[4] Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[5] Yoshua Bengio et al. "A neural probabilistic language model". In: *Journal of machine learning research* 3.Feb (2003), pp. 1137–1155.

[6] Jonathan Berant et al. "Semantic parsing on freebase from question-answer pairs". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. 2013, pp. 1533–1544.

[7] Antoine Bosselut et al. "Simulating action dynamics with neural process networks". In: *arXiv preprint arXiv:1711.05313* (2017).

[8] Danqi Chen and Christopher Manning. "A fast and accurate dependency parser using neural networks". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 740–750.

[9] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[10] Philipp Cimiano, Christina Unger, and John McCrae. "Ontology-based interpretation of natural language". In: *Synthesis Lectures on Human Language Technologies* 7.2 (2014), pp. 1–178.

[11] Michael A Covington. "A fundamental algorithm for dependency parsing". In: *Proceedings of the 39th annual ACM southeast conference*. Citeseer. 2001, pp. 95–102.

[12] Li Dong and Mirella Lapata. "Language to logical form with neural attention". In: *arXiv preprint arXiv:1601.01280* (2016).

[13] Charles J Fillmore and Collin F Baker. "Frame semantics for text understanding". In: *Proceedings of WordNet and Other Lexical Resources Workshop, NAACL*. 2001.

[14] Daniel Gildea and Daniel Jurafsky. "Automatic labeling of semantic roles". In: *Computational linguistics* 28.3 (2002), pp. 245–288.

[15] Graeme Hirst. *Semantic interpretation and the resolution of ambiguity*. Cambridge University Press, 1992.

[16] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[17] Eduard Hovy et al. "OntoNotes: the 90% solution". In: *Proceedings of the human language technology conference of the NAACL, Companion Volume: Short Papers*. 2006, pp. 57–60.

[18] CT Huang. "On the typology of zero anaphora". In: (1984).

[19] Jermsak Jermsurawong and Nizar Habash. "Predicting the structure of cooking recipes". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 2015, pp. 781–786.

[20] Dan Jurafsky. *Speech & language processing*. accessed 10/08/2019. 2018. URL: https://web.stanford.edu/~jurafsky/slp3/.

[21] Aishwarya Kamath and Rajarshi Das. "A Survey on Semantic Parsing". In: *arXiv preprint arXiv:1812.00978* (2018).

[22] Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. "Globally coherent text generation with neural checklist models". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 2016, pp. 329–339.

[23] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[24] Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. "Neural semantic parsing with type constraints for semi-structured tables". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2017, pp. 1516–1526.

[25] Chaitanya Kulkarni et al. "An annotated corpus for machine reading of instructions in wet lab protocols". In: *arXiv preprint arXiv:1805.00195* (2018).

[26] Percy Liang. "Lambda dependency-based compositional semantics". In: *arXiv preprint arXiv:1309.4408* (2013).

[27] Percy Liang. "Learning executable semantic parsers for natural language understanding". In: *arXiv preprint arXiv:1603.06677* (2016).

[28] Hirokuni Maeta, Tetsuro Sasada, and Shinsuke Mori. "A framework for procedural text understanding". In: *Proceedings of the 14th International Conference on Parsing Technologies*. 2015, pp. 50–60.

[29] Jonathan Malmaud et al. "Cooking with Semantics". In: *Proceedings of the ACL 2014 Workshop on Semantic Parsing*. Baltimore, MD: Association for Computational Linguistics, 2014, pp. 33–38. DOI: 10.3115/v1/W14-2407.

[30] Christopher D Manning, Christopher D Manning, and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[31] Christopher Manning et al. “The Stanford CoreNLP natural language processing toolkit”. In: *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*. 2014, pp. 55–60.

[32] Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. “Building a large annotated corpus of English: The Penn Treebank”. In: (1993).

[33] Deborah L McGuinness, Frank Van Harmelen, et al. “OWL web ontology language overview”. In: *W3C recommendation* 10.10 (2004), p. 2004.

[34] Antoine Miech et al. “HowTo100M: Learning a Text-Video Embedding by Watching Hundred Million Narrated Video Clips”. In: *arXiv preprint arXiv:1906.03327* (2019).

[35] Bhavana Dalvi Mishra et al. “Tracking state changes in procedural text: a challenge dataset and models for process paragraph comprehension”. In: *arXiv preprint arXiv:1805.06975* (2018).

[36] Yoshio Momouchi. “Control structures for actions in procedural texts and pt-chart”. In: *COLING 1980 Volume 1: The 8th International Conference on Computational Linguistics*. 1980.

[37] Shinsuke Mori et al. “Flow Graph Corpus from Recipe Texts.” In: *LREC*. 2014, pp. 2370–2377.

[38] Joakim Nivre et al. “Universal dependencies v1: A multilingual treebank collection”. In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*. 2016, pp. 1659–1666.

[39] Martha Palmer, Daniel Gildea, and Paul Kingsbury. “The proposition bank: An annotated corpus of semantic roles”. In: *Computational linguistics* 31.1 (2005), pp. 71–106.

[40] Lance A Ramshaw and Mitchell P Marcus. “Text chunking using transformation-based learning”. In: *Natural language processing using very large corpora*. Springer, 1999, pp. 157–176.

[41] Michael Ringgaard, Rahul Gupta, and Fernando CN Pereira. “SLING: A framework for frame semantic parsing”. In: *arXiv preprint arXiv:1710.07032* (2017).

[42] Pontus Stenetorp et al. “BRAT: a web-based tool for NLP-assisted text annotation”. In: *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics. 2012, pp. 102–107.

[43] Dan Tasse and Noah A Smith. “SOUR CREAM: Toward semantic processing of recipes”. In: *Carnegie Mellon University, Pittsburgh, Tech. Rep. CMU-LTI-08-005* (2008).

[44] Kristina Toutanova et al. “Feature-rich part-of-speech tagging with a cyclic dependency network”. In: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*. Association for computational Linguistics. 2003, pp. 173–180.

[45] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[46] Joseph Weizenbaum et al. “ELIZA—a computer program for the study of natural language communication between man and machine”. In: *Communications of the ACM* 9.1 (1966), pp. 36–45.

[47] Jason Weston, Sumit Chopra, and Antoine Bordes. “Memory networks”. In: *arXiv preprint arXiv:1410.3916* (2014).

[48] Liang-Jun Zang et al. “A survey of commonsense knowledge acquisition”. In: *Journal of Computer Science and Technology* 28.4 (2013), pp. 689–719.

[49] Junru Zhou and Hai Zhao. “Head-driven phrase structure grammar parsing on Penn Treebank”. In: *arXiv preprint arXiv:1907.02684* (2019).

A. Frame Types

A.1. List of Frame Types

We annotated our small 50-recipe dataset with the annotation scheme described in section 4.1.2. This appendix lists the final set of frames we decided upon after adjusting the initial set of frames (which are marked in **bold**).

Frame name	Definition	Example
`CleanlinessChange`	wash or dirty something	Wash vegetables well.
`Continue`	continue doing something (no change)	Let stand for 1 hour.
`Divide`	divide one thing into two or more individually nameable things	Remove any fat from the beef.
`LocationChange`	move something from somewhere to somewhere else	Slide the omelet onto a plate.
`MatterStateChange`	change the state of matter of something by melting, freezing, etc.	Melt butter.
`Merge`	merge two or more ingredients such that they become indistinguishable	Mix eggs, salt and pepper.
`ShapeChange`	change the optical shape of something	Form small rings with dough.
`SizeChange`	change the size of something by cutting it etc.(afterwards, the pieces are not individually nameable)	Cut tomatoes into small cubes.
`TemperatureChange`	heat or cool something	Heat butter.
`WetnessChange`	wet or dry something (essentially a `LocationChange` of water)	Drain water with a towel.

A.2. Disambiguations

Natural language is inherently ambiguous. Although we feel like the frames above are relatively atomic, sometimes it is not clear which of the frames to assign. We have devised the following disambiguations which help mitigate this issue.

A.2.1. `Divide` vs. `SizeChange`

If something is split up into mutiple parts and the parts can be individually named, it is a `Divide` action. Example: "Crack eggs into a bowl." This action splits up the eggs into egg whites and egg yolks, which can be individually named.

If something is split up into multiple parts but the parts cannot be individually named, such as when one cuts something into smaller pieces, it is a `SizeChange` action. Example: "Cut carrots into longish pieces." The carrot pieces afterwards cannot be individually named.

A.2.2. `Merge` vs. `LocationChange`

Ambiguous sentences:

1. "Put the mixture into the bowl."
2. "Put salt into the mixture."

If the destination (underlined in each sentence) is a utensil (something inedible), the sentence describes a `LocationChange` action. Thus, the first sentence evokes a `LocationChange` frame ("the bowl" is inedible).

If the destination is instead an ingredient (something edible), the sentence describes a `Merge` action. Thus, the second sentence evokes a `Merge` frame ("the mixture" is edible).

B. Common Format

B.1. Motivation

To measure and compare the performance of ocra and SLING on trigger identification, action classification and input identification in our dataset (the metrics defined in section 4.4.2.1), a common output format is required. While the Gold labels use brat's standoff format [42], ocra outputs a JSON file per recipe that contains all frames including their arguments, and SLING creates a special frame object containing an annotated recipe. We developed a common format and wrote converters for each of the three output formats. The common format is designed with the three performance metrics in mind. We thus focus on actions and inputs, and discard location and condition annotations (which are still part of the annotation schema).

B.2. Description of the Common Format

There is one plain text file per recipe that contains two different types of lines that represent action frames and their input and utensil entities:

1. **Frame lines** describe an annotation of a sequence of tokens (which could be an action frame or an `Entity`). Schema:

   `T<unique_index><tab><sentence_index><tab><frame_type><tab><from>:<to>`

   The `T` along with the unique index forms the `id` of the annotation, which is used to reference it in link lines (see below). The frame type can either be one of the frame names listed in Appendix A or one of `entity` and `utensil`. `from` and `to` are token indices that give the span that evokes this frame.

2. **Link lines** describe a relation between two different annotations. Schema:

   `L<tab><from_frame_id>:<to_frame_id>`

   `from_frame_id` and `to_frame_id` are frame ids in the form `T<index>` (coming from frame lines) that describe the head and dependent of the relation, respectively.