MASTER'S THESIS

FB3 Informatik

Scaling Probabilistic Completion of Robot Instructions through Semantic Information Retrieval

Skalierung von probabilistischen Modellen zur Vervollständigung von Roboterinstruktionen durch semantische Informationsrückgewinnung

Author:	Sebastian Koralewski
Advisors:	Prof. Michael Beetz Ph.D.
	Prof. Dr. Thomas Schneider
Supervisor:	Daniel Nyga
Submission Date	January 10, 2017

Universität Bremen

Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Bremen, January 10, 2017

(signature)

Abstract

The idea of having domestic robots or computer systems assisting us in our everyday work exist for ages. An ideal implementation of this idea is to instruct the agents via natural-language voice commands. However, natural-language instructions can be ambiguous and incomplete. A promising solution to interpret such instructions is the usage of joint probability distributions. With these distributions it is possible to infer semantic information from a sentence like "flavor a chicken with pepper". The inferred semantic information represents that the given instruction describes a 'Flavoring' action in which the entity "chicken" has to be seasoned with the spice "pepper". Probability distributions allow us also to capture relational knowledge between entities such as that sugar is suitable to season pancakes. This relational knowledge enables us to infer missing information for incomplete tasks where e.g. the spice is missing in a sentence like "season the pancake". One possibility to obtain such relation knowledge is to extract information from natural-language documents. Unfortunately, the complete knowledge of a corpus cannot be represented in joint probability distribution since the increase of random variables causes that the inference process becomes intractable. To overcome this scaling problem, we present in this thesis an information extraction system which extracts semantic information from the sentences contained in natural-language documents. This extracted information is stored in a knowledge base which allows us to retrieve missing information based on the semantic similarity between a given incomplete instruction and the semantically indexed sentences. We evaluate our performance of our information extraction system and semantic information retrieval algorithm on a large collection of natural-language documents from the internet.

ZUSAMMENFASSUNG

Die Idee, das uns Haushaltsroboter oder Computersysteme in unserem Alltag unterstützen existiert schon seit Jahren. Eine erfolgreiche Implementierung dieser Idee erlaubt die Anweisung der Systeme anhand von natürlicher Sprache. Natürlichsprachige Anweisungen können jedoch mehrdeutig und fehlerhaft sein. Ein viel versprechender Ansatz, um solche Anweisungen zu interpretieren, ist die Verwendung von Wahrscheinlichkeitsverteilungen. Mit diesen Verteilungen ist es möglich semantisches Wissen aus einem Satz wie "würze das Hähnchen mit Pfeffer" zu inferieren. Dieses inferierte semantische Wissen repräsentiert, dass die Anweisung eine "Würzungs"-Handlung darstellt, in der das Objekt "Hähnchen" mit dem Objekt "Pfeffer" gewürzt werden soll. Wahrscheinlichkeitsverteilungen erlauben uns relationales Wissen zwischen Objekten zu repräsentieren. Dieses Wissen kann z.B. darstellen, dass Zucker geeignet ist, um ein Pfannkuchen zu würzen. Diese Relation kann dazu genutzt werden, um unvollständige Anweisungen wie "würze die Pfannkuchen" zu vervollständigen. Eine Möglichkeit, um relationales Wissen zu erhalten ist die Extraktion von Wissen aus natürlichsprachigen Dokumenten. Leider kann das vollständige Wissen aus solchen Dokumenten nicht in einer Wahrscheinlichkeitsverteilung repräsentiert werden. Jede neue Information, die in einer Verteilung repräsentiert werden muss, erhöht die Anzahl der Zufallsvariablen in dieser Verteilung. Dadurch werden die Inferenzprozesse praktisch nicht mehr ausführbar. Um dieses Skalierungsproblem zu umgehen präsentieren wir in dieser Arbeit ein Informationsextraktion System, welches im Stande ist semantische Informationen aus natürlichsprachigen Dokumenten zu extrahieren und diese in einer Wissensdatenbank abzulegen. Die Wissensdatenbank ermöglicht uns das Vervollständigen von unvollständigen Anweisungen mit Hilfe von semantischen Relationen zwischen Anweisungen und Sätzen in der Datenbank. Wir evaluieren unser Informationsextraktion System und unseren semantischen Informationsrückgewinnungsansatz auf eine große Anzahl an natürlichsprachigen Dokumenten aus dem Internet.

Contents

List of Figures

List of Tables

CHAPTER one

INTRODUCTION

1.1 MOTIVATION

The idea of having domestic robots or computer systems assisting us in our everyday work exist for ages. An ideal implementation of this idea is to instruct the agents via natural-language voice commands. Today, with the existence of intelligent personal assistants such as Siri[1] and Google Now[2], there is a part of the idea accessible to the public. Those assistants are able to handle instructions such as "call Jane Doe". Based on this instruction, these systems execute a "call"-plan with Jane Doe as the required parameter. However, "call my girlfriend" is not quite as simple. The system has to infer who the user's girlfriend is. But at the end, the same call-plan will be executed. In this case, as long the semantic and syntax remains the same, the "call"-command will always be mapped to the same plan.

Instructions which are given to domestic robots can be more complexed to execute. For instance, consider the instructions "fill a cup with water" and "fill a mug with coffee". They have the same semantic and syntactic structure. However, they differ in the execution. To fill water into a cup, the agent can use a tap. To fill some coffee into a mug, the robot has to find a container which contains coffee. Then, it has to grab this container and pour the content into the mug. This scenario shows that a simple mapping of natural-language instructions to executable plans is not the best solution.

A promising solution to determine a reasonable plan is to utilize a joint probability distribution over actions and objects. In our example, the action could be 'Filling' and the objects are water and coffee. So the joint probability distribution can represent something like: If it is a filling action and water has to be poured, it is more likely to use a tap, and for coffee it would be most likely to use a coffee pot. More formally, it is possible to represent the plan inference as following [26]:

[1]http://www.apple.com/uk/ios/siri/
[2]https://www.google.com/search/about/learn-more/now/

$$\underset{plan}{\operatorname{argmax}} P(\text{intended(plan)}|\text{natural-language instruction})$$

An additional advantage of having a joint distribution probability is that it can be used to automatically complete instructions by inferring their most probable value from the distribution. For instance, consider the sentence "neutralize hydrochloric acid". This sentence is missing the corresponding base to neutralize the given acid. Having a distribution over the action 'Neutralizing', it is possible to infer the missing base.

Probabilistic Action Cores (PRAC)[25][26] is a framework for learning of and reasoning about such action-specific joint probability distributions. These distributions can be trained by hand-labeled instructions. For instance, for the action 'Filling' we can provide instructions such as "fill a wineglass with wine" and "fill a glass with water". These sentences can be labeled with the corresponding plans and plan parameters such as the first sentence requires a "pouring"-plan with "wine" and "wineglass" as the parameters ***theme*** and ***destination***. Additionally, the words are annotated with senses. These senses are symbolic links which refer to the semantic definition of a word. Additionally, these senses are defined in an underlying taxonomy. So it is possible to link the sense of the word "wine" to the sense which represents "alcoholic beverage", for instance. With the senses and joint probability distributions, PRAC differentiates the semantic e.g. of the word "cup" in the sentences "fill a cup with water" and "add a cup of sugar to the mixture". In the first sentence "cup" represents a container to store liquids and in the second sentence it represents a measure unit.

Based on the listed advantages of having a joint probability distribution, this approach seems to be the solution for implementing intelligent robotic agents. Unfortunately, large distributions are not tractable. Performing exact inference in probability distribution is NP-hard [30]. The running time depends on the amount of random variables. So referring back to PRAC, every new training sentence will increase the complexity to perform inference tasks on that distribution, since every new object represents a new random variable. If every random variable is represented as a binary variable, there would be 2^n possible combinations (these combinations are also called worlds), where n is the number of variables, which have to be evaluated. In other words, a joint probability distribution does not scale well on an increase of data.

To avoid this increase of complexity, we have to keep the distributions small. PRAC uses a probabilistic model which utilizes taxonomic knowledge to keep the number of possible worlds small. The benefit of this approach is that the mentioned model gets a training sentence such as "fill a wineglass with wine". Given an unseen instruction such as "fill a glass with whisky", this model is able to infer the correct plan and senses based on the similarities between "glass" to "wineglass" and "whisky" to "wine". This taxonomic knowledge decreases the complexity and the model is still able to infer reasonable results.

An abstract model is suitable to infer the senses and correct plans but it is not feasible for completing robot instructions. Consider a joint probability distribution for a 'Storing' action. This distribution represents that food can be stored in a fridge. In general, it is a correct model but there are some foods which do not have to be stored in a fridge. For instance, milk has to be stored in a refrigerator since it would perish. However, potatoes should be kept in dark and dry places rather than in a fridge. So to be able to infer the correct missing information for each possible scenario, we have to extend the distribution with additional training data. At the end, it will improve the missing information inference but the number of random variables will increase too.

This conclusion requires a solution which does not use a joint probability distribution but is still able to infer reasonable missing information to complete robot instructions. With reasonable results we mean something like that it makes more sense to use sugar as a spice to season a pancake instead of pepper. A possible solution can be oriented by the behavior how people look for something they do not know. They look up new information in books, documents or on websites such as Wikipedia[3] and WikiHow[4]. The advantages of a look up implementation are that it does not require a joint probability distribution and it can be performed in linear time.

There are many possibilities to implement such a solution. For instance, there is the solution to provide a corpus just in plain text. If the system gets the task "flavor the pancake", it can try to look up for sentences like "flavor the pancake with sugar". Unfortunately, this solution will only provide good results if the text passages in the corpus match exactly the query. For example, the sentences "flavor a pancake with sugar" or "flavor the golden brown pancakes with sugar" will not be captured by the search query. Also, if a corpus contains the sentences "flavor a waffle with sugar" and "season a steak with pepper" the solution would not provide any results too. In conclusion, a syntactic and text-based search will not provide the requested results to retrieve missing information.

This motivates to design a solution where the knowledge in natural-language documents can be stored in a semantic knowledge representation. This would allow the system to send queries in a more sufficient way. So instead of sending a query in plain text, the system could send a query like "which spice is suitable to season a pancake". In the provided knowledge base, a sentence like "flavor the pancake with sugar" can be annotated as a 'Flavoring' action with "pancake" as object to be seasoned and "sugar" as a spice. This annotation task can be done by an information extraction system which uses the joint probability distributions from PRAC. With this knowledge representation and in combination of taxonomic knowledge about the objects, it is possible to query for sentences in the corpus which are semantically similar to the given incomplete task. More formally, it is possible to represent this kind of inference as following:

[3]https://en.wikipedia.org/wiki/Main_Page
[4]http://www.wikihow.com/Main-Page

$$\underset{sentence \in corpus}{\text{argmax}} \; Similarity(\text{sentence, uncompleted natural-language instruction})$$

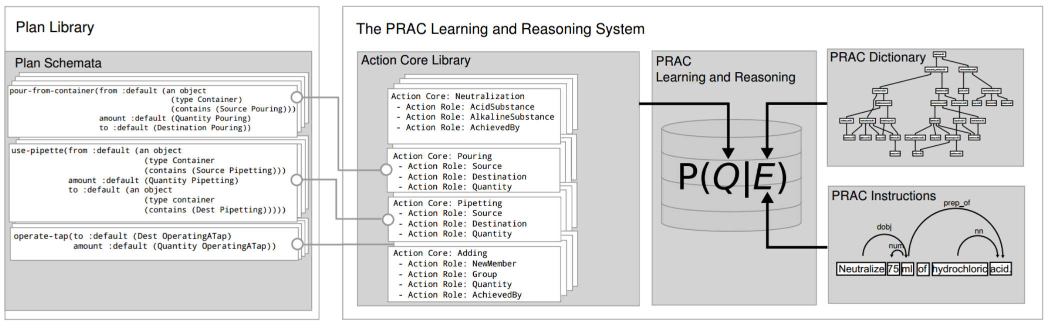
Consider the task "season a pancake" and a corpus which includes the two sentences "flavor a waffle with sugar" and "flavor a steak with pepper". With the taxonomic knowledge the system would be able to infer that "sugar" is more reasonable to be used as a spice for a pancake, since "pancake" and "waffle" are more similar to each other than "pancake" to "steak".

In other words, we avoid solving the missing information inference via a joint probability distribution and use a semantic information retrieval approach instead. We call this approach "semantic information retrieval" because it differs from the classical information retrieval [28]. The classical information retrieval is focused on finding information keyword based [11]. So the sentences "season a turkey", "season a chicken" and "season a pancake" would have the same similarity since the word "season" is contained in all these sentences and there is no relation between these objects. If we add some semantic information on these sentences, such as adding senses to the entities, we can specify a more detailed similarity. With a semantic information retrieval approach, the system is able to recognize that "turkey" is more similar to "chicken" than to "pancake". More general, with semantic information retrieval we are able to overcome the scaling problem of probabilistic models but are still able to query for reasonable results to complete robot instructions.

In this thesis we present a solution to extract and represent semantic knowledge. In addition, we extend PRAC so it can perform a missing information inference through semantic information retrieval.

1.2 PROBABILISTIC ACTION CORES

As mentioned in Section 1.1, Probabilistic Action Cores (PRAC) is a framework for learning of and reasoning about action-specific joint probability distributions. This framework is able to infer a plan from a sentence such as "fill a cup with water" which can be executed on a robotic agent.



```
pour-from-container(from :default (an object
                                   (type Container)
                                   (contains (Source Pouring)))
                    amount :default (Quantity Pouring)
                    to :default (Destination Pouring))
```

```
use-pipette(from :default (an object
                           (type Container
                           (contains (Source Pipetting)))
            amount :default (Quantity Pipetting)
            to :default (an object
                           (type container
                           (contains (Dest Pipetting)))))
```

```
operate-tap(to :default (Dest OperatingATap)
            amount :default (Quantity OperatingATap))
```

```
Action Core: Neutralization
 - Action Role: AcidSubstance
 - Action Role: AlkalineSubstance
 - Action Role: AchievedBy
```

```
Action Core: Pouring
 - Action Role: Source
 - Action Role: Destination
 - Action Role: Quantity
```

```
Action Core: Pipetting
 - Action Role: Source
 - Action Role: Destination
 - Action Role: Quantity
```

```
Action Core: Adding
 - Action Role: NewMember
 - Action Role: Group
 - Action Role: Quantity
 - Action Role: AchievedBy
```

Figure 1.1: PRAC Framework [26]

The concept of the framework is depicted in Figure 1.1. The framework consists of three components: **PRAC dictionary**, **PRAC knowledge base** and **PRAC plan library** [26].

The **PRAC dictionary** contains all possible meanings of all the words that can occur in natural language. For instance, the word 'bark' can represent the sound made by a dog or the covering of a tree. The meanings are called concepts and are represented in a taxonomy. So with the knowledge representation it is possible to determine that "orange juice" and "drinking water" are similar since they have an is-a relation to the concept "beverage". In its current state, PRAC uses the dictionary of WordNet[5] [6] (a more detailed description is given in Section 2.4.2). Using this dictionary, the framework is able to infer the semantics for the words in the given natural-language instruction.

The **PRAC knowledge base** contains a collection of action-verb-specific knowledge bases which are called action cores. These action cores are abstract definitions of actions. For example, a 'Pouring' action can be described as: Filling the content of a source to a destination. In PRAC, such a description is defined as a collection of predicates. The action 'Pouring' can be represented by the predicates $action\_core(v, Pouring)$, $action\_verb(v, Pouring)$, $theme(o_1, Pouring)$, $source(o_2, Pouring)$ and $destination(o_3, Pouring)$, where v is the word which activates the action and o_n represents the required objects to execute this action. The predicates which are representing the required parameters are called action roles. To map these objects to the concepts of the PRAC dictionary, we can use the has_sense predicate. Figure 1.2 shows the sentence "pour some water into a cup" represented as atoms. For instance, $has\_sense(water, water.n.06)$ states that the word "water" in this sentence represents the meaning of the 6th concept of the noun water which means drinking water. In this context, we say that the word "water" has the sense water.n.06. A more detailed description about the notation of these concepts is given in Section 2.4.2. So when we speak of inferring the senses, we mean that PRAC asserts the most likely concepts to the words in the given natural-language instruction. The task of inferring the correct senses is known as word-sense disambiguation problem (see Section 2.1.2 for more additional information).

```
action_core(Pour,  Pouring)
action_verb(Pour,Pouring)
theme(water,Pouring)
destination(cup,Pouring)
has_sense(Pour,pour.v.01)
has_sense(water,water.n.06)
has_sense(cup,cup.n.01)
```

Figure 1.2: 'Pour some water into a cup' represented as atoms

This atom representation can be used to train these action-specific joint probability distributions or it can be the result of an inference. The mentioned distributions are called probabilistic action core (PRAC). Using these distributions, the framework

[5]http://wordnet.princeton.edu

infers the most likely plan from the **PRAC plan library**. For our pouring example, it makes more sense to execute the "operate-tap" plan instead the "pour-from-a-container" plan. This library contains plan templates whose parameters will be asserted with the inferred action roles. These plans are designed by people with the intention to let the robotic agent perform specific tasks. For instance, if the robot operates in a kitchen, it is required that the plan library contains plans which allow the robot to perform kitchen-specific actions such as preheating the oven or cutting vegetables.

1.2.1 PRAC Reasoning Process

This section gives an overview about the reasoning process of the PRAC framework to infer an executable plan.

However, a more detailed treatment of reasoning, such as the actual design of the joint probabilities distributions and which probabilistic model is used, can be found in Section 2.5. The reason for that is that there are some technical information required which are given in Section 2.3. Figure 1.3 shows the reasoning steps of PRAC. The following subsections outline every step.

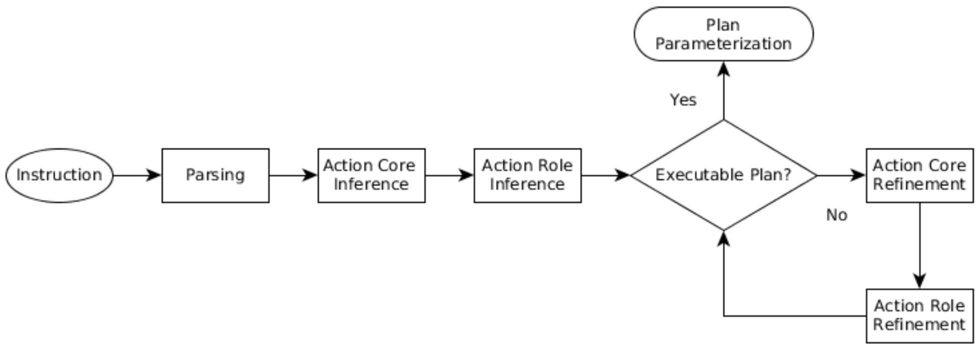


Figure 1.3: PRAC reasoning pipeline [26]

1.2.1.1 Parsing

Every instruction is given in natural-language to PRAC. So at the beginning of the reasoning process, this instruction is transformed to a set of atoms. These atoms represent the grammatical relations between the words in the instruction. For instance, consider the task "pour some water into a cup". In this sentence, "pour" is the predicate and "water" is its direct object and "cup" its prepositional object. The transformation (depicted in Figure 1.4) is done by the Stanford Parser [20] which is explained in more detail in Section 2.4.1. These atoms are stored in a database which is used as evidence for the next inference steps.

```
det(cup-6,a-5)
det(water-3,some-2)
dobj(Pour-1,water-3)
has_pos(Pour-1,VB)
has_pos(a-5,DT)
has_pos(cup-6,NN)
has_pos(some-2,DT)
has_pos(water-3,NN)
prep_into(Pour-1,cup-6)
```

Figure 1.4: 'Pour some water into a cup' processing result by the Stanford Parser[6]

1.2.1.2 Action Core Inference

The next step of this process pipeline is the action core inference. During the action core inference, the database with the grammatical relations is used as evidence. For every word in the database, all possible concepts are queried from the PRAC dictionary. Given these concepts, the classifier determines the most likely action core. With this solution, PRAC is able to infer the action cores for sentences such "fill a cup with water", "start with filling a cup with water" and "perform a neutralization on hydrochloric acid". It is even possible to handle synonyms. So for example, the classifier is trained with sentences such as "flavor the chicken with pepper" and the classifier is able to identify the 'Flavoring' action core in a sentence such as "season the rib with salt". After the action core is inferred, the next step is to determine the action roles and the correct meanings of the entities in the given instruction.

1.2.1.3 Action Roles Inference

After the action core has been determined, the next step is to infer the correct action roles and the corresponding senses. For example, the 'Pouring' action core has attached the roles ***theme*** and ***destination***. For each action core there is an individual joint probability distribution which is used for the inference. With these distributions, it is possible to differentiate between the semantics of cup in sentences like "fill a cup with water" and "add a cup of sugar to the mixture". In addition, the system infers that cup has the role of a destination in the first sentence and the role as a unit in the second one.

1.2.1.4 Executable Plan Inference

Since the action cores are designed on a conceptional level, they need some additional inference. Consider the sentences "add some pepper to the chicken" and "add some water to the mixture". Both of these sentences activate the action core 'Adding'. In addition, "water" and "pepper" represent the action role ***theme*** and "chicken" and "glass" the role ***destination*** in the corresponding sentences. To be

[6]has_pos represents that the word has a specific part-of-speech tag. Part-of-speech tags indicate e.g. that a word is a noun or verb. More details are given in Section 2.1.2

able to infer the correct plan, the system uses the action roles as evidence. For instance, based on "pepper", the system infers that it is most likely to perform the "using-spice-jar" plan. After the plan has been inferred, the system can refine the current action roles. For instance, "pepper" gets the role as a spice and "chicken" will be marked as the object to be flavored. If an executable plan is inferred, the system asserts the plan parameters with the refined action roles.

1.2.2 CURRENT STATE ABOUT THE MISSING INFORMATION INFERENCE

In it current state, PRAC is only able to infer missing information with the distributions used in the action role inference. As mentioned in Section 1.1, this solution is intractable. So the goal of this thesis is to provide a knowledge base where PRAC can look up for missing information. The idea is that this knowledge base contains sentences which are annotated with the action core and action roles. These sentences are extracted and annotated from natural-language documents by an information extraction system using the probability distributions of PRAC.

The missing information inference can be interpreted as a missing action role inference. After the regular action role inference, the system is able to determine which roles are missing. These missing roles can be looked up in the database by performing semantic information retrieval. This means, the inferred action roles can be used to find sentences in the database which are semantically similar and contain the missing action roles. Since there is a semantic comparison between the given task and the sentences stored in the database, the system is able to evaluate how suitable the missing role is for the current action core.

1.3 RELATED WORK

This section provides an overview about similar frameworks like PRAC, solutions to extract knowledge from natural-language documents and to utilize this knowledge.

Currently, there are many solutions which are able to infer executable plans using probabilistic models [8][22][24][31]. Like PRAC, some of them are able to infer reasonable plans even on unseen objects using taxonomic knowledge. However, only one solution explicitly addresses the missing information problem [24]. The **Tell Me Dave**[7] system is able to infer from a training sentence such as "throw the drinks in the trash bag" that for the instruction "throw away the chips" a "trash bag" can be used too. However, they use a joint probability distribution to achieve that. So their implementation does not scale on large data sets too. To the best of our knowledge, no system provides a solution which is able to infer reasonable missing information and still scales on large data sets.

To support the semantic information retrieval, we need an approach which is able to extract knowledge from natural-language documents and provide it to PRAC.

[7]http://tellmedave.com/

The following subsections give a brief overview about the current state-of-the-art information extraction systems.

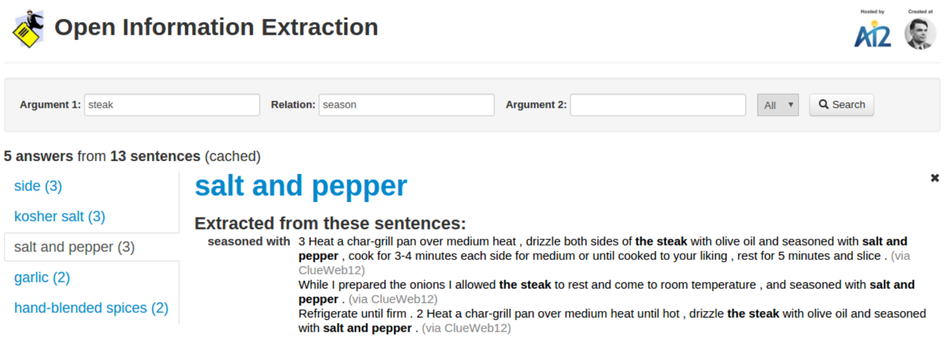
1.3.1 TEXT RUNNER

Text Runner[8] is a self-supervised learning system which extracts knowledge of natural-language documents. The extracted knowledge is represented as tuples [15]. A tuple T represents a relationship between two entities. T is defined as $T = (e_i, r_{ij}, e_j)$, where e_i and e_j are entities and r_{ij} are the text tokens defining the relation between those entities. Consider the sentence "Harry Potter is written by Rowling". This sentence can be represented as a tuple, where "Harry Potter" and "Rowling" are the entities and "is written by" is the relation.

Text Runner is called a self-supervised learning system because it generates its own training data. To create this training set, a dependency parser is used to parse a corpus. A dependency parser represents the grammatical relations of a sentence as a tree called dependency tree. A more detailed description is given in Section 2.1.1. For each dependency tree, the path between nouns is extracted. These paths are labeled as "trustworthy" if they are representing a valid relation. Predefined constraints, such as the length of a path cannot exceed a defined limit, rate if a path is trustworthy. Every trustworthy path is mapped as a feature vector to train the classifier. The sequence of part-of-speech tags is one feature, for instance.

During the information extraction process, TextRunner applies a part-of-speech tagger on each sentence in the corpus. The output of this tagger is given as evidence to the classifier which evaluates every sentence if it contains a trustworthy relation. All trustworthy relations will be stored in a knowledge base.

An evaluation of TextRunner shows that this system cannot be used for semantic information retrieval. Consider the task to determine suitable spices to season a steak. TextRunner is capable to find some correct answers like garlic, pepper and salt but it is starting to have problems to identify "pepper and salt" as single instances (Figure 1.5).



5 answers from **13 sentences** (cached)

side (3)
kosher salt (3)
salt and pepper (3)
garlic (2)
hand-blended spices (2)

salt and pepper

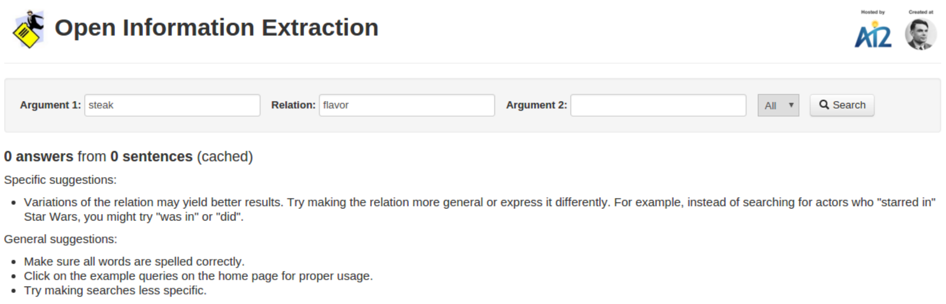
Extracted from these sentences:

seasoned with 3 Heat a char-grill pan over medium heat , drizzle both sides of **the steak** with olive oil and seasoned with **salt and pepper** , cook for 3-4 minutes each side for medium or until cooked to your liking , rest for 5 minutes and slice . (via ClueWeb12)
While I prepared the onions I allowed **the steak** to rest and come to room temperature , and seasoned with **salt and pepper** . (via ClueWeb12)
Refrigerate until firm . 2 Heat a char-grill pan over medium heat until hot , drizzle **the steak** with olive oil and seasoned with **salt and pepper** . (via ClueWeb12)

Figure 1.5: Text Runner results for the query 'season a steak'

[8]http://openie.allenai.org/

An additional example shows that a text-based knowledge representation is not suitable to perform semantic information retrieval. Referring to our "season the steak" example, just replacing "season" with "flavour/flavor" leads to a complete different result (see Figure 1.6). It shows that the system does not handle synonyms. In conclusion, we can say that text-based queries and knowledge representations can only be used for retrieving information which is explicitly mentioned in the corpus.



Open Information Extraction

Argument 1: steak **Relation:** flavor **Argument 2:** All Search

0 answers from **0 sentences** (cached)

Specific suggestions:

- Variations of the relation may yield better results. Try making the relation more general or express it differently. For example, instead of searching for actors who "starred in" Star Wars, you might try "was in" or "did".

General suggestions:

- Make sure all words are spelled correctly.
- Click on the example queries on the home page for proper usage.
- Try making searches less specific.

Figure 1.6: Text Runner results for the query 'flavor a steak.'

1.3.2 PRISMATIC

PRISMATIC defines an approach to extract knowledge from natural-language documents [16]. It was developed by IBM for creating a knowledge base for the IBM Watson system which won the Jeopardy show in 2011 [17].

PRISMATIC works in two phases. During the first stage, shallow knowledge is extracted from the corpora. Knowledge, such as Einstein and Marie Curie won the Nobel Prize, is defined as shallow knowledge. Based on this extracted information, aggregate statistics are used to perform semantic inference. To refer to the Einstein/Curie example, using a large data set, it is possible to infer that scientists win Nobel Prices. This semantic inference is performed during the second stage.

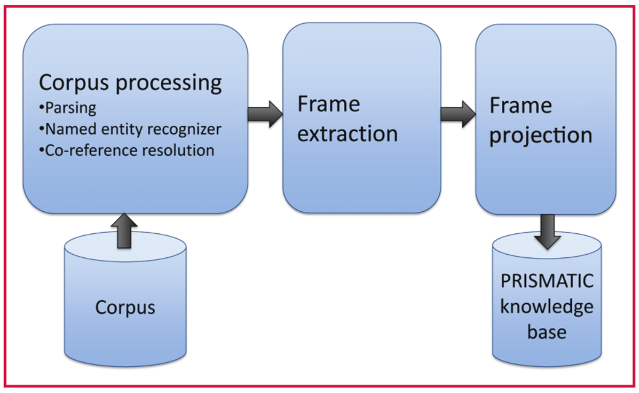


Figure 1.7: Process pipeline of PRISMATIC [16]

Figure 1.7 shows the process pipeline of IBM's knowledge extraction approach. First, corpus processing is performed. A parser creates a dependency tree. A dependency tree represents the grammatical relations of the words in the given sentence (see Section 2.1.1 for more information). An example of a parsed sentence is depicted in Figure 1.8.

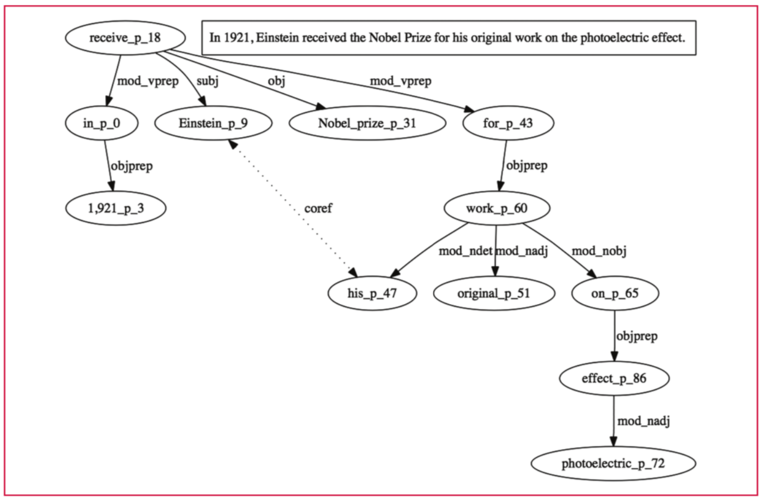
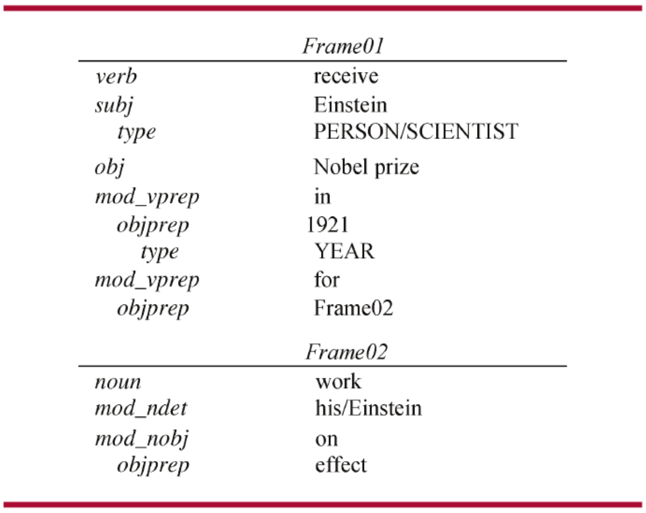


Figure 1.8: A dependency tree which is used for frame extraction [16]

Frame extraction is the second step of the process pipeline. In this phase, the extracted grammatical relations of a sentence will be represented as a frame (Figure 1.9). These frames represent the idea of describing a predicate and its immediate participants. The frame is an attribute-value pair. IBM calls the attributes **slots** and the corresponding values are called **slot values**.



Frame01	
verb	receive
subj	Einstein
type	PERSON/SCIENTIST
obj	Nobel prize
mod_vprep	in
objprep	1921
type	YEAR
mod_vprep	for
objprep	Frame02

Frame02	
noun	work
mod_ndet	his/Einstein
mod_nobj	on
objprep	effect

Figure 1.9: Extracted frame by PRISMATIC [16]

The type annotation for the extracted words is performed by using WordNet. PRISMATIC does not apply a probabilistic inference process to perform it. It gets a predefined list of concepts. Every word, which can be mapped to a concept, which is contained in this predefined list, gets this concept as a type [23].

The last step of this pipeline is frame projection. Frame projections can be understood as creating specific sub-frames. For instance, N-P-OT and S-V-O are frame projections, where N stands for noun, P for preposition, OT for object-type, S for subject, V for verb and O for object. Referring to "Frame01" in Figure 1.9, an S-V-O frame projection of "Frame01" would be a sub-frame which has the attributes "subj", "verb" and "obj" with the values "Einstein", "receive" and "Nobel prize". With frame projections it is possible to infer e.g. that the most likely object-type in context of the noun "annexation" is "region". To achieve that, we can query N-P-OT frame projections, which have "N" assert with the attribute "annexation" and "P" assert with the preposition "of". This query considers frames which are representing sentences such as "the annexation of Texas was in 1845". In the next step, the queried frames can be grouped by the object-type. The group with the highest frequency represents the most probable object-type.

Compared to Text Runner, PRISMATIC uses WordNet and does not use a text-based knowledge representation. That provides the possibility to perform semantic information retrieval. However, the current approach of PRISMATIC does not solve all our requirements. Consider a corpus with the sentences "season the steak with garlic" and "flavor the pancake with sugar". After PRISMATIC processed this corpus, we can query for answers like which spice can be used to season a steak. However, querying for an adequate spice for flavoring a waffle would result that garlic and sugar would have the same probability. In a real life scenario, it is more reasonable to use sugar for the waffle. This behavior can be explained by the implementation how PRISMATIC is using WordNet. IBM does not mention that they are using the whole taxonomy or even consider the relationship between these concepts [16][23]. Referring back to our example corpus, "steak" and "pancake" would be typed as food and "sugar" and "garlic" as spice.[9] So the system learns that food can be seasoned with spices, but it cannot learn that baked goods are flavored with sweeteners, for instance.

PRSIMATIC also uses statistical aggregation to determine the answers. This results in the side effect that the answers are dependent of the amount of information in the corpus. For example, a corpus that contains 99 sentences describing how to flip different kinds of meat with tongs and 1 sentence describing how to flip a waffle with a spatula. Sending a query to determine a suitable utensil for flipping a pancake would result that the tongs would get a higher probability than the spatula.

[9]This is just an assumption, how PRISMATIC would behave since this system is not open source and therefore it is not possible to test this system. The papers do not mention any use of a taxonomy and based on the description how the frame extraction and projection is performed, it is reasonable to assume such results.

1.3.3 NEVER-ENDING LANGUAGE LEARNER

Never-Ending Language Learner (NELL)[10] is an additional information extraction system [10]. Starting with an initial knowledge base, NELL has to extend this database by processing additional corpora.

Figure 1.10 shows the architecture of NELL. On every iteration, four subsystem components acquire new knowledge using the facts stored in the knowledge base.

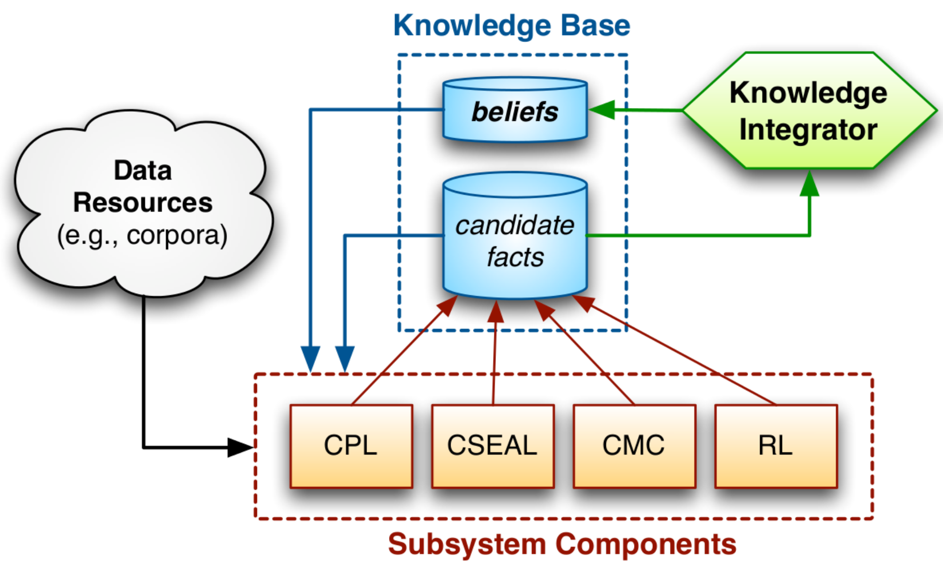


Figure 1.10: Architecture of NELL [10]

Coupled Pattern Learner (CPL) extracts knowledge of natural-language documents based on the categories and relations in the knowledge base. For instance, applying the relation **is written by** on a sentence, it is possible to determine that the entities between this relation are of the categories ***book*** and ***author***.

Coupled Set Expander for Any Language (CSEAL) extracts new entities and relations of semi-structured documents (e.g. HTML and XML).

Coupled Morphological Classifier (CMC) is a set of logistic regression models. These classifiers recognize to which category noun phrases belong to. For instance, the phrase "chemical book" will be classified as the category ***book***.

Rule Learner (RL) entails new relations based on relations which are already contained in the knowledge base.

Every subsystem component ranks its extracted knowledge. Based on this ranking, the Knowledge Integrator determines which new facts will be added to the knowledge base.

[10]http://rtw.ml.cmu.edu/rtw/kbbrowser/

In general, it would be possible to use the knowledge base of NELL for the PRAC framework. However, creating a compatibility between NELL and PRAC requires a lot of effort. NELL provides an interface for querying information based on relations. For instance, NELL learns the relation **capital city of**. If we want to query for the capital city of France, the system looks for a second entity of this relation given "France" as the first entity. So the first step to make the PRAC framework compatible to NELL is to implement a classifier which is able to map the natural-language instructions to these relations. In addition, PRAC uses WordNet to represent the senses of the words. Another step would be to map these senses to the categories of NELL or replace WordNet. Even though we are not intending to use NELL for the PRAC framework, we utilize the idea of using an initial ontology to extract knowledge for our information extraction solution. A detailed description is given Section 3.1.

1.4 Thesis Contributions

As pointed out in the previous section, the current solutions do not provide any tractable approaches to perform the missing information inference. Regarding the information extraction systems, their retrieval algorithms are not feasible enough to perform the semantic information retrieval on a level like we are intending to. So we address the listed issues by the following contributions:

1. We show that with the PRAC framework it is possible to extract knowledge from natural-language documents. Since PRAC uses joint probability distributions to perform e.g. action roles inference, we present an approach to create a small training set which trains the model to be able to extract most information of a given corpus, utilizing the taxonomic knowledge about concepts.
2. Our defined knowledge representation for the extracted information is used to overcome the scaling problem of joint probability distributions. However, it is still suitable to perform semantic information retrieval. We focused on creating a representation which is simple enough to avoid complex join queries but still able to represent the required information.
3. During this work, we developed a semantic information retrieval algorithm which is able to find possible solutions in $O(n)$ where n is the size of the sentences in the knowledge base. It uses the inferred action roles in the given instruction to determine the semantically most similar sentence in the knowledge base. So it is able to determine suitable missing action roles even then the given instruction was not mentioned in the corpus. Based on this semantic similarity, the search algorithm also returns a confidence measure to display how sure the system is about the found solution.

The remainder of the thesis is organized as follows: In Chapter 2 we provide some technical foundations to get a better understanding of the natural-language processing and the probabilistic models which are used in the PRAC framework.

Chapter 3 describes our tractable solution for the missing information inference. We present the approach to extract knowledge of natural-language documents, the knowledge represention to store the extracted information and the semantic information retrieval algorithm.

In Chapter 4 we evaluate our solution on a self-created corpus and on a corpus extracted from WikiHow. The intention of the self-created corpus is to compensate the lack of other NLP solutions which are mentioned in Section 2.1.2. WikiHow is an online service where people can upload instructions which describe how to perform certain tasks.

Chapter 5 concludes this work and gives an outlook of future investigations.

CHAPTER two

TECHNICAL FOUNDATIONS

The intention of this chapter is to contribute to a better understanding of this thesis. We start with a brief introduction to natural-language processing (NLP). We define some terminologies and provide an overview about some NLP tasks. An essential part of this section is parsing. The results of the parser are part of the evidence used by PRAC to infer the executable plans. Also, parsing is used for the information extraction process. Since the outcome of the inference and extraction process depend on a correct parsing result, we want to provide an overview about the general approach of parsing and the difficulty to create the perfect parser.

After the NLP section, we provide a brief refresher on logic. This section should explain the necessary elements to understand Markov logic network - the probabilistic model which is used in the PRAC framework to infer the action cores and the corresponding action roles.

This chapter also gives an introduction to Markov logic networks. We describe the general definition, inference and learning process and how it is possible to enable the use of taxonomic knowledge in such models.

Afterwards, we describe the additional tools which are used by PRAC to support the action role inference - Stanford Parser and WordNet. Since we want to use the action role inference to extract semantic information from natural-language documents, it is neccesarry to understand the complete process. So at the end of this chapter when all required technical information are introduced, we give a detailed description about the action role inference process.

2.1 NATURAL-LANGUAGE PROCESSING

In general, natural-language processing is about developing methods for handling human languages by computers [11]. Natural-language understanding, machine translation and question answering are some tasks to solve. The following subsections describe the encountered NLP tasks during this work.

2.1.1 PARSING

The goal of parsing is to represent a given sentence as a syntax tree. Given a set of defined grammar rules, a parser is able to verify if the given sentence is created by these grammar rules. Every valid sentence can be represented as a tree which can be used for further NLP tasks.

The next subsections describe the parse tree generation process, the different types of parse trees and the challenge which arises during parsing.

2.1.1.1 Context-free Grammar

A context-free grammar (CFG) (see Definition 2.1 [12]) is a set of rules which define a language. As a reminder, a CFG represents the Type-2 grammar of the Chomsky hierarchy [11]. By applying these grammar rules, it is possible to create sentences which are part of a defined language. Given the sentence "the man sleeps" a parser generates a tree like in Figure 2.2 by considering the grammar in Figure 2.1.

Definition 2.1 *A CFG is defined by a 4-tupel $G = (N, \Sigma, R, S)$, where:*

- *N is a finite set of non-terminal symbols.*
- *Σ is a finite set of terminal symbols.*
- *R is a finite set of rules. Each rule has the form $X \rightarrow Y_1 \ldots Y_i$, where $X \in N$ and $Y_i \in (N \cup \Sigma)$ for $i = 1 \ldots n$.*
- *S is a start symbol.*

N ={S, NP, VP, PP, DT, Vi, Vt, NN, IN}
S =S
Σ ={sleeps, saw, man, woman, dog, telescope, the, with, in}
R =

S→NP VP
VP→Vi VP→Vt NP VP→VP PP
NP→DT NN NP→NP PP
PP→IN PP
Vi→sleeps Vt→saw
NN→man NN→woman NN→telescope NN→dog
DT→the
IN→with IN→in

Figure 2.1: An example for a context free grammar [12]

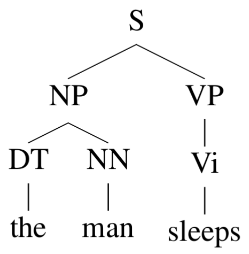


Figure 2.2: Parse tree of the sentence ’the man sleeps’ [12]

2.1.1.2 Ambiguity

The CFG (Figure 2.1) is part of the English grammar. One attribute of natural-languages is that they are ambiguous. For example, consider the sentence "the man saw the dog with the telescope". The sentence can be interpreted in at least two different ways. One interpretation is that the man used a telescope to spot the dog.

Another interpretation could be that the man saw a dog which is having a telescope [12]. Applying the rules of the example CFG, we can represent these interpretations as two distinguished parse trees (Figure 2.3). This example demonstrates the difficulty of parsing, since there exist multiple parse trees for a sentence and only one can be correct.

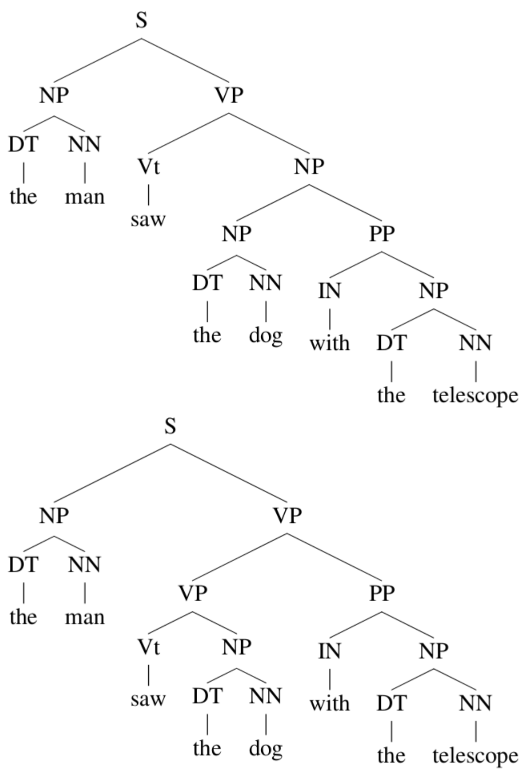


Figure 2.3: Two different parse trees of 'the man saw the dog with the telescope' [12]

2.1.1.3 Probabilistic Context Free Grammar

To approach the ambiguity problem, one method is to use a probabilistic context free grammar (PCFG). PCFG is an extension of CFG, including that every rule has a probability attached [12]. Figure 2.4 displays the example grammar as PCFG.

N ={S, NP, VP, PP, DT, Vi, Vt, NN, IN}
S =S
Σ ={sleeps, saw, man, woman, dog, telescope, the, with, in}
R =

S→NP VP	1.0
VP→Vi	0.3
VP→Vt NP	0.5
VP→VP PP	0.2
NP→DT NN	0.8
NP→NP PP	0.2
PP→IN PP	1.0
Vi→sleeps	1.0
Vt→saw	1.0
NN→man	0.1
NN→woman	0.1
NN→telescope	0.3
NN→dog	0.5
DT→the	1.0
IN→with	0.6
IN→in	0.4

Figure 2.4: An example for a probabilistic context free grammar [12]

To determine the most probable parse tree for a given sentence, the first step is to generate all possible valid parse trees. For each valid parse tree, all probabilities of the applied grammar rules will be multiplied. The parse tree with the highest result is the most probable parse tree [12].

2.1.1.4 Constituency Parse Tree VS Dependency Parse Tree

There have been two different parse tree models established during the time - constituency and dependency parse trees [11]. The tree in the previous examples (Figure 2.2) is a constituency parse tree. Constituency parsers segment sentences in phrases such as noun and verb phrases.

The main idea of a dependency parse tree is that the verb is the key element and the remaining words in the sentence depend on it. These dependencies representing e.g. the direct object or the prepositional object of the verb.

Figure 2.5 shows the two different parse tree models for the sentence "John hit the ball".

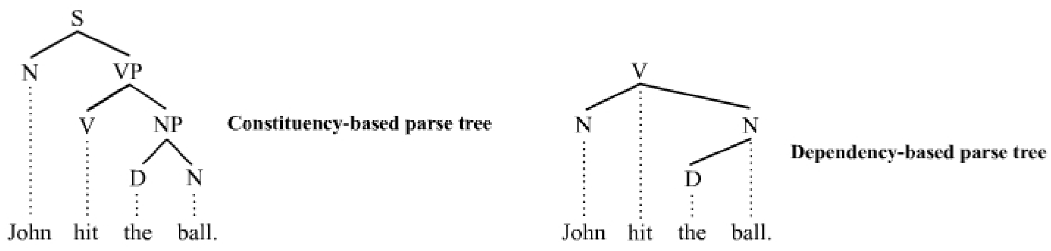


Figure 2.5: 'John hit the ball' represented as two different types of parse trees[1]

2.1.2 ADDITIONAL NATURAL-LANGUAGE TASKS

In this section we want to introduce additional natural-language tasks which occur during information extraction and retrieval. We explain briefly the goal of each task and how we handle these tasks during the thesis.

Part-of-speech Tagging The task of part-of-speech (POS) tagging is to identify the correct word types such as noun, verb or adjective [11]. Sentences like "book the flight" can be challenging for a POS tagger. Since a subject is missing in this sentence, a part-of-speech tagger could have the problem to differentiate if "book" is a noun or a verb.

To be able to e.g. infer the correct senses for the words in an instruction, it is required to know the part-of-speech of each word. The benefit of the Stanford Parser is that in addition to the grammatical relations, it provides POS tags to each word in the given sentence. So we do not need to use an extra POS tagger for this task.

Named-entity Recognition Named-entity recognizers (NER) identify and classify entities in a given sentence [11]. For instance, consider the sentence "Bob married Alice". In this sentence, "Bob" and "Alice" are the entities to be recognized. A successful recognition leads to classification. In this example there are many possibilities. "Bob" and "Alice" could be assigned with the class label ***Person***. Another classification could be ***husband*** and ***wife***.

The joint probability distributions which are used in PRAC to infer the action roles can be interpreted as Named-entity recognizer. Regarding the mentioned example, we could design a 'Marrying' action core with the roles ***husband*** and ***wife*** and perform an inference on the sentence "Bob married Alice". In addition, it would be possible to map the names "Bob" and "Alice" to the senses "husband" and "wife" from WordNet. Since these senses are represented in a taxonomy and therefore they have an is-a relation to the concept "person", it would be possible to infer that these two instances are people.

[1]Figures by Tjo3ya (Own work) [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

Coreference Resolution The task of coreference resolution is to find expressions in a text which refer to the same entity [11]. Consider the following sentences:

- Alice, who is married to Bob, works as a nurse.
- Her husband is a car mechanic.

In the first sentence, "who" is referencing to "Alice". In the second one, a coreference resolver has to resolve that "her husband" is referencing to "Bob".

In this thesis we do not use a coreference resolver. The reason for that is that PRAC does not use any resolver at the current time. So the options remain to look for open source solutions and evaluate their accuracy or starting to implement an own resolver. Since both possibilities require a lot of time, we decided to solve this task in the future.

Word Sense Disambiguation The word sense disambiguation (WSD) problem is the task of determining the correct meaning for words in a given text. The two sentences show that the word "cup" can have a different sense depending on the context.

- Fill a cup with water.
- The soccer team won the cup.

In the first sentence, "cup" has the sense of a container for holding liquids. In the second one, "cup" has the sense of a trophy.

PRAC handles this problem during the action role inference, so it is not necessary to provide any additional solutions during this thesis.

Relationship Extraction The task of relationship extraction is to identify the relation between two entities. For instance, the sentence "Google acquire YouTube". The two entities in this example are "Google" and "YouTube". We can represent their relation as a predicate such as ***acquiredBy(YouTube,Google)*** [11].

An action core represents an indirect relation between the action roles. Regarding the mentioned example, we could design an 'Acquiring' action core with the roles ***acquirer*** and ***acquiree*** and perform an inference on the sentence "Google bought YouTube".

Information Extraction Information extraction (IE) systems extract knowledge from unstructured data and represent it in a defined structure. The main approach to extract knowledge is to process every sentence by performing NER and relationship extraction and store their results in a knowledge base [11].

There is a differentiation between closed-domain and open-domain systems. Closed-domain systems apply predefined NER and relationship detectors to extract the knowledge. However, open-domain systems perform this extraction with self-learned entities and relations.

Our information extraction solution is using the joint probability distributions of PRAC to perform the named-entity and relationship detection. Since the supported action cores in PRAC are added by people, our solutions represent a closed-domain information extraction system.

Information Retrieval The aim of information retrieval is to look up efficiently for relevant information in corpora [11]. The relevant information can be contained in documents, paragraphs or single sentences. In the classical information retrieval the documents are processed and then represented as a vector. There are many possibilities to represent documents as a vector. For instance, the vector can be a boolean vector which represents the appearance of words in a document. So every vector V_i has the elements v_1 to v_n, where v_n represents if a specific word appears in the document i. For example, v_3 represents if the word "steak" appears in a document. This fix vector representation is applied for each document. So for every document represented as a vector, its element v_3 indicates the appearance of the word "steak" in respective document. This fixed representation allows comparing these vectors via similarity measures such as Euclidean distance.

To retrieve relevant documents, the query can be defined as list of keywords. This list of keywords can be transformed to a vector which has the same representation as the documents in the database. Based on similarity between the query vector and the document vectors, it is possible to determine that the document vector with the highest similarity is the most relevant document.

The classical information retrieval approach does not consider the semantic between words during the similarity calculation. This leads to the same drawbacks as the syntactic search which are mentioned in Section 1.1. In our thesis we aim to retrieve action-core-specific information from sentences contained in natural-language documents. With action-core-specific information we mean that a sentence like "season the steak with pepper" represents a 'Flavoring' action core with "steak" as object to be seasoned and "pepper" as a suitable spice. This kind of knowledge can be used to complete the instruction "flavor the chicken". Such inferences can be only performed by considering the semantic between words.

2.2 A Refresher in Logic

An additional key component in this thesis is the understanding of Markov logic networks. Since logic is an essential part of Markov logic networks, we give a brief refresher to this topic, starting with propositional logic. First-order logic is explained as second. This logic is applied in Markov logic networks. After the basics are set, we introduce fuzzy logic.

Since this topic has to be seen as a refresher, we refer to [30] if there is any interest to get more information about logic.

Propositional Logic In general, logic provides the possibility to represent sentences in a symbolic representation. This representation provides the foundation to determine the truth value of a sentence [30]. A sentence such as "when it is raining then the street is wet" can be represented as a formula. The part "it is raining" can be represented as R and "the street is wet" as W. R and W are called atoms. These atoms can be asserted with truth values - ***true*** and ***false***.

Applying logical junctions, multiple atoms can be compound to formulas. Table 2.1 lists the available junctions and the resulting truth values based on the asserted truths for the atoms A and B. Our raining example can be represented as $R \implies W$.

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \implies B$	$A \Leftrightarrow B$
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

Table 2.1: Truth table for logical junctions

First-order Logic First-order logic (FOL) is more expressive compared to propositional logic. With propositional logic we are only capable to represent facts. In FOL we define predicates to describe entities and relations [30]. Consider the rain example (see 2.2) and a world with the cities Berlin, Amsterdam and New York. Now we want to model if it rains in a specific city, the streets in this city will be wet. To design a knowledge base with propositional logic, we have to create three formulas. For each city we have to define two atoms. One representing "it is raining" and the other "the street is wet". For instance, $rains\_in\_berlin \implies streets\_are\_wet\_in\_berlin$. With first-order logic we need to define only one formula:

$$\forall x(rains\_in(x) \implies streets\_are\_wet\_in(x))$$

where x represents a city. In first-order logic we have additional quantifiers. To represent that some formulas are valid for all entities or only for some of them we can attach $\forall$ or $\exists$ to the formulas.

For the Markov logic network section (see Section 2.3.2) we need to define some additional terminology regarding FOL [29]. Variables and constants such as x and $Berlin$ can be typed. Meaning that e.g. $Berlin$ is an instance of the type **city**. These two expressions are called **terms**. A predicate applied to a tuple of terms (e.g. $rains\_in(y)$) is called an **atom**. If a term does not contain any variable, it is called a **grounded term**. A **grounded atom** is an atom whose arguments are grounded terms.

Fuzzy Logic In fuzzy logic the truth value is represented as a value between 0 and 1, where 0 equals ***false*** and 1 equals ***true***. The logic operations are defined as follows [27]:

- $T(A \wedge B) = min(T(A), T(B))$
- $T(A \vee B) = max(T(A), T(B))$
- $T(\neg A) = 1 - T(A)$

where $T(x)$ returns the truth value for the corresponding formula or literal x.

2.3 STATISTICAL RELATIONAL LEARNING

The advantage of first-order logic is that it allows us to define compactly complex relations about entities through universal quantification. In addition, the FOL knowledge bases are simple to design and to understand. One important aspect of logic knowledge bases is that they have to be consistent to be able to provide correct answers to given queries [30]. The consistency verification of a FOL knowledge base cannot be performed due to the property that the satisfiability problem in first-order logic is undecidable [7]. In addition, FOL cannot handle uncertainty. This means we are not able to provide probabilities to the evidences.

Probabilistic graphical models on the other hand are able to handle uncertainty. However, their knowledge representation is difficult to model and for people who were not evolved in the design process, it can be a challenge to understand the design. In addition, it is difficult to capture relational knowledge with probabilistic models [14].

Statistical relational learning (SRL) is the field which combines these two methods. This field concerns with the task to provide the possibility to represent compact relational knowledge about entities and also to handle uncertainty. NLP requires such an approach since determining part-of-speech tags or solving the WSD problem requires relational knowledge [9].

In this section we describe Markov logic networks which are part of the field statistical relational learning. This model is used to design the joint probability distributions which perform the action role inference in our information extraction system.

2.3.1 MARKOV NETWORK

In this section we give an introduction to Markov network. A Markov network is a probabilistic model, which is part of Markov logic networks.

2.3.1.1 Definition

A Markov network (also called Markov Random Field(MRF)) is an undirected graph model to represent a joint distribution over a set of random variables. Every node in the graph represents a random variable. The edges between nodes represent dependencies between the variables. To define a MRF, we will introduce some necessary definitions.

At first, the definition of a factor [21]:

Definition 2.2 *A factor $\phi(\boldsymbol{D})$, where $\boldsymbol{D}$ denotes a set of random variables, is a function which $\phi(\boldsymbol{D}) \mapsto \mathbb{R}^+$.*

Given multiple factors, it is possible to merge them into one factor [21]:

Definition 2.3 *Given three disjoint sets of random variables $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$ and two factors $\phi_1(\boldsymbol{X}, \boldsymbol{Y})$ and $\phi_2(\boldsymbol{Y}, \boldsymbol{Z})$. A product factor $\phi_1 \cdot \phi_2$ can be defined as a factor $\psi : Val(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) \mapsto \mathbb{R}^+$, where*

$$\psi(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) = \phi_1(\boldsymbol{X}, \boldsymbol{Y}) \cdot \phi_2(\boldsymbol{Y}, \boldsymbol{Z}).$$

$Val(X)$ returns all values of the random variable X. $Val(X_1, ..., X_n)$ returns values for all combinations with $X_1, \ldots, X_n$.

Given Definition 2.2 and 2.3, we can define a Gibbs distribution [21]:

Definition 2.4 *A distribution P_Φ can be parameterized as a Gibbs distribution as follows:*

$$P_\Phi(X_1, ...X_n) = \frac{1}{Z}\widetilde{P}_\Phi(X_1, ...X_n),$$

where

$$\boldsymbol{D}_i \subseteq \{X_1, ...X_n\},$$
$$\widetilde{P}_\Phi(X_1, ...X_n) = \phi_1(\boldsymbol{D}_1) \cdot ... \cdot \phi_m(\boldsymbol{D}_m),$$

defines an unnormalized distribution and

$$Z = \sum_{X_1, ..., X_n} \widetilde{P}_\Phi(X_1, ...X_n)$$

is called partition function and can be used as normalization constant.

A Markov network can be defined by a Gibbs distribution [21]:

Definition 2.5 *A Gibbs distribution P_Φ factorizes over a Markov network Ψ if for each factor $\phi(\boldsymbol{D}_i)$, $\boldsymbol{D}_i$ represents a clique in Ψ. The factors are also called clique potentials.*

For better understanding, we will provide an example. Consider a Markov network Ψ with three binary random variables A, B and C. The dependency relation is $(A \perp C \mid B)$. This notation represents that A and C are independent if B is known. This Markov network is depicted in Figure 2.6.

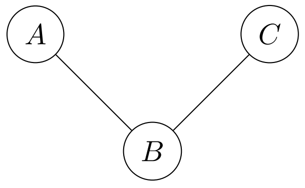


Figure 2.6: The example Markov network Ψ

Ψ has two cliques and each contains two nodes. Considering that every node represents a binary random variable, each clique has four possible states. These states are represented in Table 2.2. In this example the values attached to these states are picked randomly.

Assignments for A & B	$\phi_1(A,B)$	Assignments for B & C	$\phi_2(B,C)$
a^0 b^0	50	b^0 c^0	30
a^0 b^1	10	b^0 c^1	25
a^1 b^0	10	b^1 c^0	50
a^1 b^1	20	b^1 c^1	40

Table 2.2: The clique potentials for the Markov network Ψ

Given the factors in Table 2.2, we can calculate the joint distribution for Ψ which is represented in Table 2.3.

Assignments	Unnormalized	Normalized
a^0 b^0 c^0	$\phi_1(a^0,b^0)\cdot\phi_2(b^0,c^0)=1500$	0.25
a^0 b^0 c^1	$\phi_1(a^0,b^0)\cdot\phi_2(b^0,c^1)=1250$	0.208
a^0 b^1 c^0	$\phi_1(a^0,b^1)\cdot\phi_2(b^1,c^0)=500$	0.083
a^0 b^1 c^1	$\phi_1(a^0,b^1)\cdot\phi_2(b^1,c^1)=400$	0.067
a^1 b^0 c^0	$\phi_1(a^1,b^0)\cdot\phi_2(b^0,c^0)=300$	0.05
a^1 b^0 c^1	$\phi_1(a^1,b^0)\cdot\phi_2(b^0,c^1)=250$	0.042
a^1 b^1 c^0	$\phi_1(a^1,b^1)\cdot\phi_2(b^1,c^0)=1000$	0.167
a^1 b^1 c^1	$\phi_1(a^1,b^1)\cdot\phi_2(b^1,c^1)=800$	0.13
	$Z=6000$	

Table 2.3: Joint distribution for the Markov network Ψ

2.3.1.2 Log-Linear Model

A Markov Random Field can be represented as a log-linear model [21]. The advantage is that the representation is more compact compared to the clique potential representation. We need to introduce the definition of a feature for constructing this log-linear model [21].

Definition 2.6 *Given a set of random variables $\boldsymbol{D}$. A feature f is a function which $f(\boldsymbol{D}) \mapsto \mathbb{R}$.*

Applying Definition 2.6, a log-linear model for a Markov network can be defined as [21]:

Definition 2.7 *A distribution P over a Markov network Ψ can be defined as:*

$$P(X_1, ...X_n) = \frac{1}{Z} exp\left(\sum_{i=1}^{k} w_i f_i(\boldsymbol{D}_i)\right).$$

where

- *$f_i(\boldsymbol{D}_i) \in F$,where F is a set of features and $\boldsymbol{D}_i$ is a clique in Ψ,*
- *w_i is a weight associated to the feature f_i.*

Now we prove that this log-linear model is equivalent to the clique potential representation.

A factor $\phi(\boldsymbol{D})$ can be represented as [21]:

$$\phi(\boldsymbol{D}) = exp(ln(\phi(\boldsymbol{D}))) \tag{2.1}$$

Using Definition 2.4 and Equation 2.1 we have

$$\begin{aligned}
P(X_1, ...X_n) &= \frac{1}{Z}\left(\phi_1(\boldsymbol{D}_1) \cdot ... \cdot \phi_m(\boldsymbol{D}_m)\right) \\
&= \frac{1}{Z}\left(exp(ln(\phi_1(\boldsymbol{D}_1)) \cdot ... \cdot exp(ln(\phi_m(\boldsymbol{D}_m))\right) \\
&= \frac{1}{Z}\left(\prod_{i=1}^{m} exp(ln(\phi_i(\boldsymbol{D}_i))\right) \\
&= \frac{1}{Z} exp\left(\sum_{i=1}^{m} ln(\phi_i(\boldsymbol{D}_i))\right)
\end{aligned}$$

For each state k in clique ϕ_i we define a feature $f_{i_k}(\boldsymbol{D}_i)$, where

$$f_{i_k}(\boldsymbol{D}_i) = \begin{cases} 1 & k \in Val(\boldsymbol{D}_i) \\ 0 & \text{otherwise} \end{cases}.$$

For each $f_{i_k}(\boldsymbol{D}_i)$ the weight $w_{i_k} = ln(\phi_i(k))$ will be attached. This results in

$$\frac{1}{Z} exp\left(\sum_{i=1}^{m} ln(\phi_i(\boldsymbol{D}_i))\right) = \frac{1}{Z} exp\left(\sum_{i=1}^{k} w_i f_i(\boldsymbol{D}_i)\right)$$

To point out the benefit of this log-linear model, consider a scenario where two random variables X and Y of the same domain have each n values [21]. If X and Y define a clique there would be n^2 possible states. If we are interested in modeling the behavior $X = Y$, we can use a feature $f_i(X, Y)$, where

$$f_i(X, Y) = \begin{cases} 1 & X = Y \\ 0 & \text{otherwise} \end{cases}$$

instead of modeling a factor with n^2 values.

An additional benefit of this log-linear representation is that is more tractable to solve the optimization problem during learning since it is easier to determine the partial derivatives of a summation compared to a product.

2.3.2 MARKOV LOGIC NETWORK

A Markov logic network (MLN) [29] combines first-order logic with a probabilistic graphical model.

Definition 2.8 *A Markov logic network L is a set of pairs (F_i, w_i), where F_i is a first-order logic formula and w_i is a real number. Given a finite set of constants $C = \{c_1, \ldots, c_{|C|}\}$, it defines a Markov network $M_{L,C}$ as follows:*

1. *For each possible grounding of each predicate appearing in L, $M_{L,C}$ contains one binary node. If the ground atom is true, the value of the node is 1. Otherwise it is 0.*
2. *For each possible grounding of each formula F_i in L, $M_{L,C}$ contains one feature. If the ground formula is true, the value of this feature is 1. Otherwise it is 0. The weight of the feature is the w_i attached to F_i in L.*

A MLN can be seen as a template to create a Markov network which probability distribution is defined as follows [29]:

Definition 2.9 *The probability distribution P for a grounded Markov logic network $M_{L,C}$ can be defined as:*

$$P(X = x) = \frac{1}{Z} exp \left(\sum_{i=1}^{k} w_i n_i(x) \right)$$

where

- $n_i(x)$ *is the number of true groundings of* F_i *in* x*,*
- w_i *is a weight associated to the formula* F_i*.*

For better understanding, consider a MLN as displayed in Table 2.4 [29].

Weight	Formula	Interpretation
1.5	$\forall x Smokes(x) \Rightarrow Cancer(x)$	Smoking causes cancer.
1.1	$\forall x, y Friends(x, y) \Rightarrow (Smokes(x) \iff Smokes(y))$	If two people are friends, either both smoke or neither does.

Table 2.4: An example for a Markov logic network template

Given the terms Alice (A) and Bob (B), we can create a Markov network like in Figure 2.7 [29]. On this Markov network we are able to calculate e.g. the probability that Bob smokes if Alice is a smoker and they both are friends.

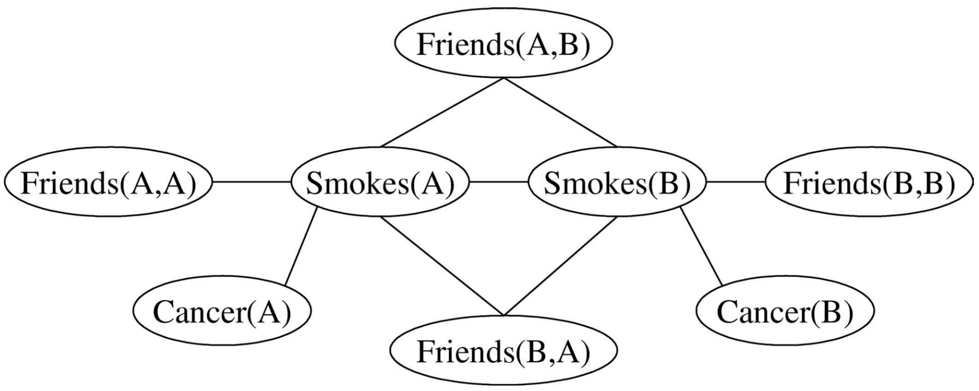


Figure 2.7: An example for a grounded Markov Network example

This example also shows the problematic scaling behavior of MLNs when the size of entities increases. Just having a knowledge base with the predicates $smokes$, $cancer$ and $friends$ will create $n^2 + 2n$ ground atoms, where n is number of entities. Since every ground atom is represented as a binary feature in a Markov network, there are 2^{n^2+2n} possible worlds to evaluate.

2.3.2.1 Inference

There exist two common query types when using probabilistic models - probability queries and maximum a posteriori (MAP) [21]. The probability query computes $P(\boldsymbol{Y}|\boldsymbol{E} = e)$, where $\boldsymbol{Y}$ and $\boldsymbol{E}$ are sets of random variables and e is assignment to the evidence variables. The result will be the posterior probability distribution over the values y of $\boldsymbol{Y}$. MAP queries try to find the most probable assignment to the variables in $\boldsymbol{Y}$. In other words:

$$MAP(\boldsymbol{Y}|e) = \operatorname*{argmax}_y P(y|e)$$

In this work we only consider MAP queries. Inference in MLNs is NP-hard [29] but MAP queries are less computational complex than probability queries. This can be explained by considering Bayes' theorem:

$$\operatorname*{argmax}_y P(y|e) = \operatorname*{argmax}_y \frac{P(y,e)}{P(e)} = \operatorname*{argmax}_y P(y,e)$$

However, to handle probability queries it is necessary to determine $P(e)$ which requires additional computational effort. In practice, there are use cases where performing a probability query is intractable. However, a MAP query can be still executed. One such use case is the inference of action roles. So to determine the action core and the corresponding roles, PRAC performs a MAP query instead of a probability query.

It is possible to convert a grounded MLN into a weighted constraint satisfaction problem (WCSP) (see Definition 2.10) [19]. Solving WCSP problems is still NP-hard but compared to MLN inference algorithms these problems are better studied. At the current time, they are efficient WCSP solver available e.g. ToulBar2[2]. Applying these solvers enables us to perform MAP queries more efficient compared to the current MLN inference algorithms.

[2]https://mulcyber.toulouse.inra.fr/projects/toulbar2/

Definition 2.10 *A WSCP is a 3-tupel $R = (Y, D, C)$:*

1. *$Y = \{Y_1 \dots Y_i\}$ is a finite set of variables.*
2. *$D = \{D_1 \dots D_i\}$ is a finite set of domains, where $D_i = dom(Y_i)$.*
3. *$C = \{c_1 \dots c_r\}$ is a finite set of soft constraints. Each constraint $c_i(V_i)$ is a function which maps an assignment $V_i = v$ to a cost value cv_j, where $V_i \subseteq D$ and $cv_j \in \{0, 1, \dots \top\}$. Will be a cost value asserted with $\top$, it is considered inconsistent.*
4. *A consistent assignment to the constraints is a solution to R. A valid solution is optimal if the summation of cost values over the constraints is minimal.*

In general, every grounded MLN can be transformed to a WCSP by adjusting the weights of the formulas. Every negative weight has to be positive and real numbers has to be mapped to natural numbers. Each grounded formula in the MLN is defined as a soft constraint. If the formula is not satisfied, the attached weight defines the cost of the constraint. Every satisfied formula has the cost of 0 [19]. In conclusion, a valid optimal solution in WCSP represents the world of a grounded MLN with the most probable assignment for a given MAP query. It has to be considered that a WCSP solver only returns the costs for the best solution. Unfortunately, to the best of our knowledge it is not possible to determine a probability for the solution based on these costs. This property has a negative impact for the information extraction which is described in Section 3.3.3.

2.3.2.2 Learning

The maximum log-likelihood method can be used to train a Markov network. The following partial derivative with respect to the weight w_i can be used for solving the optimization problem [29]:

$$\frac{\partial}{\partial w_i} log P_w(X = x) = n_i(x) - \sum_{x'} P_w(X = x')n_i(x')$$

where x is a given database. A database is a collection of grounded atoms with attached truth values.

Obviously the naive approach is intractable for larger databases since it has to perform an inference for all possible combinations of truth values x' of x. To improve the learning process, we can approximate the weights through a pseudo-likelihood function. Before introducing this function, we first need to define a function to calculate the probability of a ground atom X_l considering its Markov blanket MB_l with the state mb_l [29]:

$$P(X_l = x_l | MB_l = mb_l) =$$

$$\frac{exp(\sum_{f_i \in F} w_i f_i(X_l = x_l, MB_l = b_l))}{exp(\sum_{f_i \in F} w_i f_i(X_l = 0, MB_l = mb_l)) + exp(\sum_{f_i \in F} w_i f_i(X_l = 1, MB_l = mb_l))}$$

A Markov blanket of a node X_l in a Markov network is a set of all neighbors nodes of X_l.

The pseudo-likelihood is defined as follows [29]:

$$P_w^*(X = x) = \prod_{l=1}^{n} P_w(X_l = x_l | MB_x(X_l))$$

where $MB_x(X_l)$ is the state of the Markov blanket of X_l in the database. During the learning process we make a closed world assumption. Closed world assumption defines if something is not explicit mention as true, is as treated as false. So if the evidence database misses some grounded atoms, these grounded atoms will be treated as if they are set false. The partial derivative of the pseudo-likelihood with respect to the weights is [29]:

$$\frac{\partial}{\partial w_i} log P_w^*(X = x) = \sum_{l=1}^{n} [n_i(x) - P_w(X_l = 0 | MB_x(X_l)) n_i(x_{[X_l=0]}) \\ - P_w(X_l = 1 | MB_x(X_l)) n_i(x_{[X_l=1]})]$$

where $n_i(x_{[X_l=0]})$ is the number of true groundings of the ith formula when the ground atom X_l is set false.

2.3.3 FUZZY MARKOV LOGIC NETWORK

Fuzzy Markov logic networks are similar to regular MLN except they use the fuzzy logic calculus for the evaluation of the formulas (see Def. 2.11 [27]).

Definition 2.11 *The probability distribution P for a grounded Fuzzy Markov logic network $Y_{L,C}$ can be defined as:*

$$P(X = x) = \frac{1}{Z} exp\left(\sum_{i=1}^{|G|} w_i \pi_x(g_i)\right)$$

where

- *$|G|$ is the number of grounding,*
- *$\pi_i(x)$ evaluates the grounded formula applying the fuzzy logic calculus,*
- *w_i is a weight associated to the grounded formula g_i.*

The fuzzy calculus is applied only during inference. During learning the fuzzy predicates are treated like first-order predicates. A fuzzy predicate is a predicate which truth value is represented as a value between 0 and 1, where 0 equals ***false*** and 1 equals ***true***.

One advantage of fuzzy MLN is that it enables the use of taxonomic knowledge. Due to this property it is possible to represent these MLNs more compactly. These fuzzy MLNs are used in PRAC to perform the action roles and senses inference. How such MLNs are designed is described in Section 2.5.

2.4 INTEGRATED TOOLS

In this section we describe the additional tools which are supporting the action role inference in PRAC. Since the joint probability distributions of PRAC are used to perform the information extraction of natural-language documents, it is necessary to provide an overview about these components.

2.4.1 STANFORD PARSER

For the information extraction we use the Stanford Parser to process a given sentence. The Stanford Parser is a PCFG dependency parser [20]. In addition to the grammatical relations, the parser provides part-of-speech tags to the words in a given sentence. Instead of just tagging if a word is a noun or a verb, this parser uses Penn Treebank part-of-speech tags. With Penn Treebank POS it is possible to achieve a more detailed classification. For example, there exist 6 tags for tagging a verb - e.g. the tag *VBG* means the verb is in past tense. A list of all tags is given in [4].

Following reasons motivated us to apply this parser:

Used in PRAC The Standford Parser has been already used in PRAC. It has proven successful that this parser performs well on imperative sentences and other types of sentences. Additionally, modules to interact with the Standford Parser and the probabilistic models of PRAC exist. Since we are intending to use these models to extract information, it is reasonable to use the provided modules instead implementing new ones for a different parser.

Trained with various corpora The Standford Parser is trained with multiple corpora. These corpora include (amongst others) hundreds of imperative sentences [5]. This provides a great foundation to extract information of natural-language instructions.

No current parser is perfect The sentence "season the chicken with pepper" provides difficulties to current parsers[3] [4]. Tagging the word "season" as noun is a common error. The Stanford Parser tags the word incorrect too but we have developed a set of workarounds to handle this type of error (see Section 3.3.1.1).

[3]http://demo.ark.cs.cmu.edu/parse?
[4]http://babelfy.org

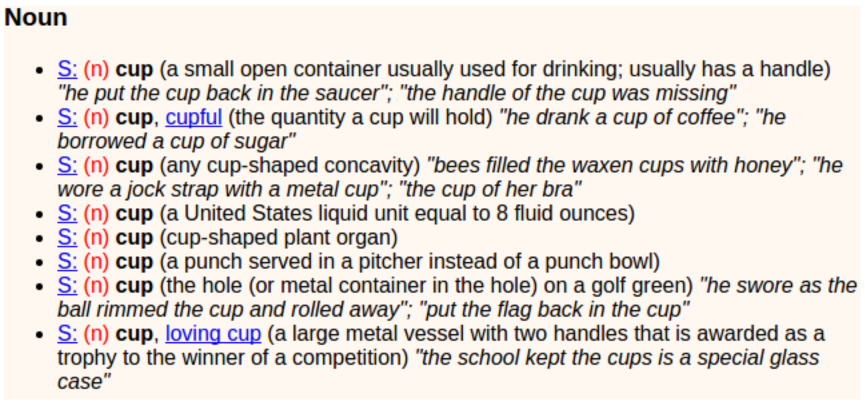
2.4.2 WORDNET

As mentioned in Section 1.2, PRAC uses WordNet to annotated words in a given natural-language instruction with senses and since we are using these senses also to perform the semantic information retrieval, we describe WordNet in this section.

WordNet is a lexical database. It groups nouns, verbs and adjectives into synsets. A synset contains words which represent the same concept. Meaning, these words are interchangeable without altering the semantic of a sentence [6].

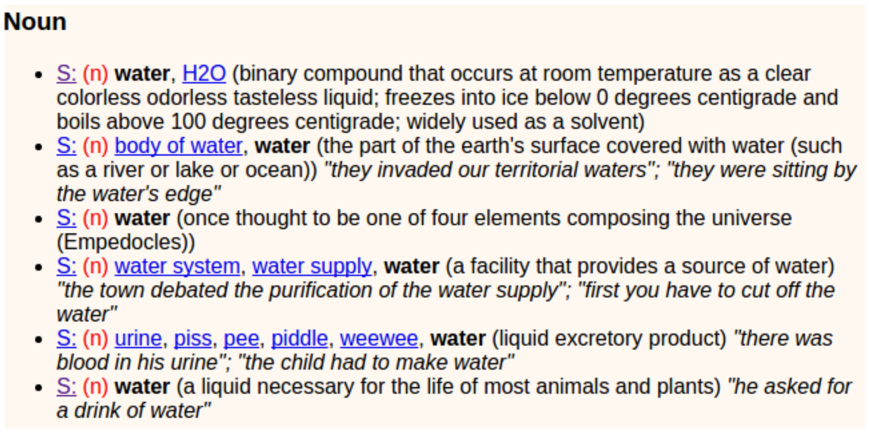
WordNet represents these synsets in a taxonomy. For instance, the synset of the word "water" has an is-a relation to the more general synset "liquid". Consider the sentence "fill a cup with water". Querying WordNet for the words "cup" and "water" will return 8 synsets for "cup" and 6 for "water". The corresponding synsets are depicted in Figure 2.8 and Figure 2.9. It is also possible to query synsets for verbs. These verbs are also represented in a taxonomy. To keep this example short, we are not going into detail about the correct synset for "fill".

Based on our example, for most people the first synset of "cup" is correct and for "water" the last (sixth) one. PRAC uses the NLTK notation to link the words in a sentence to synsets. There are two notations to represent a specific synset of WordNet. One representation is the Synset ID which is a sequence of numbers. Another representation is the NLTK notation. Regarding our mentioned "fill a cup with water" example the NLTK notation for the correct senses of "cup" and "water" is "cup.n.01" and "water.n.06". To represent a synset which is a verb, we can use e.g. "fill.v.01". The benefit of the NLTK notation is that it does not change by an update of WordNet. This means e.g. that "water.n.06" will still represent the synset which describes drinking water. However, the Synset ID changes after an WordNet update and that means that an older ID can map to a completely different synset as in the previous version. This leads to the negative effect that the used synsets (e.g. in the training set) have to be updated too.


Noun

- S: (n) **cup** (a small open container usually used for drinking; usually has a handle) *"he put the cup back in the saucer"; "the handle of the cup was missing"*
- S: (n) **cup**, cupful (the quantity a cup will hold) *"he drank a cup of coffee"; "he borrowed a cup of sugar"*
- S: (n) **cup** (any cup-shaped concavity) *"bees filled the waxen cups with honey"; "he wore a jock strap with a metal cup"; "the cup of her bra"*
- S: (n) **cup** (a United States liquid unit equal to 8 fluid ounces)
- S: (n) **cup** (cup-shaped plant organ)
- S: (n) **cup** (a punch served in a pitcher instead of a punch bowl)
- S: (n) **cup** (the hole (or metal container in the hole) on a golf green) *"he swore as the ball rimmed the cup and rolled away"; "put the flag back in the cup"*
- S: (n) **cup**, loving cup (a large metal vessel with two handles that is awarded as a trophy to the winner of a competition) *"the school kept the cups is a special glass case"*

Figure 2.8: Cup synsets

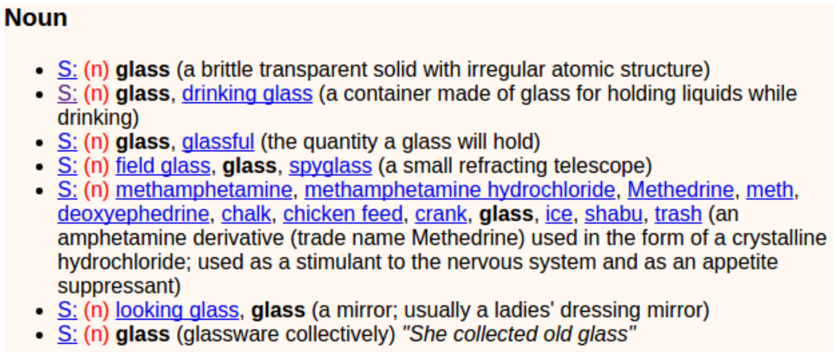


Noun

- S: (n) **water**, H2O (binary compound that occurs at room temperature as a clear colorless odorless tasteless liquid; freezes into ice below 0 degrees centigrade and boils above 100 degrees centigrade; widely used as a solvent)
- S: (n) body of water, **water** (the part of the earth's surface covered with water (such as a river or lake or ocean)) *"they invaded our territorial waters"; "they were sitting by the water's edge"*
- S: (n) **water** (once thought to be one of four elements composing the universe (Empedocles))
- S: (n) water system, water supply, **water** (a facility that provides a source of water) *"the town debated the purification of the water supply"; "first you have to cut off the water"*
- S: (n) urine, piss, pee, piddle, weewee, **water** (liquid excretory product) *"there was blood in his urine"; "the child had to make water"*
- S: (n) **water** (a liquid necessary for the life of most animals and plants) *"he asked for a drink of water"*

Figure 2.9: Water synsets

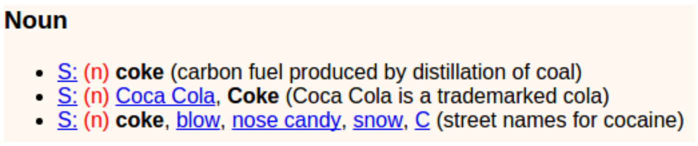
To handle the WSD problem in the action roles inference, PRAC get such instructions like "fill a cup with water" as training examples. Since the joint probability distributions can make use of the taxonomy, the system can determine the correct sense even to unseen samples. Given a task such as "fill a glass with coke" (Figure 2.10 and Figure 2.11 show the synsets), the system infers the correct sense just by taking the similarities between the synsets in the instruction and the synsets in the training set as evidence.



Noun

- S: (n) **glass** (a brittle transparent solid with irregular atomic structure)
- S: (n) **glass**, drinking glass (a container made of glass for holding liquids while drinking)
- S: (n) **glass**, glassful (the quantity a glass will hold)
- S: (n) field glass, **glass**, spyglass (a small refracting telescope)
- S: (n) methamphetamine, methamphetamine hydrochloride, Methedrine, meth, deoxyephedrine, chalk, chicken feed, crank, **glass**, ice, shabu, trash (an amphetamine derivative (trade name Methedrine) used in the form of a crystalline hydrochloride; used as a stimulant to the nervous system and as an appetite suppressant)
- S: (n) looking glass, **glass** (a mirror; usually a ladies' dressing mirror)
- S: (n) **glass** (glassware collectively) *"She collected old glass"*

Figure 2.10: Glass synsets



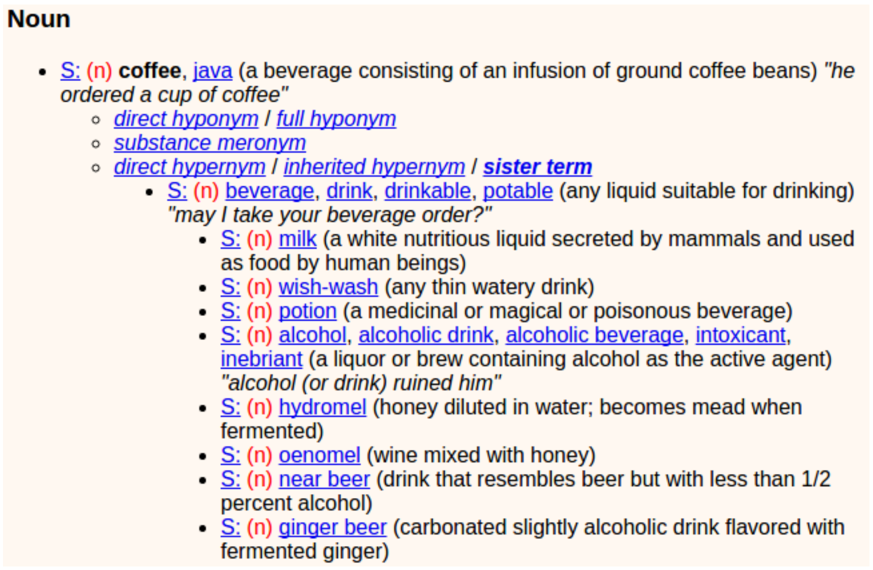
Noun

- S: (n) **coke** (carbon fuel produced by distillation of coal)
- S: (n) Coca Cola, **Coke** (Coca Cola is a trademarked cola)
- S: (n) **coke**, blow, nose candy, snow, C (street names for cocaine)

Figure 2.11: Coke synsets

2.4.2.1 Similarity Measures

During this thesis, we use two measures to determine the similarity between concepts - path similarity and WUP similarity. The reason for that is that due to the different calculation methods, the resulting similarity values between sister terms differ. Sister terms are a set of synsets which share the same parent synset. For instance, the sister terms of "coffee" are "milk" and "ginger beer" because they share the same super-concept "beverage" (see Figure 2.12).



Noun

- S: (n) **coffee**, java (a beverage consisting of an infusion of ground coffee beans) *"he ordered a cup of coffee"*
  - *direct hyponym* / *full hyponym*
  - *substance meronym*
  - *direct hypernym* / *inherited hypernym* / ***sister term***
    - S: (n) beverage, drink, drinkable, potable (any liquid suitable for drinking) *"may I take your beverage order?"*
      - S: (n) milk (a white nutritious liquid secreted by mammals and used as food by human beings)
      - S: (n) wish-wash (any thin watery drink)
      - S: (n) potion (a medicinal or magical or poisonous beverage)
      - S: (n) alcohol, alcoholic drink, alcoholic beverage, intoxicant, inebriant (a liquor or brew containing alcohol as the active agent) *"alcohol (or drink) ruined him"*
      - S: (n) hydromel (honey diluted in water; becomes mead when fermented)
      - S: (n) oenomel (wine mixed with honey)
      - S: (n) near beer (drink that resembles beer but with less than 1/2 percent alcohol)
      - S: (n) ginger beer (carbonated slightly alcoholic drink flavored with fermented ginger)

Figure 2.12: Sister terms of the synset 'coffee'

The path similarity calculates the similarity between two synsets by dividing 1 by the length of the shortest path between these synsets [3]. Whereat the WUP is defined as follows (consider Figure 2.13):

$$WUP(C1, C2) = \frac{2 * N3}{N1 + N2 + 2 * N3}$$

where $C3$ is the least common super-concept of $C1$ and $C2$ and Ni is the number of nodes between the corresponding start and end nodes [32].

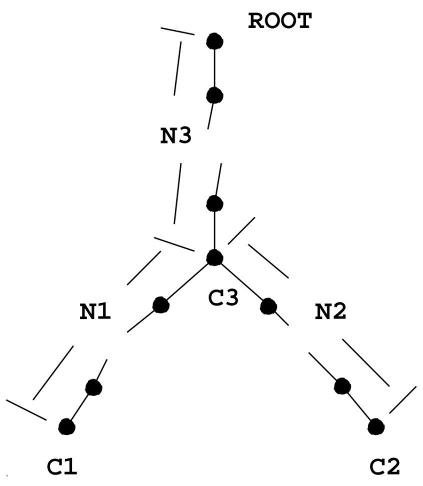


Figure 2.13: Graphical concept of the WUP similarity [32]

Regarding the sister terms, the path similarity returns always the value 0.33. This can be explained because the shortest path equals always 3. The shortest path consists of the two sister terms and their shared parent node. However, the WUP similarity between sister terms varies because the length of $N3$ is variable. In addition, WUP similarity rates the similarity between specific sister terms higher than sister terms between more abstract sister terms[32].

It turns out that the inference results for the roles and senses are more accurate if evidences are determined via path similarity. A more detailed explanation about this behavior is given in Section 2.5.4. For the information retrieval it is more reasonable to use the WUP similarity. The reason for that are stated in Section 3.4.1.

2.5 A Detailed Example of the Action Role Inference

Since the technical foundation is provided to be able to understand the action role inference process, we describe in this section the necessary procedures to be able to infer the action roles of the sentence "fill mug with tea".

2.5.1 Parsing the Given Instruction with the Stanford Parser

Before PRAC performs any kind of inference, every given instruction is parsed by the Stanford Parser. The recognized dependencies and part-of-speech tags are represented as ground atoms. These ground atoms are stored in a database. Figure 2.14 shows the Stanford Parser results of the sentence "fill a mug with tea" represented as a database.

```
1.00   det(mug−3, a−2)
1.00   dobj(Fill −1, mug−3)
1.00   prep_with(Fill −1, tea −5)
1.00   has_pos(Fill −1, VB)
1.00   has_pos(a−2, DT)
1.00   has_pos(mug−3, NN)
1.00   has_pos(tea −5, NN)
1.00   has_pos(with−4, IN)
```

Figure 2.14: The Stanford Parser results of the sentence "fill a mug with tea"

The predicates det, $dobj$ and $prep\_with$ represent the dependencies between the words. For instance, $dobj(\textit{Fill-1}, \textit{mug-3})$ represents that "mug" is the direct object of the verb "fill". All supported dependencies by the Stanford Parser are given in [13]. The part-of-speech tag mapping is expressed by the predicate has_pos. In this example, "tea" is mapped to the Penn Treebank POS "NN".

Every ground atom in the database has a truth value attached. Since has_pos and the dependency predicates are first-order predicates, they can only be asserted with the truth value 0 or 1. During the inference processes, these predicates can be treated with the closed-world assumption because every word in the sentence can only be asserted with one part-of-speech tag and every dependency which exists between those words is captured by the Stanford Parser. Referring to the example, if a Markov logic network uses the has_pos atoms as evidence, then the combination e.g. $has\_pos(\textit{mug-3}, DT)$ is treated with the truth value 0 since it is not defined in the database.

2.5.2 Action Core Inference

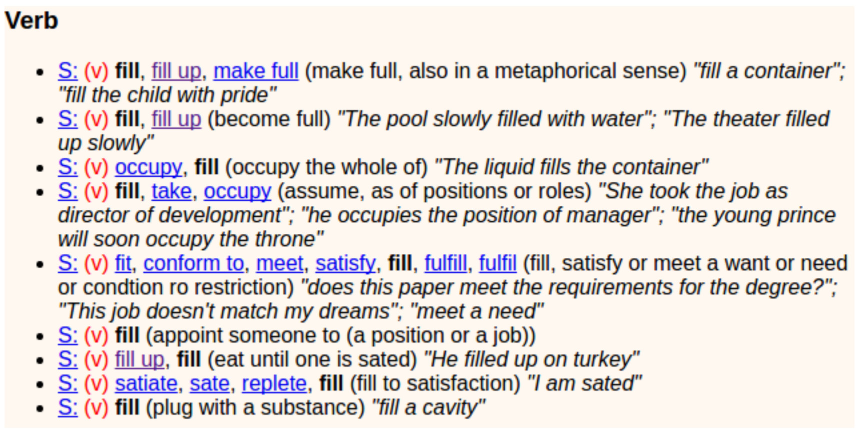
After the parsing process, PRAC loads the MLN which performs the action core recognition. A part of this MLN is represented in Figure 2.15. We are not able to provide the complete MLN since it contains over 200 weighted formulas. Before we start to explain the design of this Markov logic network, we need to describe the predicates $is\_a(sense, concept)$ and $has\_sense(word, sense)$. These predicates are required for the sense inference. However, they can also be used to determine the most probable action core of a given instruction. The is_a predicate is defined as a fuzzy predicate. This means, that this predicate can be asserted with truth values between 0 and 1. The domain $concept$ represents the synsets which occurred during training and $sense$ the synsets during the inference task. In other words, this predicate allows us to utilize the taxonomic knowledge of WordNet. To utilize this knowledge, we apply the path similarity between the synsets of the $concept$ domain and synsets of the $sense$ domain. The similarity values are represented as truth values of the corresponding grounded is_a atoms. These atoms are used as evidence for the action core and action role inference.

```
.....

3.621935    has_pos(?w1, VB) ^ is_a(?s1, fill.v.01) ^ has_sense(?w1, ?s1)
            ^ action_core(?w1, Filling)

−0.318413   has_pos(?w1, VB) ^ is_a(?s1, slice.v.03) ^ has_sense(?w1, ?s1)
            ^ action_core(?w1, Filling)

−0.318413   has_pos(?w1, VB) ^ is_a(?s1, season.v.01) ^ has_sense(?w1, ?s1)
            ^ action_core(?w1, Filling)

−0.318413   has_pos(?w1, VB) ^ is_a(?s1, fill.v.01) ^ has_sense(?w1, ?s1)
            ^ action_core(?w1, Slicing)
......
```

Figure 2.15: Part of the MLN to infer the action core in a given instruction

Referring back to Figure 2.15, the represented MLN contains formulas that describe which verb activates which action core. For instance, the first formula represents that the verb which has a high similarity to the synset "fill.v.01" is most likely to activate the 'Filling' action core. However, the second formula represents that the verb which has a high similarity to the synset "slice.v.03" does not activate the 'Filling' action core.

To be able to utilize the knowledge of the action core MLN it is necessary to extend the database of the parsing process with the similarities between the synsets of the instruction and the synsets contained in the MLN. To achieve this, PRAC queries all synsets for the verb contained in the database. In the "fill a mug with tea" sentence the verb is "fill". Figure 2.16 shows all possible synsets for "fill". The extended database is depicted in Figure 2.17. For the sake of simplicity we only represent the similarities for two synsets in this figure.



Verb

- S: (v) **fill**, fill up, make full (make full, also in a metaphorical sense) *"fill a container"; "fill the child with pride"*
- S: (v) **fill**, fill up (become full) *"The pool slowly filled with water"; "The theater filled up slowly"*
- S: (v) occupy, **fill** (occupy the whole of) *"The liquid fills the container"*
- S: (v) **fill**, take, occupy (assume, as of positions or roles) *"She took the job as director of development"; "he occupies the position of manager"; "the young prince will soon occupy the throne"*
- S: (v) fit, conform to, meet, satisfy, **fill**, fulfill, fulfil (fill, satisfy or meet a want or need or condtion ro restriction) *"does this paper meet the requirements for the degree?"; "This job doesn't match my dreams"; "meet a need"*
- S: (v) **fill** (appoint someone to (a position or a job))
- S: (v) fill up, **fill** (eat until one is sated) *"He filled up on turkey"*
- S: (v) satiate, sate, replete, **fill** (fill to satisfaction) *"I am sated"*
- S: (v) **fill** (plug with a substance) *"fill a cavity"*

Figure 2.16: Fill synsets

```
1.00  det(mug-3, a-2)
1.00  dobj(Fill-1, mug-3)
1.00  prep_with(Fill-1, tea-5)
1.00  has_pos(Fill-1, VB)
1.00  has_pos(a-2, DT)
1.00  has_pos(mug-3, NN)
1.00  has_pos(tea-5, NN)
1.00  has_pos(with-4, IN)
//Represents synset: fill,fill up, make full
1.00  is_a(fill.v.01, fill.v.01)
0.00  is_a(fill.v.01, season.v.01)
0.00  is_a(fill.v.01, slice.v.03)
//Represents synset: fill up, fill (eat until one is sated)
0.00  is_a(fill_up.v.04, fill.v.01)
0.00  is_a(fill_up.v.04, season.v.01)
0.00  is_a(fill_up.v.04, slice.v.03)
```

Figure 2.17: Evidence database for action core inference

Given that evidence database, it is possible to ground the action core MLN. This grounded MLN is depicted in Figure 2.18. The showed weights for each grounded formula are determined by multiplying the truth value of the is_a atom contained in the evidence database and the weight of the ungrounded formula[5]. This grounded MLN can be transformed to a WCSP and a WCSP solver determines the most probable action core. In our example the solver returns the action core 'Filling'.

```
.....

3.621935   has_pos(Fill-1, VB) ^ is_a(fill.v.01, fill.v.01)
           ^ has_sense(Fill-1, fill.v.01) ^ action_core(Fill-1, Filling)

0          has_pos(Fill-1, VB) ^ is_a(fill.v.01, slice.v.03)
           ^ has_sense(Fill-1, fill.v.01) ^ action_core(Fill-1, Filling)

0          has_pos(Fill-1, VB) ^ is_a(fill.v.01, season.v.01)
           ^ has_sense(Fill-1, fill.v.01) ^ action_core(Fill-1, Filling)

-0.318413  has_pos(Fill-1, VB) ^ is_a(fill.v.01, fill.v.01)
           ^ has_sense(Fill-1, fill.v.01) ^ action_core(Fill-1, Filling)

0          has_pos(Fill-1, VB) ^ is_a(fill_up.v.04, fill.v.01)
           ^ has_sense(Fill-1, fill_up.v.04) ^ action_core(Fill-1, Filling)

0          has_pos(Fill-1, VB) ^ is_a(fill_up.v.04, slice.v.03)
           ^ has_sense(Fill-1, fill_up.v.04) ^ action_core(Fill-1, Filling)

0          has_pos(Fill-1, VB) ^ is_a(fill_up.v.04, season.v.01)
           ^ has_sense(Fill-1, fill_up.v.04) ^ action_core(Fill-1, Filling)

0          has_pos(Fill-1, VB) ^ is_a(fill_up.v.04, fill.v.01)
           ^ has_sense(Fill-1, fill_up.v.04) ^ action_core(Fill-1, Filling)
.....
```

Figure 2.18: The grounded action core MLN

[5]This calculation is performed based on the fuzzy calculus used in fuzzy MLNs (see Section 2.3.3)

2.5.3 ACTION ROLE INFERENCE

After PRAC determined the action core, it loads the corresponding MLN to infer the action roles. In our example PRAC would use the 'Filling' MLN to infer the roles and sense. A part of this MLN is represented in Figure 2.19.

There are three roles which are attached to the 'Filling' action core - ***action verb***, ***stuff*** and ***goal***, where ***action verb*** represents the word which activated the action core, ***stuff*** represents the object which should be filled and ***goal*** represents the container. These roles are represented as predicates in the Filling MLN. The predicates have the form $role\_name(word, action\_core)$, where $word$ represents the word in a given sentence and $action\_core$ the corresponding action core. For every formula which contains has_sense as a predicate has one action role predicate attached. This has the advantage that PRAC can perform the senses and roles inferences simultaneously.

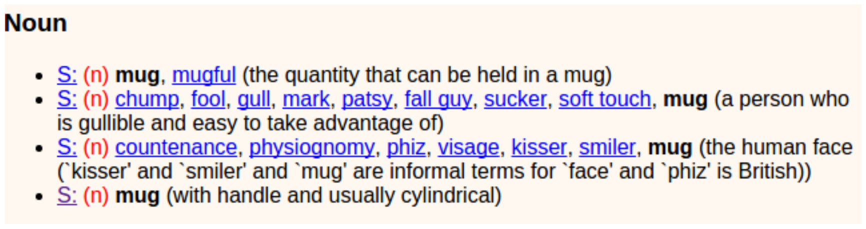
```
....

−0.518182   action_core(?w3, ?ac) ^ goal(?w1, ?ac) ^ stuff(?w2, ?ac)
            ^ has_sense(?w1, ?s1) ^ is_a(?s1, fill.v.01) ^ has_sense(?w2, ?s2)
            ^ is_a(?s2, fruit_juice.n.01) ^ ?w1=/=?w2

0.704993    action_core(?w3, ?ac) ^ action_verb(?w2, ?ac) ^ stuff(?w1, ?ac)
            ^ has_sense(?w1, ?s1) ^ is_a(?s1, fruit_juice.n.01)
            ^ has_sense(?w2, ?s2) ^ is_a(?s2, fill.v.01) ^ ?w1=/=?w2

0.686252    action_core(?w3, ?ac) ^ goal(?w1, ?ac) ^ stuff(?w2, ?ac)
            ^ has_sense(?w1, ?s1) ^ is_a(?s1, bottle.n.01) ^ has_sense(?w2, ?s2)
            ^ is_a(?s2, fruit_juice.n.01) ^ ?w1=/=?w2
....
```

Figure 2.19: The 'Filling' MLN to perform the action role inference

Figure 2.19 shows part of the formulas which represent the training instruction "fill a bottle with juice". The first formula represents that a word which is similar to the concept "fruit_juice.n.01" cannot be asserted to the role ***stuff*** if in the same time the verb is asserted to the role ***goal***. However, the second formula represents that a word which is similar to the concept "fruit_juice.n.01" can be asserted to the role ***stuff*** if the verb is asserted to the role ***action verb***.

The general inference procedures are the same as in the action core inference. First, PRAC queries the synsets for "fill", "mug" and "tea" which are depicted in Figure 2.16, 2.20 and 2.21. Based on these synsets, PRAC extends the database with the corresponding similarities. A part of the evidence database is depicted in Figure 2.22.

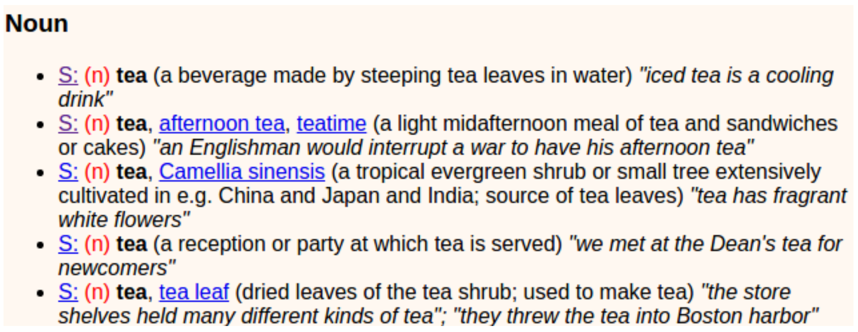
Given that evidence database, the Filling MLN can be grounded and then transformed into a WCSP. The inference results for the query predicates $goal$, $action\_verb$, $stuff$ and has_sense for this example are depicted in Figure 2.23.



Noun

- S: (n) **mug**, mugful (the quantity that can be held in a mug)
- S: (n) chump, fool, gull, mark, patsy, fall guy, sucker, soft touch, **mug** (a person who is gullible and easy to take advantage of)
- S: (n) countenance, physiognomy, phiz, visage, kisser, smiler, **mug** (the human face (`kisser' and `smiler' and `mug' are informal terms for `face' and `phiz' is British))
- S: (n) **mug** (with handle and usually cylindrical)

Figure 2.20: Mug synsets



Noun

- S: (n) **tea** (a beverage made by steeping tea leaves in water) *"iced tea is a cooling drink"*
- S: (n) **tea**, afternoon tea, teatime (a light midafternoon meal of tea and sandwiches or cakes) *"an Englishman would interrupt a war to have his afternoon tea"*
- S: (n) **tea**, Camellia sinensis (a tropical evergreen shrub or small tree extensively cultivated in e.g. China and Japan and India; source of tea leaves) *"tea has fragrant white flowers"*
- S: (n) **tea** (a reception or party at which tea is served) *"we met at the Dean's tea for newcomers"*
- S: (n) **tea**, tea leaf (dried leaves of the tea shrub; used to make tea) *"the store shelves held many different kinds of tea"; "they threw the tea into Boston harbor"*

Figure 2.21: Tea synsets

```
1.00  action_core(Fill-1, Filling)
1.00  det(mug-3, a-2)
1.00  dobj(Fill-1, mug-3)
1.00  prep_with(Fill-1, tea-5)
1.00  has_pos(Fill-1, VB)
1.00  has_pos(a-2, DT)
1.00  has_pos(mug-3, NN)
1.00  has_pos(tea-5, NN)
1.00  has_pos(with-4, IN)

//Represents synset: mug, mugful
0.07  is_a(mug.n.01,bottle.n.01)
0.08  is_a(mug.n.01,fruit_juice.n.01)

//Represents synset: mug
0.07  is_a(mug.n.04,fruit_juice.n.01)
0.25  is_a(mug.n.04,bottle.n.01)

//Represents synset: tea (a beverage made by steeping tea leaves in water)
0.33  is_a(tea.n.01,fruit_juice.n.01)
0.08  is_a(tea.n.01,bottle.n.01)

//Represents synset: tea, afternoon tea, teatime
0.17  is_a(tea.n.02,fruit_juice.n.01)
0.07  is_a(tea.n.02,fruit_juice.n.01)
```

Figure 2.22: Evidence database for action role inference

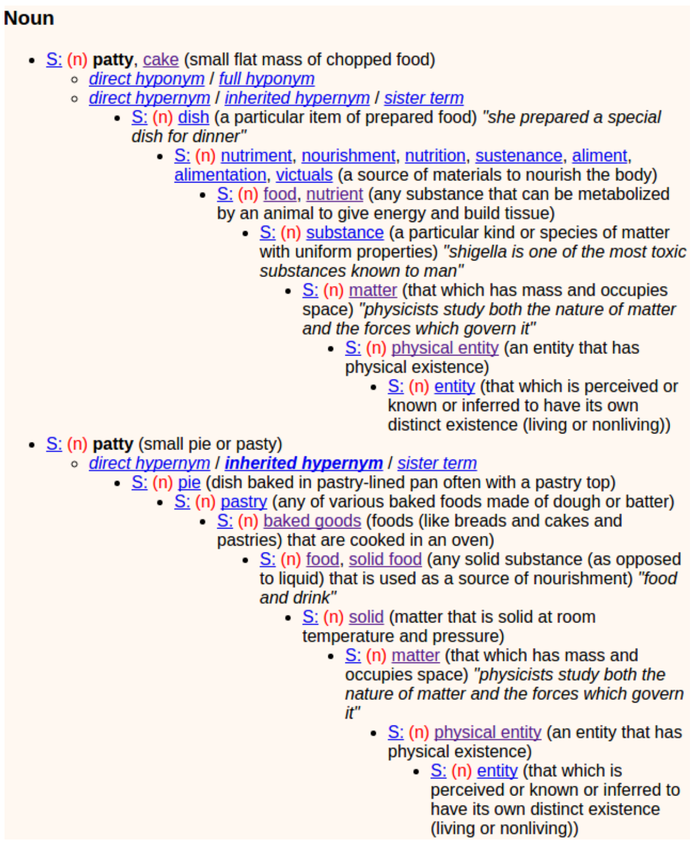
```
1.00      has_sense(tea-5,tea.n.01)
1.00      has_sense(Fill-1,fill.v.01)
1.00      has_sense(mug-3,mug.n.04)
1.00      goal(mug-3,Filling)
1.00      stuff(tea-5,Filling)
1.00      action_verb(Fill-1,Filling)
```

Figure 2.23: The results of the inference represented as ground atoms

2.5.4 Motivation for Path Similarity

In this section we want to explain why we use the path similarity for the action core and role inference. We discovered that the inference fails if fuzzy MLNs are trained with an imbalanced training set. Using the path similarity instead of the WUP similarity allows us to cope with the imbalanced data problem regarding fuzzy MLNs. We are intending to provide an example in this section which explains how an imbalanced data set can look like and why the path similarity is a more reasonable choice compared to the WUP similarity to perform the action role inference. The presented example is an artificial example but should be complex enough to introduce the definition of an imbalanced training set and the advantage of using the path similarity.

Consider a fuzzy MLN which should be able to infer the correct senses for words of the food domain. For instance, this model should infer the correct senses for the words "pizza", "paella" or "salad". In this case, the senses for these words are easy to infer since they activate only one synset in WordNet. Now consider the word "patty". In WordNet this word activates 3 synsets where each represents a concept of the food domain. For the sake of simplicity, we consider a scenario where "patty" activates only 2 synsets. Figure 2.24 shows the possible synsets.



Noun

- S: (n) **patty**, cake (small flat mass of chopped food)
  - direct hyponym / full hyponym
  - direct hypernym / inherited hypernym / sister term
    - S: (n) dish (a particular item of prepared food) *"she prepared a special dish for dinner"*
      - S: (n) nutriment, nourishment, nutrition, sustenance, aliment, alimentation, victuals (a source of materials to nourish the body)
        - S: (n) food, nutrient (any substance that can be metabolized by an animal to give energy and build tissue)
          - S: (n) substance (a particular kind or species of matter with uniform properties) *"shigella is one of the most toxic substances known to man"*
            - S: (n) matter (that which has mass and occupies space) *"physicists study both the nature of matter and the forces which govern it"*
              - S: (n) physical entity (an entity that has physical existence)
                - S: (n) entity (that which is perceived or known or inferred to have its own distinct existence (living or nonliving))
- S: (n) **patty** (small pie or pasty)
  - direct hypernym / ***inherited hypernym*** / sister term
    - S: (n) pie (dish baked in pastry-lined pan often with a pastry top)
      - S: (n) pastry (any of various baked foods made of dough or batter)
        - S: (n) baked goods (foods (like breads and cakes and pastries) that are cooked in an oven)
          - S: (n) food, solid food (any solid substance (as opposed to liquid) that is used as a source of nourishment) *"food and drink"*
            - S: (n) solid (matter that is solid at room temperature and pressure)
              - S: (n) matter (that which has mass and occupies space) *"physicists study both the nature of matter and the forces which govern it"*
                - S: (n) physical entity (an entity that has physical existence)
                  - S: (n) entity (that which is perceived or known or inferred to have its own distinct existence (living or nonliving))

Figure 2.24: Patty synsets

The first synset of "patty" represents a sense which is of the dish domain. However, the second synsets represents a sense of the pie domain. For this example, we want that the MLN should infer the first synset for the word "patty". To do this, we can represent this information in a training set where the word "patty" is mapped to the synset "patty.n.01". Since the MLN should be able to recognize a range of different foods, we can extend this training set with additional samples. For instance, we can add the words "apple pie", "blueberry pie" and "pumpkin pie" with the senses "apple_pie.n.01", "blueberry_pie.n.01", "pumpkin_pie.n.01" to the training set. We have to mention that these synsets are concepts of pie domain. More specific, they are sister terms of "patty.n.02". Without any additional samples, this training set is considered as imbalanced since it contains 1 entity of the dish domain and 3 entities of the pie domain.

Figure 2.25 defines a MLN which is trained with this set. This MLN is able to infer the senses for words of the food domain. To keep this example simple, the weights of the formulas are set by ourself instead of using actual trained weights.

```
1 is_a(?s1,apple_pie.n.01) ^ has_sense(?w1,?s1)
1 is_a(?s1,blueberry_pie.n.01) ^ has_sense(?w1,?s1)
1 is_a(?s1,pumpkin_pie.n.01) ^ has_sense(?w1,?s1)
1 is_a(?s1,patty.n.01) ^ has_sense(?w1,?s1)
```

Figure 2.25: A MLN trained with the imbalanced training set

Now consider, we want to perform a sense inference for the word "patty". First, we show the inference results for using the path similarity between the concepts in the MLN and in the evidence database. Afterwards, we perform the same inference but we use the WUP similarity instead of the path similarity.

Figure 2.26 shows the evidence database. This database contains is_a atoms which truth values are determined with the path similarity. Figure 2.27 shows the grounded MLN based on the given evidence. If we sum all weights of the formulas which contain $has\_sense(patty, patty.n.01)$, we get a value of 1.24. Calculating the weights of the formulas which contain $has\_sense(patty, patty.n.02)$ results in 1.07[6]. In this scenario, the fuzzy MLN would infer the expected sense for the word "patty".

```
1.00  is_a(patty.n.01, patty.n.01)
0.08  is_a(patty.n.01, apple_pie.n.01)
0.08  is_a(patty.n.01, blueberry_pie.n.01)
0.08  is_a(patty.n.01, pumpkin_pie.n.01)
0.08  is_a(patty.n.02, patty.n.01)
0.33  is_a(patty.n.02, apple_pie.n.01)
0.33  is_a(patty.n.02, blueberry_pie.n.01)
0.33  is_a(patty.n.02, pumpkin_pie.n.01)
```

Figure 2.26: Is_a atoms based on the path similarity

```
0.08  is_a(patty.n.01,apple_pie.n.01) ^ has_sense(patty,patty.n.01)
0.08  is_a(patty.n.01,blueberry_pie.n.01) ^ has_sense(patty,patty.n.01)
0.08  is_a(patty.n.01,pumpkin_pie.n.01) ^ has_sense(patty,patty.n.01)
0.33  is_a(patty.n.02,apple_pie.n.01) ^ has_sense(patty,patty.n.02)
0.33  is_a(patty.n.02,blueberry_pie.n.01) ^ has_sense(patty,patty.n.02)
0.33  is_a(patty.n.02,pumpkin_pie.n.01) ^ has_sense(patty,patty.n.02)
1.00  is_a(patty.n.01,patty.n.01) ^ has_sense(patty,patty.n.01)
0.08  is_a(patty.n.02,patty.n.01) ^ has_sense(patty,patty.n.02)
```

Figure 2.27: Grounded MLN with considering the path similarity

Now we perform the same inference but instead of using the path similarity we use the WUP similarity. Figure 2.28 shows the is_a atoms for the same query and Figure 2.29 the grounded MLN with the corresponding weights. In this case, the inference does not return the expected sense for the word "patty". Calculating the weights of the formulas which contain $has\_sense(patty, patty.n.01)$ results in 2.05. However, the sum of the weights of the formulas with $has\_sense(patty, patty.n.02)$ results in 3.02.

[6]We argument with the calculation of a probability distribution of a grounded Markov logic network (see Section 2.3.2). It provides a more intuitive of understanding compared to represent this example as a WCSP

```
1.00   is_a(patty.n.01, patty.n.01)
0.35   is_a(patty.n.01, apple_pie.n.01)
0.35   is_a(patty.n.01, blueberry_pie.n.01)
0.35   is_a(patty.n.01, pumpkin_pie.n.01)
0.35   is_a(patty.n.02, patty.n.01)
0.89   is_a(patty.n.02, apple_pie.n.01)
0.89   is_a(patty.n.02, blueberry_pie.n.01)
0.89   is_a(patty.n.02, pumpkin_pie.n.01)
```

Figure 2.28: Is_a atoms based on the WUP similarity

```
0.35   is_a(patty.n.01,apple_pie.n.01) ^ has_sense(patty,patty.n.01)
0.35   is_a(patty.n.01,blueberry_pie.n.01) ^ has_sense(patty,patty.n.01)
0.35   is_a(patty.n.01,pumpkin_pie.n.01) ^ has_sense(patty,patty.n.01)
0.89   is_a(patty.n.02,apple_pie.n.01) ^ has_sense(patty,patty.n.02)
0.89   is_a(patty.n.02,blueberry_pie.n.01) ^ has_sense(patty,patty.n.02)
0.89   is_a(patty.n.02,pumpkin_pie.n.01) ^ has_sense(patty,patty.n.02)
1.00   is_a(patty.n.01,patty.n.01) ^ has_sense(patty,patty.n.01)
0.35   is_a(patty.n.02,patty.n.01) ^ has_sense(patty,patty.n.02)
```

Figure 2.29: Grounded MLN with considering the WUP similarity

To summarize, it is recommended to provide a balanced training set to learn fuzzy Markov logic networks. We define a training set as balanced if the training set contains objects of multiple domains and the number of these objects are nearly the same. If this is not provided, the inference results can provide unexpected results. Using the path similarity for the inference allows us to cope with small imbalanced training sets.

Chapter three

Inferring Missing Information through Semantic Information Retrieval

In this chapter we present a solution to infer missing information through semantic information retrieval. We start with presenting the joint probability distributions which are used to extract knowledge of natural-language sentences. Afterwards, we introduce the knowledge representation to store the extracted knowledge. Then we continue with describing the information extraction process. At the end of this chapter, we describe the semantic information retrieval algorithm which completes the robot instructions which are given to PRAC.

3.1 Developing Joint Probability Distributions to Extract Knowledge of Natural-language documents

As mentioned in Chapter 1, we are intending to provide a database which stores extracted information from natural-language documents. With this database, we want to overcome the scaling problem of joint probability distributions but still be able to perform inference of missing information. However, natural-language documents are unstructured data and therefore it requires a solution to structure this data so it can be utilized by PRAC. It would be reasonable to annotate the sentences in the corpus with the action core and the corresponding action roles which these sentences are representing. This annotation can be performed by the same fuzzy MLNs which are used by PRAC to identify the action core and roles in a given natural-language instruction. After the action role inference, PRAC is able to determine which required roles are missing to execute the plan successfully. If some roles are missing in the given instruction, PRAC can search in the database for sentences which activate the same action core and contain the missing action roles. For example, given a document which contains the sentence "flavor the chicken with pepper", a successful action role inference results that this sentence activates the 'Flavoring' action core with "chicken" as the ***goal*** parameter and "pepper" as the ***spice*** parameter. Given the natural-language instruction "flavor the rib", PRAC can determine that the action role ***spice*** is missing. Since the sentence "flavor the chicken with pepper" is stored in the database and it is annotated with the same action core as the given instruction, its value of the ***spice*** parameter can be used to complete the instruction.

For this thesis, we consider a knowledge base of four action cores - 'Flavoring', 'Neutralizing', 'Flipping' and 'Storing'. PRAC has already supported 'Flavoring' and 'Neutralizing', however we updated their model design during this thesis to speed up the inference process. 'Flipping' and 'Storing' were designed during this thesis. The 'Neutralizing' action core is representing an action of the chemical domain. The remaining three action cores are representing actions of the kitchen domain. In the evaluation we show that our solution for the missing role inference can perform in multiple domains. The reason why we decided to use these action cores is that the modeling and training process of their distributions is simpler compared to generic actions such as 'Adding'. So for instance, given the information that the robot has to perform a neutralization task in a chemical domain, the objects appearing in this context are most likely to be chemical substances. A more generic action core such as 'Adding' is much more difficult to train. Sentences like "add some water to the mixture", "add some salt to the soup" and "add some hydrochloric acid to the sodium hydroxide" show that the fuzzy MLNs for generic action cores have to handle objects of multiple domains. Due to the arguments with the path similarity (see Section 2.5.4), it requires a lot of effort to keep a high accuracy for the senses over multiple domains. PRAC has already shown that it is capable to handle such generic action core, so we are intending to focus the resources during the thesis rather on the missing role inference instead of the engineering process of the action core knowledge base. In this section, we illustrate the approach to design and train the Markov logic networks which are used for the information extraction process.

3.1.1 MODELING THE JOINT PROBABILITY DISTRIBUTIONS

In this section, we introduce the joint probability distributions which are used to perform the action role inference during the information extraction process. These inference results are used to annotate the sentences from a given corpus and store them in a database. These joint probability distributions are represented as fuzzy Markov logic networks. We create for each action core an individual network.

Given a natural-language instruction, PRAC applies the Stanford Parser to determine the grammatical relations of this sentence. After the action core for the given instruction is determined, the corresponding fuzzy MLN is used to infer the action roles. This probabilistic model uses the grammatical relations and the synsets of the words in the instruction as evidence to perform this inference. This same process pipeline is used by our information extraction system. The difference is that the results are stored in a database instead of using them to infer an executable plan. Additional information about this process are given in the further course of this thesis.

For this thesis we designed the MLN templates to handle imperative sentences. However, since we are using grammatical relations as evidence to infer the action roles, we are even able to handle sentences such as "Alice flips the pancake with a spatula". Of course there are some sentences which are not captured by the proposed MLN design, such as passive sentences, but if it is required, it is not an effort to extend these models.

3.1.1.1 Action Core Definition

Before the start of the design process of the Markov logic networks, we first need to define the roles for each action core. Table 3.1 shows the action cores and their roles.

Action Core	**Action Roles**	**Example Task**
Flavoring	**Spice**: Spice used to flavour the goal object. **Goal**: Object to be flavored. **Action Verb**: Verb which triggers action core.	Flavour the chicken with pepper.
Neutralizing	**Acid**: The acid to be neutralized. **Base**: The base to be neutralized. **Action Verb**: See Flavoring.	Neutralize the methacrylic acid with cyanuramide.
Flipping	**Object to be flipped**: Object which should be flipped. **Utensil**: The utensil which should be used to flip the object. **Action Verb**: See Flavoring.	Flip the pancake with a spatula.
Storing	**Object to be stored**: Object which should be stored. **Location**: The location where the object should be stored. **Action Verb**: See Flavoring.	Store the milk in a fridge.

Table 3.1: The action cores and their corresponding action roles

3.1.1.2 Markov Logic Network Design

The MLN in Section 2.5.3 which we picked for in the action role example considered only the given synsets as evidence. However, some action cores need additional Stanford Parser dependencies as evidence. For instance, "flavor the pancakes with honey" and "flavor the honey with sugar". The preposition "with" gives us evidence enough to decide if "honey" is the spice or the object to be flavored. Now consider a sentence like "neutralize the hydrochloric acid with sodium hydroxide". Since a neutralization works in both ways, it is necessary to identify the acid and base in the sentence rather than which objects neutralizes the other object. This task can be solved without considering the syntax.

Hereinafter, we present the models for each action core and a brief description about the designing process. The presented MLNs are templates, meaning they are not trained. Every formula starts with the initial weight of zero. We define every variable by a question mark. The plus operator instantiates these variables to elements of the domain which appeared during training. So at the end of the training process, all variables with a plus operator will be converted to constants.

Flavoring With this action core we want to determine the most suitable spice for a type of food. For instance, for most people it is reasonable to season a steak with pepper instead of e.g. sugar and cinnamon. However, the last two ingredients would be perfect to flavor a pancake. Figure 3.1 depicts a design possibility to infer the correct senses and roles.

```
0 spice(?w2,+?ac) ^ prep_with(?w1,?w2) ^ is_a(?s1,+?c1) ^
  has_sense(?w2,?s1)  ^ ?w1=/=?w2

0 goal(?w2,+?ac) ^ dobj(?w1,?w2) ^ is_a(?s1,+?c1) ^
  has_sense(?w2,?s1)  ^ ?w1=/=?w2

0 action_verb(?w1,+?ac) ^ is_a(?s1,+?c1) ^  has_sense(?w1,?s1)
```

Figure 3.1: Flavoring Markov Logic Network

Neutralizing With this action core we intend to determine the correct neutralizer for a given chemical substance. Since three action cores in this thesis are related to the kitchen domain, we aim to show with this one that our solution can perform on multiple domains.

As mentioned, this action core has been already implemented in PRAC. Nevertheless, for this thesis we altered the model. The previous design contained the roles ***neutralizee*** and ***neutralizer*** instead of ***acid*** and ***base***. The loss of this design was that the roles depended on the syntax. Since the acid can neutralize the base and the other way around, the former model would require that we need two natural language sentences to express this symmetric relation. We encountered this disadvantage by modeling the MLN (see Figure 3.2) to identify the acid and base by using just the evidence from WordNet.

Originally, the 'Neutralizing' action core had some additional roles. ***Unit*** and ***amount*** were defined to represent the required quantity of a chemical substance to neutralize the other one. For simplicity, we ignore these two roles and focus on providing the knowledge to look up the missing chemical substance.

```
0 action_verb(?w1,+?ac) ^  is_a(?s1,+?c1) ^ has_sense(?w1,?s1)

0 base(?w1,+?ac) ^ !acid(?w1,+?ac) ^ is_a(?s1,+?c1) ^
  has_sense(?w1,?s1)

0 acid(?w1,+?ac) ^ !base(?w1,+?ac) ^ is_a(?s1,+?c1) ^
  has_sense(?w1,?s1)
```

Figure 3.2: Neutralizing Markov logic network

Flipping With the 'Flipping' action core, we want to model what utensil has to be used to flip a specific food. Figure 3.3 shows a design to support the 'Flipping' action core.

```
0 utensil(?w2,+?ac) ^ prep_with(?w1,?w2) ^ is_a(?s1,+?c1) ^
  has_sense(?w2,?s1) ^ ?w1=/=?w2

0 obj_to_be_flipped(?w2,+?ac) ^ dobj(?w1,?w2) ^ is_a(?s1,+?c1) ^
  has_sense(?w2,?s1) ^ ?w1=/=?w2

0 action_verb(?w1,+?ac) ^ is_a(?s1,+?c1) ^ has_sense(?w1,?s1)
```

Figure 3.3: Flipping Markov logic network

Storing With 'Storing' we are intending to infer suitable locations to store e.g. milk or a pan. For this MLN (see Figure 3.4) it is important to consider the syntax. Consider the object "crate". A crate can be stored in a pantry but it can be also used as a container to store other objects.

```
0 action_verb(?w1, +?ac) ^ is_a(?s1, +?c1) ^ has_sense(?w1, ?s1)

0 obj_to_be_stored(?w2, +?ac) ^ dobj(?w1, ?w2) ^ is_a(?s2, +?c1) ^
  has_sense(?w2, ?s2) ^ ?w1=/=?w2

0 location(?w2, +?ac) ^ prep_in(?w1, ?w2) ^ is_a(?s1, +?c1) ^
  has_sense(?w2, ?s1) ^ ?w1=/=?w2
```

Figure 3.4: Storing Markov logic network

Arguments for the Design Choice The presented MLN templates differ from the usual MLN design approach which was used for the previous action cores in PRAC (see Section 2.5). The defined formulas consider only one object. There are no formulas which set multiple objects in relation. Compared to the previous design, the defined formulas were considering at least two roles. The reason why we decided against the previous approach is that the new model reduces the training and inference complexity. This property has a significant impact on parsing a large corpus. To show the benefit of the design, consider the following example.

Given two MLN templates which each represent models to infer the senses and roles for the action core 'Storing'. One MLN considers only one role per formula (see

Figure 3.5) and the other one considers two roles per formula (see Figure 3.6). To keep this example short, we do not consider the role ***action verb*** in these MLNs.

```
0 obj_to_be_stored(?w2, +?ac) ^ dobj(?w1, ?w2) ^ is_a(?s2, +?c1) ^
  has_sense(?w2, ?s2) ^ ?w1=/=?w2

0 location(?w2, +?ac) ^ prep_in(?w1, ?w2) ^ is_a(?s1, +?c1) ^
  has_sense(?w2, ?s1) ^ ?w1=/=?w2
```

Figure 3.5: Storing MLN template which considers one action role per formula

```
0 obj_to_be_stored(?w2,+?ac) ^ dobj(?w1,?w2) ^ is_a(?s2,+?c2)
  ^ has_sense(?w2,?s2) ^ ?w1=/=?w2 ^ location(?w3,+?ac)
  ^ prep_in(?w1,?w3) ^ is_a(?s1,+?c1)
  ^  has_sense(?w3,?s1) ^ ?w2=/=?w3 ^ ?w1=/=?w3
```

Figure 3.6: Storing MLN template which considers two action roles per formula

Next, consider that we train these MLNs with one training sentence. Figure 3.7 shows the sentence "store the mushroom in the basket" represented as a database of ground atoms. The trained MLNs are depicted in Figure 3.8 and Figure 3.9. The trained MLN which considers only one entity per formula contains 6 weighted formulas. However, the second MLN contains 9 weighted formulas. The number of weighted formulas depends on the number of unique concepts in the training set. For example, consider a training set with the sentences "store the mushroom in the basket" and "store the milk in the fridge". This training set includes 5 unique concepts - "mushroom", "basket", "milk", "fridge" and since "store" has the same sense in these two sentences, it is only counted as one concept. Let's define n as the number of unique concepts in the database. The MLN template which considers only one role per formula has two formulas which contain the **is_a** predicate. Since this predicate has the argument $+?c_k$, this formula will be expanded to all unique concepts which are represented in the database. This property leads to that the trained MLN with one role contains $2n$ weighted formulas and the second MLN n^2 formulas after the learning process. The number of weighted formulas has an impact on the running time on the training process, since it is necessary to determine a weight for each formula.

```
1.00   action_core(store-1, Storing)
1.00   action_verb(store-1, Storing)
1.00   dobj(store-1, mushroom-3)
1.00   has_sense(basket-6, basket.n.01)
1.00   has_sense(mushroom-3, mushroom.n.05)
1.00   has_sense(store-1, store.v.02)
1.00   is_a(basket.n.01, basket.n.01)
1.00   is_a(mushroom.n.05, mushroom.n.05)
1.00   is_a(store.v.02, store.v.02)
1.00   location(basket-6, Storing)
1.00   obj_to_be_stored(mushroom-3, Storing)
1.00   prep_in(store-1, basket-6)
```

Figure 3.7: 'Store the mushroom in the basket' represented as training set

```
14.895264    obj_to_be_stored(?w2,Storing) ^ dobj(?w1,?w2) ^ is_a(?s2,mushroom.n.05)
             ^ has_sense(?w2,?s2) ^ (?w1=/=?w2)

-2.803733    obj_to_be_stored(?w2,Storing) ^ dobj(?w1,?w2) ^ is_a(?s2,basket.n.01)
             ^ has_sense(?w2,?s2) ^ (?w1=/=?w2)

-2.803733    obj_to_be_stored(?w2,Storing) ^ dobj(?w1,?w2) ^ is_a(?s2,store.v.02)
             ^ has_sense(?w2,?s2) ^ (?w1=/=?w2)

-2.803733    location(?w2,Storing) ^ prep_in(?w1,?w2) ^ is_a(?s1,mushroom.n.05)
             ^ has_sense(?w2,?s1) ^ (?w1=/=?w2)

14.895264    location(?w2,Storing) ^ prep_in(?w1,?w2) ^ is_a(?s1,basket.n.01)
             ^ has_sense(?w2,?s1) ^ (?w1=/=?w2)

-2.803733    location(?w2,Storing) ^ prep_in(?w1,?w2) ^ is_a(?s1,store.v.02)
             ^ has_sense(?w2,?s1) ^ (?w1=/=?w2)
```

Figure 3.8: Trained Storing MLN which considers one action role per formula

```
−1.514542   obj_to_be_stored(?w2,Storing) ^ dobj(?w1,?w2) ^ is_a(?s2,mushroom.n.05)
            ^ has_sense(?w2,?s2) ^ (?w1=/=?w2) ^ location(?w3,Storing)
            ^ prep_in(?w1,?w3) ^ is_a(?s1,mushroom.n.05)
            ^ has_sense(?w3,?s1) ^ (?w2=/=?w3) ^ (?w1=/=?w3)

14.948770   obj_to_be_stored(?w2,Storing) ^ dobj(?w1,?w2) ^ is_a(?s2,mushroom.n.05)
            ^ has_sense(?w2,?s2) ^ (?w1=/=?w2) ^ location(?w3,Storing)
            ^ prep_in(?w1,?w3) ^ is_a(?s1,basket.n.01)
            ^ has_sense(?w3,?s1) ^ (?w2=/=?w3) ^ (?w1=/=?w3)

−1.514542   obj_to_be_stored(?w2,Storing) ^ dobj(?w1,?w2) ^ is_a(?s2,mushroom.n.05)
            ^ has_sense(?w2,?s2) ^ (?w1=/=?w2) ^ location(?w3,Storing)
            ^ prep_in(?w1,?w3) ^ is_a(?s1,store.v.02)
            ^ has_sense(?w3,?s1) ^ (?w2=/=?w3) ^ (?w1=/=?w3)

−1.514542   obj_to_be_stored(?w2,Storing) ^ dobj(?w1,?w2) ^ is_a(?s2,basket.n.01)
            ^ has_sense(?w2,?s2) ^ (?w1=/=?w2) ^ location(?w3,Storing)
            ^ prep_in(?w1,?w3) ^ is_a(?s1,basket.n.01)
            ^ has_sense(?w3,?s1) ^ (?w2=/=?w3) ^ (?w1=/=?w3)

−1.514542   obj_to_be_stored(?w2,Storing) ^ dobj(?w1,?w2) ^ is_a(?s2,store.v.02)
            ^ has_sense(?w2,?s2) ^ (?w1=/=?w2) ^ location(?w3,Storing)
            ^ prep_in(?w1,?w3) ^ is_a(?s1,basket.n.01)
            ^ has_sense(?w3,?s1) ^ (?w2=/=?w3) ^ (?w1=/=?w3)

0.000000    obj_to_be_stored(?w2,Storing) ^ dobj(?w1,?w2) ^ is_a(?s2,basket.n.01)
            ^ has_sense(?w2,?s2) ^ (?w1=/=?w2) ^ location(?w3,Storing)
            ^ prep_in(?w1,?w3) ^ is_a(?s1,mushroom.n.05)
            ^ has_sense(?w3,?s1) ^ (?w2=/=?w3) ^ (?w1=/=?w3)

0.000000    obj_to_be_stored(?w2,Storing) ^ dobj(?w1,?w2) ^ is_a(?s2,basket.n.01)
            ^ has_sense(?w2,?s2) ^ (?w1=/=?w2) ^ location(?w3,Storing)
            ^ prep_in(?w1,?w3) ^ is_a(?s1,store.v.02)
            ^ has_sense(?w3,?s1) ^ (?w2=/=?w3) ^ (?w1=/=?w3)

0.000000    obj_to_be_stored(?w2,Storing) ^ dobj(?w1,?w2) ^ is_a(?s2,store.v.02)
            ^ has_sense(?w2,?s2) ^ (?w1=/=?w2) ^ location(?w3,Storing)
            ^ prep_in(?w1,?w3) ^ is_a(?s1,mushroom.n.05)
            ^ has_sense(?w3,?s1) ^ (?w2=/=?w3) ^ (?w1=/=?w3)

0.000000    obj_to_be_stored(?w2,Storing) ^ dobj(?w1,?w2) ^ is_a(?s2,store.v.02)
            ^ has_sense(?w2,?s2) ^ (?w1=/=?w2) ^ location(?w3,Storing)
            ^ prep_in(?w1,?w3) ^ is_a(?s1,store.v.02)
            ^ has_sense(?w3,?s1) ^ (?w2=/=?w3) ^ (?w1=/=?w3)
```

Figure 3.9: Trained Storing MLN which considers two action roles per formula

The number of weighted formulas has also an impact on the complexity of the inference process. Consider the natural-language instruction which is given to PRAC to infer the senses and roles. The words in this instruction generate m possible synsets. These synsets are given as evidence to infer the action roles. During this inference, the first model creates $2nm$ grounded formulas, where $2n$ are the mentioned number of weighted formulas. However, the second model generates $n^2 * m^2$ grounded formulas. In the first model, there are 2^{2nm} worlds which have to be evaluated to infer the most likely action roles. In the second scenario there are $2^{n^2 * m^2}$ worlds. Theoretically, the inference process has in both models a complexity of $O(2^n)$. However, practically it has a significant impact on the performance. Especially if the action role inference is performed on a corpus with thousands of sentences.

Though the model with one role improves the performance of the inference, it has one disadvantage. For instance, consider the action core ’Adding’. A MLN which contains only formulas with one role is not able to represent a distribution which should handle instructions such as "add some plates to the table" and "add 5 and 4". In the first sentence "add" has the sense of putting an object to a table. In the second sentence it has the sense of summing up two numbers. This kind of inference can be only performed if the sense inference for the action verb considers the given objects. In this thesis the action verbs of the considered action cores have only one sense, so it is not required to consider the relation between objects and verbs.

An additional disadvantage of not representing the relational knowledge is that these MLNs cannot be used to retrieve missing roles since it is not possible to model e.g. that pepper can be used to season a chicken. However, for the information extraction process it is not required to perform this kind of inference. Also, the extracted knowledge stored in the knowledge base is able to represent such relations.

3.1.2 LEARNING THE JOINT PROBABILITY DISTRIBUTIONS

The presented MLN templates are not able to infer action roles without being trained with data. There are many possibilities to acquire training data for the models. One possibility would be to use Amazons Mechanical Turk[1], a platform where machine learning engineers provide unlabeled data and other people can earn money by annotating this data. Since this method requires money and consumes much time due to the preparation and evaluation of the data, this approach would be beyond the scope of this thesis and decided to add the task of developing an annotated data set to our future agenda. The focus of this thesis is to develop and evaluate a solution to overcome the scaling problem of joint probability distribution for missing information inference. From it the conclusion results that we decided to create our own training set for this thesis.

For the evaluation of the two action cores ’Flavoring’ and ’Neutralizing’, we want to use a self-designed corpus to demonstrate the performance of our solution. This corpus contains simple sentences to compensate for the missing coreference resolver. Consider a document which contains the sentences "put the steak in the pan. After 2 minutes, flip it with a fork." Without any coreference resolution, our information extraction system is not able to resolve "it" to "steak". So the system does not capture the information that steaks can be flipped with a fork. The self-designed corpus contains sentences e.g. "flip the steak with a fork" which guarantee correct parsing results and also avoids the problem of a missing coreference resolver. Since we are creating this corpus for the evaluation by ourself, we know the content of this corpus and therefore it is not a challenge to create a training set for these two action cores.

For the evaluation of ’Flipping’ and ’Storing’, we want to use the WikiHow corpus to show that our solution is able to extract information of documents from the internet. The challenge is to create a training set which trains these two MLNs so that they infer the correct action roles. A possible solution is presented in the following section.

[1]https://www.mturk.com/mturk/welcome

3.1.2.1 Creating suitable training sets to extract information from natural-language documents

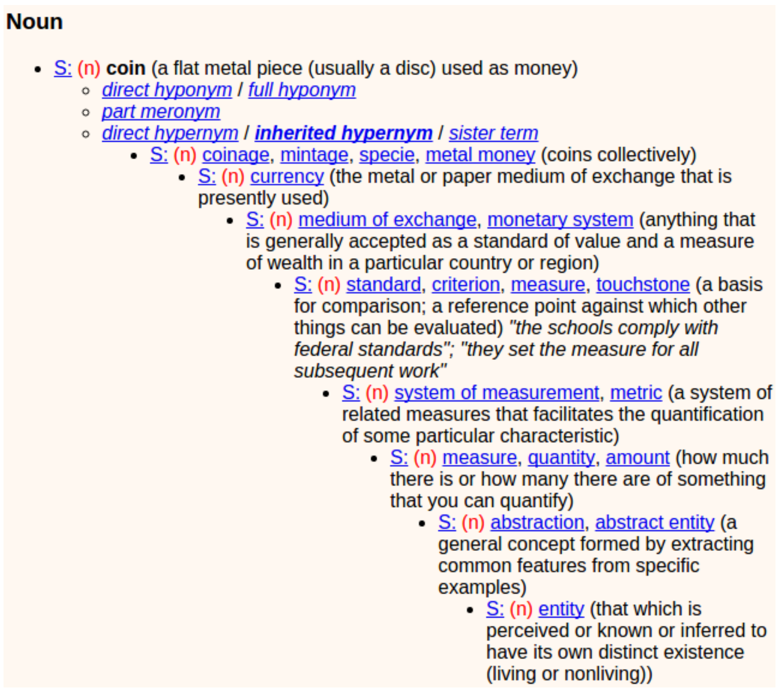
It is a challenge to create a suitable training set for the fuzzy MLNs to infer the correct action roles in natural-language documents. One thing to consider is that the joint probability distributions cannot contain too many random variables because this will slow down the inference process. The other challenge is to create a training set which provides enough information to train the MLNs such that they able to infer the correct senses and role of the sentences contained in the corpus.

Before suggesting any solution to this problem, we first need to define what we consider as a suitable training set. In Section 1.2, we mentioned that the PRAC framework contains a plan library. These plans are designed by people with the intention to perform certain tasks in a domain e.g. performing a neutralization in a chemical domain or using a spice jar to season food. So in conclusion, PRAC can only produce reasonable results for known actions. For instance, consider a robot which performs actions in the kitchen domain. If a robotic agent gets a 'Flipping' instruction, it is expected that it uses a utensil to flip a specific food. Without any additional training, PRAC cannot produce reasonable results for a sentence such as "flip a coin with your fingers". Since we know what kind of plans are supported in PRAC, we know what kind of information we want to extract from the natural-language documents. Considering the 'Flipping' action core, we want to provide information e.g. which utensil can be used to flip a steak or a pancake. To summarize, with the information extraction system we are indenting to extract action core specific knowledge from natural-language document. So we define a training set as suitable if it keeps the number of random variables small and provides enough information so that the probabilistic model is able to infer the correct action roles for the sentences in the corpus which contain action core specific knowledge.

In general, they are two points which have to be considered for extracting action core specific information. First, it has to be evaluated if a given corpus contains the necessary information. For example, if a corpus describes how to flip non-food objects, a robot operating in the kitchen domain cannot utilize this knowledge. The second point is to have joint probability distributions which infer the correct action cores and the corresponding roles. The difficulty is that a corpus is unstructured and to be able to structure the data, it is necessary to have trained joint probability distributions. To get the joint probability distributions, it requires a suitable training set. Since we have no labeled data of the corpus to create a training set, we decided to collect statistics about the corpus and based on the results we want to solve these two tasks. In the following we describe what statistics we are creating and also how we utilize this information to create our training set.

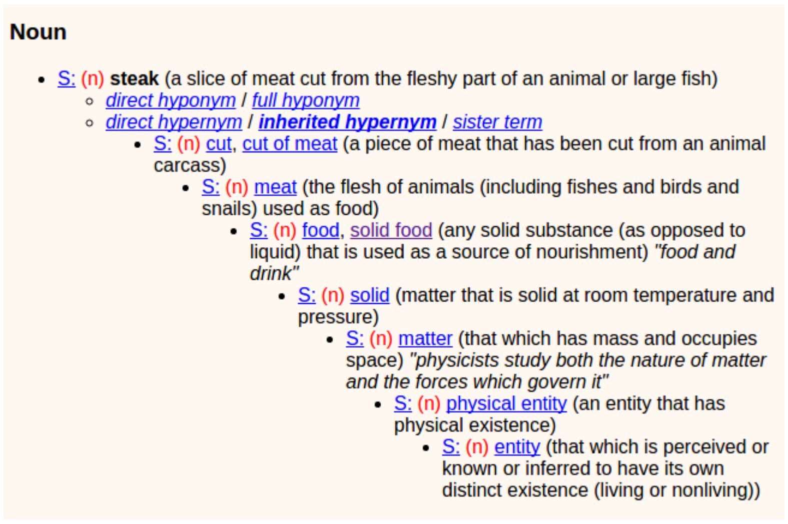
To create these statistics, we started to collect all kind of information which we can get from a natural-language corpus without having these probability distributions. First, we can extract all grammatical relations of each sentence in the corpus with the Stanford Parser. In the following, we define the grammatical relations which represent objects, such as direct objects and prepositional objects, object-types. Also, we make a differentiation between prepositional objects, since prepositions provide semantic evidence too. For example, "flip the pancake with a spatula" and "flip the pancakes after two minutes". The prepositional object of the first sentence is a utensil. However, the prepositional object of the second sentence is a time constraint. In addition to the grammatical relations, the parser provides POS tags of each word in the sentence which can be used to query all possible synsets for the given word. So in conclusion, based on the information given by the Standford Parser, we are able to query for e.g. all synsets of the direct objects of the verb "flip". Since a synset is a concept in a taxonomy, it is possible to get the super-concepts (called hypernyms in WordNet) of these synsets. Given all object-types and hypernyms, we want to aggregate these data to determine the hypernyms which describe best the majority of the synsets in the corpus. For instance, if we are looking for a corpus which contains knowledge about the 'Flipping' action core, an analysis of the corpus should provide that the majority of direct objects of the verb 'flip' activate food related synsets and the prepositional objects with the preposition "with" activate synsets of the utensil domain.

To illustrate the intention of this approach and how it is possible to generate such aggregate data, consider a corpus with the sentences "flip a coin with your fingers", "flip a steak with a fork", "flip a pancake with a spatula" and "flip a pancake with a turner". Parsing these sentences results that "coin", "steak" and "pancake" are the direct objects and "fingers","fork" and "spatula" are the prepositional objects of the verb "flip". To keep this example short, we consider only an analysis for the direct objects. The synsets and the corresponding hypernyms for the direct objects are represented in Figures 3.10, 3.11 and 3.12. In our example, these direct objects have only one synset in WordNet.


Noun

- S: (n) **coin** (a flat metal piece (usually a disc) used as money)
  - *direct hyponym* / *full hyponym*
  - *part meronym*
  - *direct hypernym* / ***inherited hypernym*** / *sister term*
    - S: (n) coinage, mintage, specie, metal money (coins collectively)
      - S: (n) currency (the metal or paper medium of exchange that is presently used)
        - S: (n) medium of exchange, monetary system (anything that is generally accepted as a standard of value and a measure of wealth in a particular country or region)
          - S: (n) standard, criterion, measure, touchstone (a basis for comparison; a reference point against which other things can be evaluated) *"the schools comply with federal standards"; "they set the measure for all subsequent work"*
            - S: (n) system of measurement, metric (a system of related measures that facilitates the quantification of some particular characteristic)
              - S: (n) measure, quantity, amount (how much there is or how many there are of something that you can quantify)
                - S: (n) abstraction, abstract entity (a general concept formed by extracting common features from specific examples)
                  - S: (n) entity (that which is perceived or known or inferred to have its own distinct existence (living or nonliving))

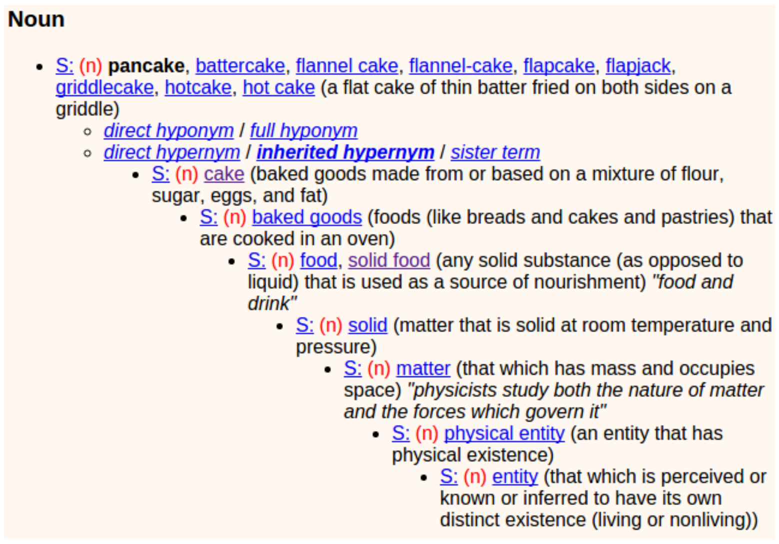
Figure 3.10: Hypernym path of the synset coin



Noun

- S: (n) **steak** (a slice of meat cut from the fleshy part of an animal or large fish)
  - *direct hyponym* / *full hyponym*
  - *direct hypernym* / ***inherited hypernym*** / *sister term*
    - S: (n) cut, cut of meat (a piece of meat that has been cut from an animal carcass)
      - S: (n) meat (the flesh of animals (including fishes and birds and snails) used as food)
        - S: (n) food, solid food (any solid substance (as opposed to liquid) that is used as a source of nourishment) *"food and drink"*
          - S: (n) solid (matter that is solid at room temperature and pressure)
            - S: (n) matter (that which has mass and occupies space) *"physicists study both the nature of matter and the forces which govern it"*
              - S: (n) physical entity (an entity that has physical existence)
                - S: (n) entity (that which is perceived or known or inferred to have its own distinct existence (living or nonliving))

Figure 3.11: Hypernym path of the synset steak



Noun

- S: (n) **pancake**, battercake, flannel cake, flannel-cake, flapcake, flapjack, griddlecake, hotcake, hot cake (a flat cake of thin batter fried on both sides on a griddle)
  - *direct hyponym* / *full hyponym*
  - *direct hypernym* / ***inherited hypernym*** / *sister term*
    - S: (n) cake (baked goods made from or based on a mixture of flour, sugar, eggs, and fat)
      - S: (n) baked goods (foods (like breads and cakes and pastries) that are cooked in an oven)
        - S: (n) food, solid food (any solid substance (as opposed to liquid) that is used as a source of nourishment) *"food and drink"*
          - S: (n) solid (matter that is solid at room temperature and pressure)
            - S: (n) matter (that which has mass and occupies space) *"physicists study both the nature of matter and the forces which govern it"*
              - S: (n) physical entity (an entity that has physical existence)
                - S: (n) entity (that which is perceived or known or inferred to have its own distinct existence (living or nonliving))

Figure 3.12: Hypernym path of the synset pancake

Having this information, the next step is to determine the hypernyms which describe best the majority of synsets of the direct objects. Before explaining how we define the best hypernym, we want to motivate why we want to consider the number of occurrence of a hypernym. First, we want to mention that we only consider the amount of unique words in a corpus for the statistics. By unique words we mean that in our example, "pancake" as a direct object is considered only once. This should make sure that the statistics provide a general overview about the domains of the object-types and are not affected by repetitive sentences. The motivation for taking the numbers of hypernym occurrence into account can be illustrated by considering the example corpus. "Steak" and "pancake" have some identical hypernyms. However, "coin" is not related to "steak" and "pancake". By ignoring these facts, an analysis would present that the direct objects of the verb "flip" are of the domain "food" and "coinage" but we would not have any information which domain is represented stronger in the corpus. In general, a corpus contains a large amount of sentences and we are not able to analyze each sentence manually. So it could be that the corpus contains a majority of the sentences which do not represent action core specific knowledge. Since an information extraction process requires time, it would be better to look for a corpus which contains a greater number of relevant sentences.

To determine the best hypernym, we want to consider its number of appearance in the corpus. What we define as the best hypernym is related to the task of creating a suitable training set for the MLNs which perform the action role inference. We are indenting to use a score which defines the best hypernym. Based on this score these hypernyms can be ranked. An additional example should illustrate what we define as the best hypernym. For instance, consider if we would calculate the score by the frequency of appearance of a hypernym. If we would apply this approach on the example corpus, the highest scored hypernym would be "entity". In WordNet, every concept is a sub-concept of the hypernym "entity". So this approach would rank "entity" as the best hypernym for each corpus. In general, the more abstract concepts would get a higher score than more specific hypernyms. Rating abstract hypernyms high does not provide any benefit since the results cannot give any detailed information about domains of the object-types.

Another approach could be to count the appearance of the most specific hypernyms. In our example, the most specific hypernyms are "cake ", "cut of meat" and "coinage". Such a result would provide detailed information about the domains of the direct objects. However, this information is not suitable to create a training set with the condition to have low number of random variables. Based on this information, it looks like we have to add entities of the "cake ", "cut of meat" and "coinage" domain to the training set to infer the correct action roles for the direct objects in the example corpus. By analyzing the hypernyms, it would be more reasonable to represent "cake " and "cut of meat" with the "food" hypernym. With this new result, we would be able to create a smaller training set, which contains one entity of the domain "food" and one of the "coinage", without decreasing the accuracy of the action role inference. In conclusion, the best hypernym represents the domain of an object-entity, which describes the majority of synsets appearing in a corpus. In addition, it is specific enough to provide an understanding of the represented domain and it is abstract enough to be able to define a compact training set to extract this represented information.

These arguments show that to determine the score for these hypernyms it is necessary to consider their number of occurrences and also their average abstraction level. For the abstraction level we want to consider the path length of the synset to its hypernym. However, in this path we do not want to consider the synset itself. The argument for that is that we want to create an analysis for the hypernyms. In our example it is better to represent that "cake" has an abstraction level of 1 for "pancake" instead of 2. For instance, the abstraction level of "entity" is 7 for "pancake". It is also 7 for "steak" and for "coin" it is 8. The average abstraction level for the hypernym "entity" in this corpus is 7.33. In general, we define the average abstraction level as follows:

$$\overline{abstraction\_levels(x)} = \frac{\sum_{s \in synsets\_activated(x)}(path\_length(s,x) - 1)}{|synsets\_activated(x)|}$$

where $synsets\_activated(x)$ denotes the set containing all synsets in the corpus which activated the hypernym x.

So the abstraction level mean of "entity" in our example can be represented as:

$$\overline{abstraction\_levels(entity.n.01)} = \frac{\sum_{s \in \{pancake.n.01, steak.n.01, coin.n.01\}}(path\_length(s, entity.n.01) - 1)}{|\{pancake.n.01, steak.n.01, coin.n.01\}|} = 7.33$$

The abstraction level mean of "food" in our example is:

$$\overline{abstraction\_levels(food.n.01)} = \frac{\sum_{s \in \{pancake.n.01, steak.n.01\}}(path\_length(s, food.n.01) - 1)}{|\{pancake.n.01, steak.n.01\}|} = 3$$

Since hypernyms are also synsets, we represent them in the following in the NLTK notation. To calculate a score for each hypernym which considers the abstraction level and the number of occurrences, we develop the following formula:

$$score(x) = \left(1 - \frac{\overline{abstraction\_levels(x)}}{\max_{y \in Hypernym} \overline{abstraction\_levels(y)}}\right) * \frac{|words\_activated(x)|}{total\_num\_of\_words}$$

We indent to score more specific hypernyms higher and also consider the amount of words which activated the hypernyms. One additional constraint is to give the hypernym "entity.n.01" the score of 0 since all synsets are part of this synset. This is represented with the term:

$$\max_{y \in Hypernym} \overline{abstraction\_levels(y)}$$

Table 3.2 represents the analysis result for the direct object for the verb 'flip' in the example corpus. To determine if this corpus provides action core specific knowledge, we have to perform the same analysis for the prepositional object "with". The results also show that they have to be evaluated by people and cannot be automatically used by an application to generate a suitable training set. The presented equation is not able to penalize all abstract hypernyms (see "solid.n.01") but is able to present an overview about the represented domains. It shows that "food.n.01" is the strongest represented hypernym in the corpus and "coinage.n.01" also got a high score. Since fuzzy MLNs are able to utilize taxonomic knowledge, we can create a training set which contains entities of the "food" and "coinage" domain to infer the correct senses of the objects in the corpus.

During this thesis we developed a tool, called Corpus Analyzer, which performs this analysis for a given corpus. In the following section, we present the results of the Corpus Analyzer for the WikiHow corpus.

Hypernym	Mean abstraction level	Num of actiavted words	Score
food.n.01	3	2	0.394
solid.n.01	4	2	0.303
coinage.n.01	1	1	0.288
cake.n.03	1	1	0.288
cut.n.06	1	1	0.288
currency.n.01	2	1	0.242
baked_goods.n.01	2	1	0.242
meat.n.01	2	1	0.242
matter.n.03	5	2	0.212
medium_of_exchange.n.01	3	1	0.197
standard.n.01	4	1	0.151
physical_entity.n.03	6	2	0.121
system_of_measurement.n.01	5	1	0.106
measure.n.02	6	1	0.06
abstract.n.06	7	1	0.015
entity.n.01	7.33	3	0

Table 3.2: Corpus Analyzer results for the direct objects of the verb ’flip’ in the example corpus

3.1.2.2 Creating the Training Sets for the Joint Probability Distributions

In this section we present the training sets which are used to train the fuzzy MLNs to perform the action role inference during the knowledge extraction. As mentioned, for the evaluation of the action cores ’Flavoring’ and ’Neutralizing’ we use a self-designed corpus. This corpus contains sentences describing how to season different kinds of food such as meat and baked goods. In addition, there are sentences which describe which acid has to be used to neutralize a specific base. Since we know what kind of knowledge is represented in the self-designed corpus, we do not have to perform a corpus analysis to create a suitable training set. However, since fuzzy MLNs are able to utilize taxonomic knowledge, we add entities to the training set which are not contained in the corpus to avoid a direct look up of the action roles.

For the evaluation of the 'Flipping' and 'Storing' action core we want to use the WikiHow corpus. To analyze if the corpus contains action core specific knowledge for these two action cores, we create an analysis with the Corpus Analyzer. While presenting the results of the analysis, we show only the 10 highest scored hypernyms. One reason is to provide a compact overview about the results. Additionally, we are interested to see if the corpus contains a high number of action-core-specific sentences. If the 10 best scored hypernyms are not representing the relevant domains, then this is an indicator that the corpus does not contain many action core specific sentences. Based on this analysis, we create suitable training sets for the fuzzy MLNs to extract the knowledge in the corpus.

One advantage of the application of fuzzy MLNs is that we do not need to label the sentences in the corpus to train these models. Due to the property that fuzzy MLNs utilize taxonomic knowledge, we are able to add similar sentences to the training set and the models are still able to infer the correct senses and roles. Since we are not intending to let the sentences in the corpus be labeled by people due to time and financial constraints, we are creating the training sets based on similar objects which appear in the corpus. This approach can be compared with NELL (see Section 1.3.3). This system gets also initial knowledge to extract information of natural-language documents.

Flavoring To train the MLN for 'Flavoring', we used the sentences in Figure 3.13. Our intention is to train the MLN to cover a various range of foods and spices.

```
Flavor the soup with salt.
Flavor the steak with pepper.
Flavor the chicken with sassafras.
Flavor the sauce with coriander.
Flavor the meat with fenugreek.
Flavor the omelette with salt.
Flavor the rib with pepper.
Flavor the egg with salt.
Flavor the jambalaya with chili powder.
Flavor the cookie with sugar.
```

Figure 3.13: Flavoring training set

Neutralizing For the 'Neutralize' action core we also created 10 sentences (see Figure 3.14). These sentences are created by using the sister terms of "hydrochloric acid", "melamine" and "sodium hydroxide". This training set contains 10 acids and 10 bases, there the bases are divided into 5 child synsets of ***base.n.08*** and 5 of ***hydroxide.n.01***. The combination of the acids and bases in these sentences might not be correct regarding the chemical domain. These sentences are created by random. The most important aspect of this training set is to have the information which words in these sentences represent the acid or the base.

```
Neutralize the pectic acid with the pyridine.
Neutralize the permanganic acid with the purine.
Neutralize the phthalic acid with the imidazole.
Neutralize the picric acid with the the melamine.
Neutralize the pyruvic acid with the pyrimidine.
Neutralize the maleic acid with the aluminum hydroxide.
Neutralize the oxalacetic acid with the calcium hydroxide.
Neutralize the oxalic acid with the magnesium hydroxide.
Neutralize the oxyacid with the potassium hydroxide,
Neutralize the pantothenic acid with the sodium hydroxide.
```

Figure 3.14: Neutralizing training set

Flipping To demonstrate how the system performs on a real corpus, we want to use the WikiHow corpus to infer the missing roles for the 'Flipping' action core. We applied the Corpus Analyzer to determine the domains for the direct object and prepositional objects with the preposition "with". Figure 3.15 and Table 3.3 show the result for the direct object and Figure 3.16 and Table 3.4 for the prepositional object. The shown taxonomies highlight the scoring. A higher concentration of green represents a higher score.

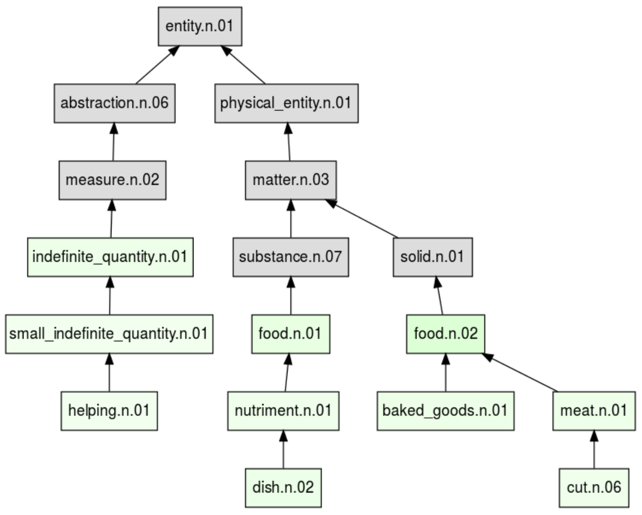


Figure 3.15: The 10 highest scored hypernyms for the direct object of the verb 'flip'

Hypernym	**Score**	**Num of Words**	**Example Words in the corpus**
food.n.02	0.216	29	bacon,vegetables,fish
food.n.01	0.147	23	tofu,toast,burger
dish.n.02	0.119	12	omelette,patties,sandwich
indefinite_quantity.n.01	0.115	15	toast,slice,deal
baked_goods.n.01	0.110	12	pancake,tortilla
meat.n.01	0.105	12	cut,ham,bacon
nutriment.n.01	0.1	13	omelette,cakes,sandwich
cut.n.06	0.094	9	fillets,steaks,sides
helping.n.01	0.083	8	wing,fillets,pieces
small_indefinite_quantity.n.01	0.079	9	fillets,pieces,hair

Table 3.3: The 10 highest scored hypernyms for the 79 unique direct objects of the verb ’flip’

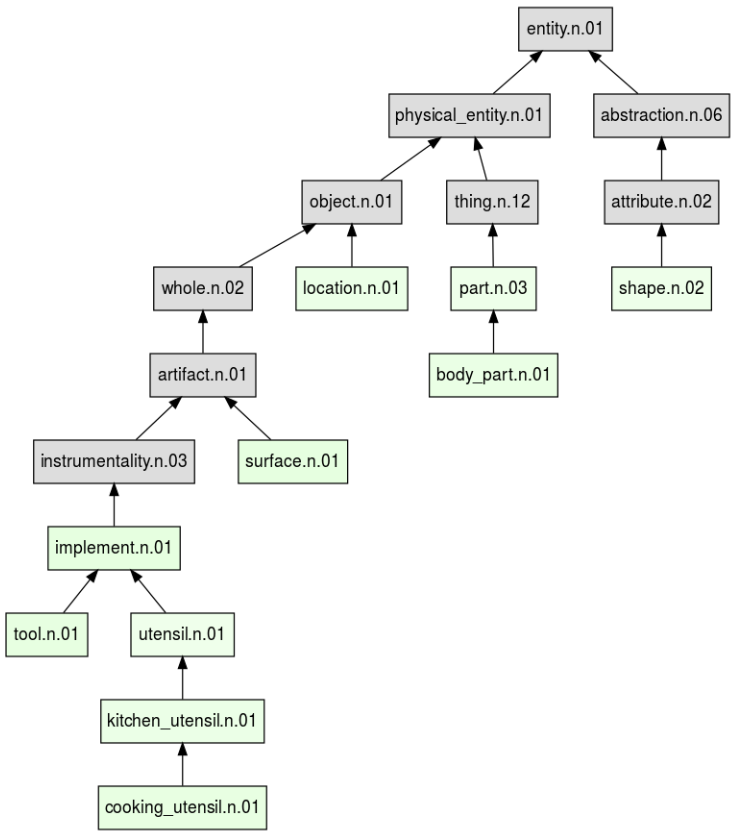


Figure 3.16: The 10 highest scored hypernyms for the preposition ’with’ of the verb ’flip’

Hypernym	**Score**	**Num of Words**	**Example Words in the corpus**
surface.n.01	0.158	2	rim,side
implement.n.01	0.158	3	fork,spatula,skillet
tool.n.01	0.156	2	fork,spatula
cooking_utensil.n.01	0.15	2	spatula,skillet
body_part.n.01	0.133	2	wrist,side
location.n.01	0.125	2	fork,side
kitchen_utensil.n.01	0.125	2	spatula,skillet
shape.n.02	0.109	2	fork,rim
part.n.03	0.108	2	wrist,side
utensil.n.01	0.1	2	spatula,skillet

Table 3.4: The 10 highest scored hypernyms for the 10 unique preposition 'with' objects of the verb 'flip'

Since we are focusing on inferring a utensil to flip a given food, the analysis shows that the selected corpus contains action core specific knowledge. The analysis of the direct object shows a huge concentration of the food domain. Compared to the prepositional analysis there are only 10 unique objects which are captured by the Stanford Parser. However, it is possible to see that there are utensils represented in the corpus. In conclusion, the WikiHow corpus is suitable to provide action core specific knowledge for the 'Flipping' action core.

We want to train our MLN that the direct object are of the food domain and the prepositional object is a tool which can be used by a robot. For the training set we consider for the direct object the words **omelette, patties, sandwich** (dish.n.01), **tofu, toast** (food.n.01), **pancake, tortilla** (baked_goods.n.01) and **bacon, vegetables, fish** (food.n.02). For the prepositional object we considered the words **fork, spatula** and **skillet**. To avoid a direct look up of the senses during the corpus processing and to demonstrate how well fuzzy MLNs perform on unseen objects, we create the training set based on the sister terms of the considered words. For instance, we added "meatballs" instead of "omelette" to the training set. The created training set is shown in Figure 3.17.

```
Flip the meatballs with the omelet pan.
Flip the rissole with the spork.
Flip potato skin.
Flip the tofu with the table knife .
Flip the loaf.
Flip the waffles with the pancake turner.
Flip the latke.
Flip the spareribs with the saucepan.
Flip the vegetables.
Flip the seafood with the fish slice.
```

Figure 3.17: Flipping training set

Storing Figure 3.18 and Table 3.5 show the Corpus Analyzer results for the direct object and Figure 3.19 and Table 3.6 the prepositional object of the verb "store".

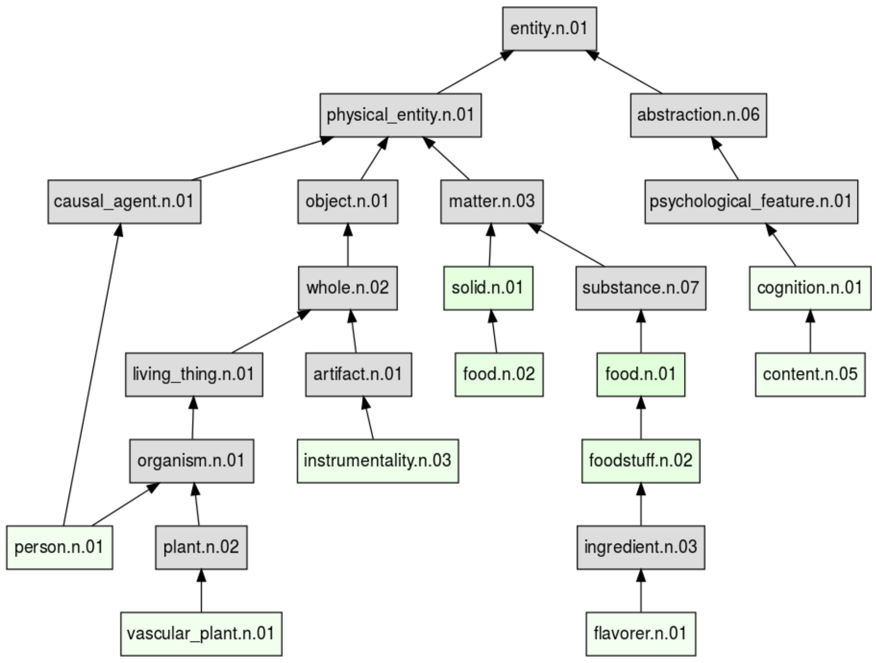


Figure 3.18: The 10 highest scored hypernyms for the direct object of the verb 'store'

Hypernym	Score	Num of Words	Example Words in the corpus
food.n.01	0.185	33	cordial,stuffing,tea
solid.n.01	0.166	26	raw meat,coconut,fish
foodstuff.n.02	0.145	22	honey,milk,cream,ingredients
food.n.02	0.138	21	rambutans,coconut,meat
instrumentality.n.03	0.105	19	circles,cereal boxes,bottles
content.n.05	0.085	12	product,food,stuff
cognition.n.01	0.082	15	unit,way,items
vascular_plant.n.01	0.08	14	onions,wheat,mustard
person.n.01	0.079	12	worms,Fahrenheit,dish
flavorer.n.01	0.075	10	vinegar,paste,mayo

Table 3.5: The 10 highest scored hypernyms for the 94 unique direct object of the verb ’store’

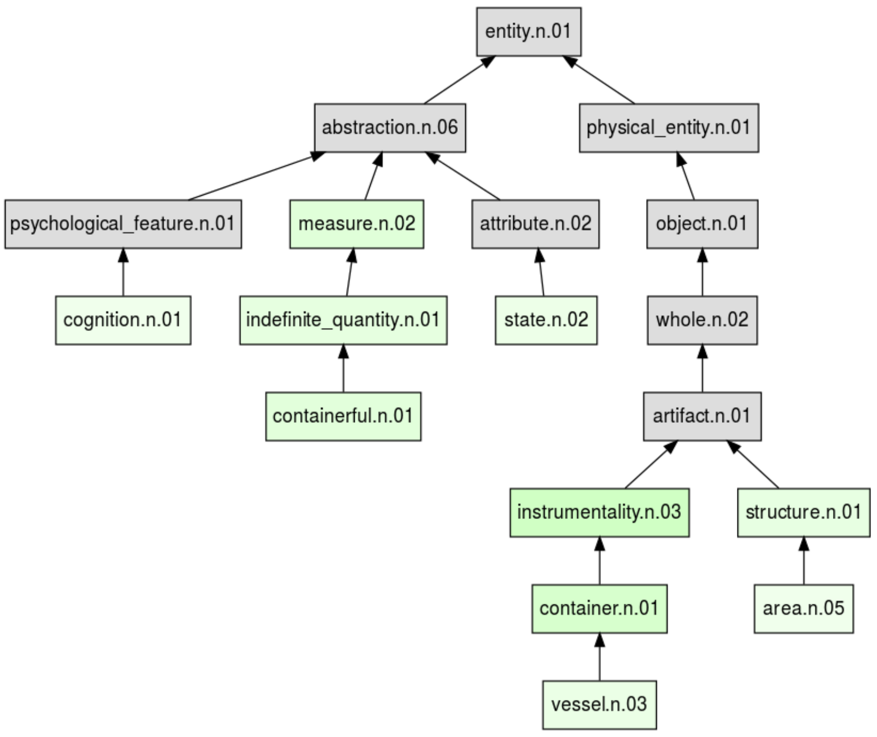


Figure 3.19: The 10 highest scored hypernyms for the preposition ’in’ of the verb ’store’

Hypernym	**Score**	**Num of Words**	**Example Words in the corpus**
instrumentality.n.03	0.306	29	plastic bag,packet,container
container.n.01	0.260	21	tin,canister,paper bag
measure.n.02	0.196	20	space,even,days
containerful.n.01	0.183	13	box,glass,pot
indefinite_quantity.n.01	0.162	13	bag,carton,bins
structure.n.01	0.151	15	house,root cellar,basement
vessel.n.03	0.126	9	bottles,tanks,jar
state.n.02	0.115	11	top,days,environment
area.n.05	0.097	9	cupboard,closet,pantry
cognition.n.01	0.096	9	darkness,area,top

Table 3.6: The 10 highest scored hypernyms for the 61 unique preposition 'in' objects of the verb 'store'

The Corpus Analyzer provides the information that the 'Storing' action core can be more complex. The direct object analysis shows that the words represent the food domain but also different types of objects such as a cereal boxes and bottles. Also, there are multiple domains regarding the prepositional object. The robotic agent can use a container such as a box and a bag as storing locations, but also areas such as a pantry. For the training set we considered for the direct object the words **cereal boxes, bottles, pot, jar, pan** (instrumentality.n.03), **tea, beer, milk, sauce, batter** (food.n.01) and **meat, onions, pie, chicken, rambutans** (food.n.02). For the prepositional object we consider the words **cupboard, pantry, cellar** (area.n.05), **bottles, jar, pot** (vessel.n.05) and **bag, carton, glass** (container.n.01). Again, to avoid a direct look up of the senses we created the training set using the sister terms. The created training set is shown in Figure 3.20.

```
Store the chest.
Store the crate.
Store the pitcher.
Store the bucket in an attic.
Store the tin.
Store the barrel in a lumber room.
Store the bowl in a drawer.
Store the drum.
Store the dish.
Store the cup.
Store the seafood in a crate.
Store the mushroom in a basket.
Store the strudel.
Store the duck in a bucket.
Store the berry in a barrel.
Store the coffee.
Store the mead.
Store the yogurt in a cup.
Store the dip.
Store the dough in a drum.
```

Figure 3.20: Storing training set

3.2 Knowledge Representation for Storing Extracted Information of Natural-language Documents

In this section we describe the design process of the knowledge representation to store the extracted information from natural-language documents. First, we describe the requirements which have to be met by our knowledge representation. After that description, we present the design of the knowledge representation and we outline how this design satisfies these requirements.

3.2.1 The Requirements for the Knowledge Representation

In the following, we present the requirements which have to be met by the knowledge representation:

Representing required information Our information extraction system has to represent the knowledge in a way that PRAC can utilize it to infer missing action roles. As mentioned in Section 3.1, we use fuzzy MLNs to extract the action core, roles and senses of the sentences contained in a given corpus. The results of this inference process should be stored in the knowledge base. If PRAC gets an incomplete instruction, it should be able to retrieve the sentences of the knowledge base which have the same action core like the instruction. Once the sentences are retrieved, PRAC should be able to determine the semantically most similar sentence by comparing the action roles between the instruction and the sentences. In conclusion, we define knowledge representation as suitable if it enables us to implement the mentioned procedures to complete instructions.

Apply simple queries to access knowledge Despite the first requirement, we attempt to develop a knowledge base where simple queries are enough to get the required knowledge. Meaning, the user does not need to use a complex query language or write nested join queries to retrieve the action-core-specific sentences and their action roles. The benefits of avoiding join queries are that the user does not have to invest time to study the tables and designing the query. Additionally, some join queries types are not tractable for large data sets. For instance, a cross join between the tables *A* and *B* returns the Cartesian product between the rows of the given tables. This operation cannot be performed an a database which contains tables with thousands of rows.

Knowledge base can be edited or extended without greater effort There are use cases where it is necessary to modify the knowledge base. For instance, change the name of action cores and action roles. The modification of the extracted sentences has to be done fast and without affecting the other results. That includes avoiding changes of database schema if an action core will be expanded with new roles, for instance.

PRAC can use this knowledge base Since the main goal of this thesis is to infer missing action roles, PRAC needs to access this knowledge base. One requirement is that PRAC can access this knowledge base with a simple designed and maintainable interface.

Easy debugging of extracted information An additional important aspect is to understand how the information extraction system creates its results. One approach could be to use log files and try to understand how the system behaved on a given sentence. However, for this application it is not suitable to create a log file. First, most log files work with timestamps. So if we want to understand how the inference results were created, we need the time in which the inference process was performed. In addition, we have to make sure that the knowledge base and log files are kept together. To avoid the mentioned drawbacks, it would be feasible to model the knowledge representation such that the actions of the information extraction process can be reproduced.

3.2.2 THE KNOWLEDGE REPRESENTATION DESIGN

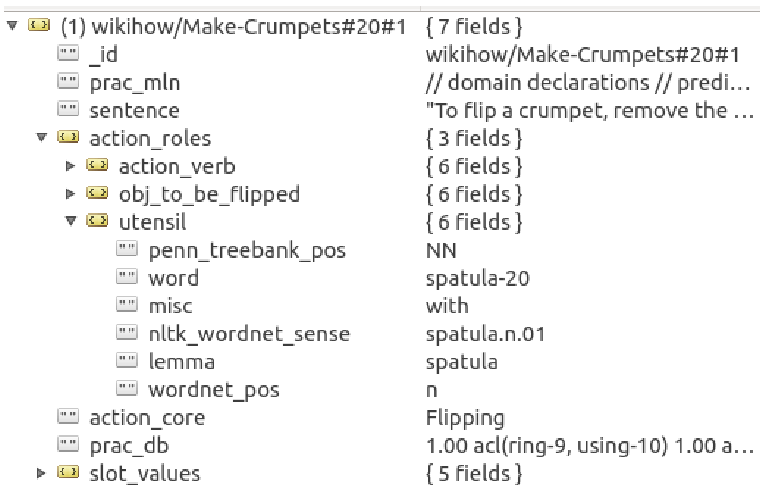
Our knowledge base design is inspired by IBM's frame concept used in PRISMATIC (see Section 1.3.2). We are intending to perform the action role inference on a single predicate and its corresponding entities such as subject, direct object and prepositional object. We decided to represent the extracted information as an attribute-value pair. This allows us to represent the data as objects and we are not required to use relational databases. In our case it is more reasonable to avoid a relational knowledge representation. The arguments for that are stated in the next section. We decided to name our attribute-value pair representation "frame", since our knowledge representation is similar to IBM's design in PRISMATIC (see Section 1.3.2). To summarize, a frame represents the action core and its corresponding roles which were inferred from a sentence contained in a natural-language document. Table 3.7 shows and describes our frame definition. Figure 3.21 depicts an example for an extracted frame of the WikiHow corpus.

Attribute	**Description**
ID	A unique ID for the extracted frame. The ID is a combination of the text source name, the number of the sentence and the number of the recognized predicate in this sentence.
Sentence	The original sentence from which this frame is extracted.
PRAC MLN	A copy of the required MLN to read the PRAC database. The predicate definitions are stored in a MLN. Without these definitions PRAC cannot process the database stored in the PRAC DB attribute.
PRAC DB	The dependencies of the Stanford Parser are represented as ground atoms. So the parsing results are stored in the frame. This allows us to understand which evidences were used to perform the action role inference. We do not store the is_a atoms to save storage space.
Slot Values	Is a dictionary where the keys are the slot names and the values are Sense objects (see Table 3.8). We decided to call this attribute "slot values", since our frame definition is inspired by IBM (see Section 1.3.2). Currently, there are four slot names supported - *Subj*, *dobj*, *prepobj* and *predicate*. This stored information allows us to perform the corpus analysis by the Corpus Analyzer, for instance. Additionally, this interface allows performing syntactic information retrieval.
Action Core	Represents the action core which is activated by this frame. Entry contains *UNKNOWN* if no action core could be inferred.
Action Roles	Is a dictionary where the keys are the roles corresponding to the action core and the values are Sense objects. One alternative possibility to represent this attribute could be use the WordNet senses as values instead of the Sense object. However, we decided to provide additional information for use cases which might be relevant in the future.

Table 3.7: Frame representation

Attribute	**Description**
Word	Origin word contained in the processed sentence.
Lemma	Word stem of the represented word.
Penn Treebank Pos	Penn Treebank part-of-speech tags which are determined by the Stanford Parser.
WordNet Pos	In contrast to the Penn Treebank POS, WordNet uses only three kinds of part-of-speech tags (noun, verb and adjective).
NLTK WordNet Sense	The WordNet sense is represented in NLTK notation.
Misc	Currently this field is used to store the prepositions of the prepositional objects.

Table 3.8: Sense object representation



```
▼ (1) wikihow/Make-Crumpets#20#1   { 7 fields }
    _id                            wikihow/Make-Crumpets#20#1
    prac_mln                       // domain declarations // predi...
    sentence                       "To flip a crumpet, remove the ...
  ▼ action_roles                   { 3 fields }
    ▶ action_verb                  { 6 fields }
    ▶ obj_to_be_flipped            { 6 fields }
    ▼ utensil                      { 6 fields }
        penn_treebank_pos          NN
        word                       spatula-20
        misc                       with
        nltk_wordnet_sense         spatula.n.01
        lemma                      spatula
        wordnet_pos                n
    action_core                    Flipping
    prac_db                        1.00 acl(ring-9, using-10) 1.00 a...
  ▶ slot_values                    { 5 fields }
```

Figure 3.21: Example frame extracted from the WikiHow corpus

The frame in Figure 3.21 represents one action core which was inferred from the sentence "to flip a crumpet, remove the ring using tongs, and then flip the crumpet with a spatula" which is contained in the WikiHow corpus. Based on this one sentence, it is possible to infer that a spatula can be used to flip a crumpet. Since the word "crumpet" is annotated with a concept of WordNet, it is possible to infer that similar objects such as pancakes can be flipped with a spatula.

3.2.3 ARGUMENTATION FOR THE KNOWLEDGE REPRESENTATION

In this section we argument how the defined knowledge representation satisfies all requirements.

Representing required information Each frame can be annotated with the activated action core and its corresponding action roles. So it is possible to store the action role inference results of the fuzzy MLNs which are used for the information extraction. These results can be then accessed by PRAC to infer missing action roles.

Apply simple queries to access knowledge The standard approach to represent data is the use of relational databases. We decided against it. The first reason is that querying for the information would require that the user had to write complex join queries. Another reason is the difficulty to model the action roles representation. Every action core has different action roles. So for each action core we would have to define a separate table. Every update of an action core definition would request that the database schema has to be altered. We decided to represent the frames as JSON objects rather than a collection of tables. The benefit is that we need to define this data structure once and do not need to update it when new action cores are added to PRAC.

We decided to store the JSON objects in a MongoDB[2]. The main reason why we use MongoDB instead of object databases or other NoSQL databases is that MongoDB is used in other projects such as Open Ease[3] where PRAC is a part of.

In combination with the JSON representation and MongoDB, it is possible to send simple query to retrieve the information. For instance, we can ask for all 'Flipping' frames just by querying all frames whose ***Action Core*** attribute is equal to ***Flipping***. To get the senses of the action role ***utensil*** we can send a query like `frame.action_roles.utensil.nltk_wordnet_sense` to the database.

Knowledge base can be edited or extended without greater effort Every frame exists independent of each other. So it is possible to delete or modify a single frame without affecting the remaining ones. Since every frame is a JSON object, single attribute modifications can be done without greater effort.

PRAC can use this knowledge base A MongoDB library exists for Python[4]. Since PRAC is written in Python, we use this library to implement the support that PRAC can access the knowledge base.

Easy debugging of extracted information Every result of the Standford Parser is stored in the frame. So it is easy to retrace how a frame was derived by the system.

3.3 BUILDING UP THE KNOWLEDGE BASE

In this section we describe how we apply fuzzy MLNs to extract the action-core-specific information of natural-language documents and store it in the proposed

[2]https://www.mongodb.org/
[3]http://www.open-ease.org/
[4]https://api.mongodb.org/python/current/

knowledge representation (Section 3.2). The information extraction process is divided in five steps. Figure 3.22 shows the process pipeline of our system.

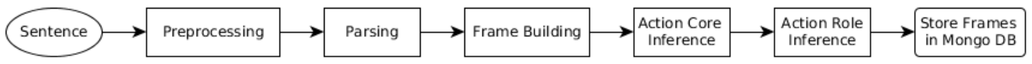


Figure 3.22: Information extraction pipeline

We are not going in detail about the "Parsing", "Action Core Inference" and "Action Role Inference" processes since we provided a detail example in Section 2.5. For this thesis we trained the action core MLN to support the four action cores ’Flavoring’, ’Neutralizing’, ’Flipping’ and ’Storing’. During the action role inference, the information extraction systems use the MLNs which are mentioned in Section 3.1.1.2.

At the end of this section we also point out some drawbacks of using MAP queries to perform information extraction and describe how we are intending to cope with them.

3.3.1 PREPROCESSING

Before the general extraction process starts, every sentence is going through a preprocess to minimize the error rate during parsing. Currently, we implemented a simple compound noun handling and some methods to handle imperative sentences due to common parsing errors like described in Section 2.4.1.

3.3.1.1 Imperative Sentence Handling

Imperative sentences such as "season the steak" or "store the bowl" can be challenging for modern parsers since "season" and "store" can be tagged as nouns. The Stanford Parser identifies these two verbs as nouns if no preprocessing is performed. We discovered that setting these sentences in quotation marks before parsing, minimizes the error rate in the Stanford Parser. We apply this technique on every sentence which will be processed. Currently, we do not have discovered any other errors which occur by applying this technique. Even comparing sentences like "bob down the hill with a sledge" and "Bob went to Anna" resulted that the word "bob" was tagged correctly.

We are aware that this approach is not the best solution but developing the perfect parser for natural-language is still a current research topic. Since this solution improves the information extraction process for the action cores which we are using for the evaluation, we decided to develop a more sophisticated solution in later future.

3.3.1.2 Compound Noun Handling

Consider the sentence "neutralize hydrochloric acid with sodium hydroxide.". Without compound noun handling the Stanford Parser represents "acid" as direct object and "hydroxide" as a propositional object. With this result we are only able to infer that acid can be neutralized with hydroxide and we cannot capture the more specific knowledge which is actually represented in this sentence. So to improve the knowledge coverage, we implemented a basic compound noun handling.

Our system uses the Stanford Parser and WordNet to determine compound nouns. At first, we analyze the extracted dependencies given by the Standford Parser to identify compound nouns. Some compound nouns are recognized by the parser e.g. "swimming pool" or sequences of proper nouns.

Unfortunately, the parser does not recognize all compound nouns. For example, "washing machine" and "baking sheet" will not be identified. The parser tags "washing" and "baking" as adjectives. Also, not every recognized compound noun is a correct compound noun e.g. "sugar bowl" versus "metal bowl". "Sugar bowl" is actually a name for a bowl which stores sugar. However, a "metal bowl" is a bowl made of metal.

Nevertheless, we can use these behaviors for finding compound nouns. We composite every suggested compound noun or every noun and its corresponding adjective to one word. For instance, "hydrochloric acid" is composited to "hydrochloric_acid". Using WordNet, we can check if a synset exists for this composition. If a synset exists for the compound noun, the system replaces the words in the original sentence with the composition. Regarding our example, the compound noun procedure would return "neutralize hydrochloric_acid with sodium_hydroxide". Sequences of proper nouns are marked as compound nouns without using the WordNet sanity test.

3.3.2 FRAME BUILDING

After the Stanford Parser generates the dependency tree, our system creates a list of all recognized verbs. Auxiliary and modal verbs are not contained in this list. The parser is able to identify auxiliary and modal verbs. For each verb, every subject, direct, indirect and propositional object is extracted. Since our system does not support coreference resolution, each object which is a pronoun is revoked. If no valid object of the corresponding verb can be acquired, then no frame will be built.

For example, consider the sentence "flip the golden brown pancake with a spatula and season the pancake with sugar". Figure 3.23 shows part of the parsing results for the given sentence. Based on the information that "flip" and "flavor" are tagged as verbs, the information system assumes that this sentence contains two frames. Since at least one object exists for every verb in this example, the system extracts two frames. One frame represents "flip the pancake with a spatula" and the second frame represents "flavor the pancake with sugar". The following action core and role inference is performed separately for each frame. After the inference process, these two frames are stored in the MongoDB.

```
1.00   dobj(flavor-11, pancake-13)
1.00   dobj(flip-2, pancake-6)
1.00   has_pos(flavor-11, VB)
1.00   has_pos(flip-2, VB)
1.00   has_pos(pancake-13, NN)
1.00   has_pos(pancake-6, NN)
1.00   has_pos(spatula-9, NN)
1.00   has_pos(sugar-15, NN)
1.00   prep_with(flavor-11, sugar-15)
1.00   prep_with(flip-2, spatula-9)
```

Figure 3.23: Parsing results for the frame building example sentence

To represent sentences like "season the chicken and the bacon with pepper and salt", we create multiple frames. In this example our system would build four frames. One frame would represent "seaon the chicken with pepper". The other three would represent "season the chicken with salt", "season the bacon with pepper" and "season the bacon with salt".

The first evaluations showed that the frame building process needed some improvement. The sentence "flip over each piece of chicken with a fork" resulted in a frame where "piece" was marked as a direct object of "flip". A part of the parsing results of the sentence is depicted in Figure 3.24. A better result would be to resolve "piece of chicken" to "chicken" and then tag it as a direct object. In this thesis we handle sentences like this by checking if one of the recognized objects is linked through a "prep_of" dependency to another object. If this is the case, we extract the second object as the corresponding object of the verb. In our example the direct object "piece" has an "prep_of" dependency to the word "chicken". So instead of tagging "piece" as direct object in the frame, the system tags "chicken". In the future, we are intending to develop a more sufficient solution to handle this kind of inference.

```
1.00   dobj(flip-1, piece-4)
1.00   has_pos(chicken-6, NN)
1.00   has_pos(flip-1, VB)
1.00   has_pos(fork-9, NN)
1.00   has_pos(piece-4, NN)
1.00   prep_of(piece-4, chicken-6)
1.00   prep_with(flip-1, fork-9)
```

Figure 3.24: Parsing results for 'flip over each piece of chicken with a fork'

3.3.3 DRAWBACKS OF USING MAP QUERIES FOR INFORMATION EXTRACTION

As mentioned in Section 2.3.2.1, PRAC performs MAP queries to infer the most probable action core and the corresponding roles. Probability queries cannot be performed because they require a high computational effort and therefore a practical solution to extract knowledge from natural-language documents could not be implemented. However, performing information extraction from an open-domain corpus requires a probability distribution over the possible solutions for the query random variables. The following examples should demonstrate why it would be more suitable to create a probability distribution rather than determine the most probable assignment for the query random variables.

Consider the sentences "make some pancakes" and "flip a coin with your fingers". If we perform an action core inference on the sentence "make some pancakes", the WCSP solver determines that the action core 'Neutralizing' is the most probable solution. Since the WCSP solver returns only the costs for the best solution, we are not able to evaluate how reasonable the inferred solution is. If we would have a probability over the best solution, we could define a threshold and revoke solutions whose probability is below the defined threshold. Since we are limited to use MAP queries, we have no other choice than accepting this result and perform the action role inference with the 'Neutralizing' MLN in this example.

A similar scenario happens if our information extraction system gets the sentence "flip a coin with your fingers". In the first inference step the system infers the correct 'Flipping' action core. However, the 'Flipping' MLN infers the wrong sense for the word "fingers" due to the fact that this MLN is trained with objects of the kitchen domain. Without a probability attached to this solution, we are not able to evaluate the solution and therefore we have to store the frame which contains wrong senses. Since we are forced to use the WCSP inference due to its better performance, we added a sanity check to the information retrieval process. Our information retrieval algorithm considers semantic similarity to retrieve the missing information. So the idea is to utilize this semantic similarity to determine e.g. that "make some pancakes" does not describe how to neutralize chemical substances and that "flip a coin" does not describe how to flip a pancake.

3.4 RETRIEVING THE SEMANTICALLY MOST SIMILAR SENTENCES TO COMPLETE ROBOT INSTRUCTIONS

In Section 1.1 and 1.3 we mention the drawbacks of a syntactic and text-based information retrieval approach. One flaw of this approach is that we can only complete a robot instruction if this instruction was mentioned in the given corpus. For instance, to complete the instruction "season a steak" it is required that a sentence like "you can season a steak with pepper" appeared in the processed corpus. Imagine a scenario where you have a cookbook available which describes how to season different kinds of meat. Now consider the task to season a rib. In this scenario this cookbook does not mention explicitly how to season a rib. However, since this cookbook describes how to season other kinds of meat, it is possible to determine which spices might be suitable to season a rib. So we are able to infer reasonable information based on semantic similarity. A text-based approach would not be able to solve the presented task. So to overcome this problem, we aim with this thesis to model this kind of behavior with our information retrieval algorithm. In conclusion, the idea of our retrieval algorithm is to look for the semantically most similar frame which completes the given instruction.

3.4.1 SEMANTIC SIMILARITY BETWEEN FRAMES

Before we introduce our retrieval algorithm, we want to present our similarity measure which represents the semantic similarity between the extracted sentences of a corpus and a given instruction. To be more specific, this similarity measure determines the semantic similarity between frames. So to be able to retrieve the frames to complete a given instruction, PRAC transforms the instruction into a frame. This transformation is performed after PRAC has inferred the action roles of the instruction.

For our frame similarity measure we decided to use a combination of harmonic mean and the WUP measure. Equation 3.1 shows how to determine the harmonic mean for a list of n positive real numbers [2].

$$harmonic\_mean(x_1, x_2, \ldots, x_n) = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}}, \tag{3.1}$$

For our defined measure we decided against the path similarity and choose the WUP measure instead. One argument for that is that the WUP calculation rates the similarity between more specific synsets higher than general ones [32]. For instance, the WUP similarity between the sister terms "milk" and "coffee" is higher than the similarity between the sister terms of their super-concept. An additional reason for the WUP similarity is that it represents the similarity values between sister terms much higher compared to the path similarity.

The reasons why we want to consider these two properties are that we want to use the similarity value also as a confidence score. The idea of a confidence score is to represent how suitable the missing roles in the extracted frame are to perform the given instruction. We define a missing role as suitable if it can be directly asserted as a plan parameter and the plan can be executed successfully by the robot. The idea for the confidence score arose from letting a robotic agent operate in the chemical domain. For instance, if the robot performs chemical experiments and during these experiments the agent would infer e.g. a wrong chemical substance, then this could trigger serious consequences. The main argument for choosing the WUP measure for the frame similarity value, and therefore for the modeling the confidence score, is that it rates the similarity between specific synsets higher than more abstract synsets. Consider the following two scenarios: In the first scenario we want to determine how suitable the sentence "flip the meat with a spatula" describes to perform the instruction "flip the fish". In the second scenario we want to determine how suitable the sentence "flip the bacon with a fork" describes how to perform the instruction "flip the spareribs". In WordNet "meat" and "fish" are sister terms. However, "bacon" and "spareribs" are sister terms too but they are more specific. If we would determine the same confidence score for the first scenario as for the second, then we would not be able to differentiate that the first scenario is more abstract compared to the second scenario. We want to score abstract scenarios less than specific ones since it might be possible that the inferred missing roles are not suitable to perform the given instruction.

An additional reason for choosing the WUP similarity instead of the path similarity is that the WUP measure rates sister terms higher. For instance, the WUP similarity between "bacon" and "spareribs" is 0.89. However, their path similarity is 0.33. Since we are intending to model a confidence score, a solution with a score of 89% promises a higher success rate to complete the instruction rather than a score of 33%.

Since we stated all our intenions which we want to represent with this similarity value, we present in the following our frame similarity measure. The numbers which are considered for the frame similarity calculation are the WUP similarities between the roles in frame fr_1 and frame fr_2. The Equation 3.2 shows the calculation. This equation should only be applied for frames with the same action core.

$$frame\_sim(fr_1, fr_2) = \frac{|R|}{\sum_{i=0}^{|R|} \frac{1}{wup(fr_1.action\_role.I(i,R).sense, fr_2.action\_role.I(i,R).sense)}}, \tag{3.2}$$

where $R = fr_1.action\_roles \cap fr_2.action\_roles$ and $I(n, R)$ returns the nth element of the set R.

The reason why we use the harmonic mean is that the resulting value tends to the value of the smallest number in the given set [1]. Since we are using this semantic similarity as confidence score, this property provides that the action role with the lowest semantic similarity has a strong impact on the scoring. So a high value guarantees that all compared action roles have a high semantic similarity.

The semantic retrieval algorithm does not work if an action role similarity is 0. In our current implementation this scenario cannot happen. Every WUP similarity comparison between nouns will return a value greater than 0. The comparison between verbs will also never be 0 due to the constraints we present in Section 3.4.2.1.

3.4.2 EXTENDING PRAC WITH THE MISSING ROLE INFERENCE FUNCTIONALITY

Figure 3.25 shows where PRAC performs the missing role inference. After the action role inference, PRAC examines if the given task misses some required roles. If this is the case, it starts the information retrieval process which is described in the following subsections.

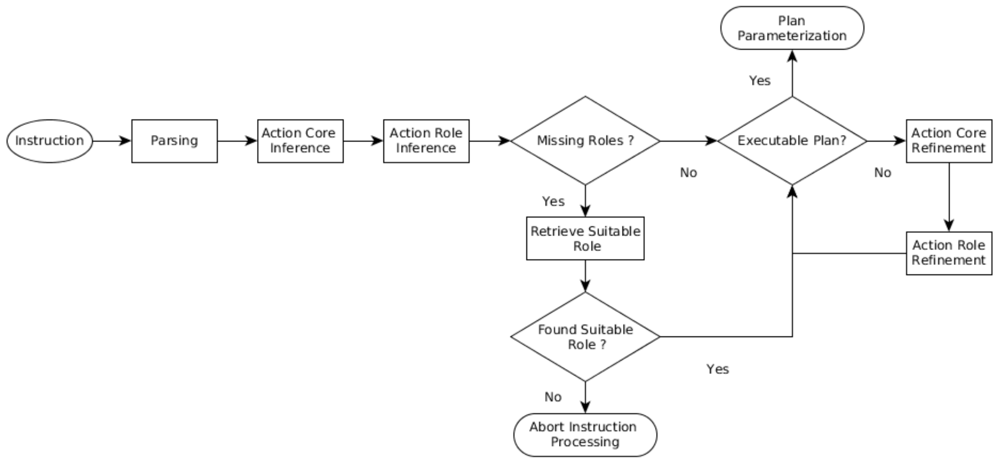


Figure 3.25: PRAC process pipeline with role inference support

3.4.2.1 Query for Semantic Similar Frames

Before the missing role retrieval can be triggered, PRAC needs to perform the action role inference on the given instruction. Based on these results, the system is able to determine the missing roles which are required to perform the instruction. Before the retrieval process starts, PRAC transforms the instruction into a frame. So it is able to compare the instruction to the frames which are stored in the database.

The information retrieval process is divided in three steps. First, PRAC queries all frames in the knowledge base which contain the same action core and action roles as the given instruction. In addition, these frames have to contain all missing roles. To motivate the last constraint consider the following situation: Let's define an action core with 5 action roles. It might be possible that an incomplete instruction which activates this action core misses 3 roles. Consider that during the missing role retrieval there are 4 frames which are potential candidates to provide the missing roles. One frame contains a possible solution for missing role number 1. The second frame contains a solution for the second missing role and the third frame contains a solution for missing role number 3. Each of these frames have the same semantic similarity to the given instruction. The fourth retrieved frame contains all three missing roles but has a slightly lower similarity to the instruction as the other three frames. The difficulty in this scenario is to determine which frames should be used to complete the given instruction. One possibility is to merge the first 3 frames. The other possibility is to infer the missing roles which are contained in the fourth frame. We decided to consider only frames which provide all missing information since it provides more semantic information than frames which provide only a subset of the missing roles. Our motivation of the retrieval algorithm is to use as much semantic information as possible since the inferred solutions have an impact on the real world in which the robotic agent is operating. So it is a better trade of to revoke some solutions which might be correct from merging multiple frames to one frame than causing accidents in the real word.

Due to the WCSP inference, the knowledge base contains frames with false action cores (see Section 3.3.3). To filter these frames, the system considers the WUP similarity between the action verb of the given task and the extracted frames. If the value is less than 85%, then PRAC revokes these frames. We determined this value by evaluating the extracted frames from WikiHow. We retrieved all frames which activated the 'Storing' and 'Flipping' action core. From these frames we extracted the synsets of the inferred action verbs. Then, we determined the WUP similarity between the extracted synsets and the synsets we used to train the action verbs of the 'Flipping' and 'Storing' action core. The synsets with the similarity 0.85 appeared to be similar to the training set sentences. These frames represented sentences like "you can keep the sauce in the fridge" and "turn the pancake with the help of turner".

The last step of the role retrieval process is to determine the frame similarity between the remaining frames and the instruction. These frames are attached with their similarity value and then stored in a list. Afterwards, this list is sorted in descending order based on their similarity. The resulted list can be used to complete the given instruction. A more detailed description about the completion process is given in the next section. Algorithm 1 shows the described procedure to retrieve semantic similar frames.

Algorithm 1 Retrieve semantic similar frames

```
 1: function RETRIEVE_SEMANTIC_SIMILAR_FRAMES(instruction_frame)
    input: The given instruction represented as a frame
    output: A list of frames with attached similarities
            which are sorted in descending order
            based on the semantic similarity to instruction_frame
 2:     missing_roles ← DETERMINE all missing roles in instruction_frame
 3:     frame_list ← QUERY all frames which have the same action core,
                     contain all missing roles and
                     the same inferred action roles as instruction frame
 4:     filtered_frame_list ← REMOVE all frames whose WUP similarity
                              between the action verb of instruction_frame
                              is below 85%
 5:     query_results ← INIT empty list
 6:     for all frame ∈ filtered_frame_list do
 7:         sim ← CALL frame_sim(frame,instruction_frame)
 8:         tuple ← CREATE (sim,frame)
 9:         query_result ← APPEND tuple
10:     query_result ← SORT tuples in query_result based on sim in descending order
11:     return query_result
```

3.4.2.2 Completion of Robot Instructions

Since PRAC is able to retrieve the semantically most similar frames to an incomplete instruction, it can use these frames to infer the missing roles. The simplest approach to complete the instruction is to use the missing roles which are contained in the frame with the highest similarity. With this approach we are able to infer missing roles in $O(n)$ where n is the number of stored frames in the database.

This simple and fast approach has some disadvantages. It is possible that the proposed missing roles do not exist in the environment in which the robotic agent is operating. A possible solution would be that the robotic agent checks if the missing roles in the second highest scored frame exist in the environment. This procedure can be repeated until the robot finds objects which exist in its environment, the list of frames gets empty or the list does only contain frames which have a confidence score below a defined limit. We call this limit confidence limit. For instance, if the robotic agent is operating in a chemical domain then it would be reasonable to set the confidence limit to 1. However, in the kitchen domain it is not necessary to consider only information which are 100% semantically similar to the given instruction since a sentence like "flip the waffles with a spatula" is suitable to complete the instruction "flip a pancake". At the moment this confidence limit can be only set manually. We mention in Section 4.3 why it is difficult to determine such a limit.

Since the term of a confidence limit is defined, let us refer back to the mentioned missing role approach and its disadvantage. An additional disadvantage of this approach is that some missing roles cannot be utilized by a robot. Our evaluation in Section 4.3 shows that after parsing the WikiHow corpus and querying for information where to store a steak the sentence "store basil in the fall" was returned as the highest rated frames. Obviously, "fall" cannot be interpreted as a real storing location.

The last disadvantage which we discovered is that the robot cannot utilize objects in its environment which are similar to the inferred missing roles. For instance, the robotic agent knows that in its environment a "turner" is available. If the agent gets the task to flip a pancake and the retrieval algorithm returns a spatula as solution, then the robot is not able to infer that a turner is also suitable to perform the task. In the following we present a solution which overcomes the mentioned disadvantage.

The idea of the solution is to develop an algorithm which determines all possible completions based on the given instruction and a list which contains all objects available in the environment of the robotic agent. These objects in the list are represented as WordNet synsets. As an example, consider the task "flip a pancake" and the given list contains objects such as tongs, spatula and bowl. To determine which object is the most suitable, PRAC transforms the given instruction into a frame and creates a list containing all missing roles. Given that object list and missing roles list, PRAC asserts every object with every missing role. This process creates $|\text{object list}|^{|\text{missing roles}|}$ combinations since it is possible that one object can be asserted with multiple roles. Based on these combinations, PRAC creates all possible completions for the given instruction and represent them as frames. In our example PRAC would create 3 frames where one frame represents "flip a pancake with tongs". The second frame represents "flip a pancake with a spatula" and the third frame represents "flip a pancake with a bowl". Given the 3 frames, PRAC determines based on the frames stored in the database which of the three frame describes the best completion. To perform this kind of inference we can represent this problem with the following equation:

$$\underset{f_i \in CF, f_j \in QF}{\operatorname{argmax}} \; frame\_sim(f_i, f_j)$$

where CF is the set containing the created completion frames based on the given object list and QF contains the queried frames. Algorithm 2 shows the described procedure.

This proposed solution should not replace the completion through retrieving semantic similar frames since it requires a high computational effort to create all possible completion which have to be evaluated. However, if the number of missing roles and objects in the environment is small then it can be considered as an alternative to the first solution.

Algorithm 2 Infer Missing Roles of an Object List

```
 1: function INFER_MISSING_ROLES_OF_A_OBJ_LIST(instruction_frame, obj_list)
    input: The given instruction represented as a frame and
           list of objects which are represented as synsets
    output: A list of frames with attached similarities
            which are sorted in descending order
            based on the semantic similarity to instruction_frame
 2:     missing_roles ← DETERMINE all missing roles in instruction_frame
 3:     query_results ← INIT empty list
 4:     if missing_roles $\neq \emptyset$ then
 5:         frame_list ← CREATE all $|obj\_list|^{|missing\_roles|}$ possible frames
 6:         for all frame ∈ frame_list do
 7:             sub_query_result ← CALL Retrieve_Semantic_Similar_Frames(frame)
 8:             sim ← RETRIEVE sim of sub_query_result[0]
 9:             tuple ← CREATE (sim,frame)
10:             query_result ← APPEND tuple
11:     query_result ← SORT tuples in query_result based on sim in descending order
12:     return query_result
```

Chapter four

Evaluation

In this chapter we evaluate our information extraction system which extracts action-core-specific knowledge from natural-language documents and represents this knowledge in a semantic knowledge representation. In addition, we evaluate our semantic information retrieval algorithm to complete robot instructions. The evaluation for 'Flavoring' and 'Neutralizing' is performed on a self-created corpus. The evaluation for 'Flipping' and 'Storing' is performed on the WikiHow corpus. How we perform these evaluations are described in their respective sections.

Before we introduce the results for the information extraction and retrieval process, we evaluate the fuzzy MLNs which we designed in Section 3.1. We have to evaluate if the MLNs are able to infer the correct roles and senses for unseen objects. The inference of correct action roles is essential since our retrieval algorithm uses semantic similarity to complete the given instructions. For the evaluation of the performance of these models we perform a 10-fold cross-validation. As accuracy measure for the evaluation we use the F_1 score [18]. If the procedure of a cross-validation is not familiar, we refer to [21] for additional information.

4.1 Evaluation of the Markov Logic Network Models

In this section we present the results of the 10-fold cross-validations for each action core. We perform this evaluation on the sets which are intended to be used to train the MLNs. These sets are mentioned in Section 3.1.2.2. For every action core we performed two cross-validations. One validation was performed to determine the F_1 for the sense inference and the other validation should determine the F_1 for the role inference.

Flavoring The evaluation for the 'Flavoring' action core resulted in a 100% F_1 score for the senses and roles inference. Table 4.1 and 4.2 show the respective confusion matrices.

This result shows that the MLN generalizes well and can infer the correct senses and roles on unseen objects. However, this behavior can be only expected if the given sentences contain entities of the food domain. In addition, the provided sentences require a similar syntactical structure to the imperative sentences which were included in the training set. So for instance, passive sentence cannot be handled by this MLN. Since these arguments also hold true for the remaining MLNs in this section, we do not mention them in the following paragraphs.

Prediction/Ground Truth	chicken.n.01	chili_powder.n.01	cookie.n.01	coriander.n.02	egg.n.02	fenugreek.n.02	jambalaya.n.01	meat.n.01	omelet.n.01	pepper.n.03	rib.n.03	salt.n.02	sassafras.n.02	sauce.n.01	season.v.01	soup.n.01	steak.n.01	sugar.n.01
chicken.n.01	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
chili_powder.n.01	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cookie.n.01	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
coriander.n.02	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
egg.n.02	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
fenugreek.n.02	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0
jambalaya.n.01	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0
meat.n.01	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
omelet.n.01	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
pepper.n.03	0	0	0	0	0	0	0	0	0	**2**	0	0	0	0	0	0	0	0
rib.n.03	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
salt.n.02	0	0	0	0	0	0	0	0	0	0	0	**3**	0	0	0	0	0	0
sassafras.n.02	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0
sauce.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0
season.v.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**10**	0	0	0
soup.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0
steak.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0
sugar.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**

Table 4.1: Flavoring Senses Result

Prediction/Ground Truth	action_verb	goal	spice
action_verb	**10**	0	0
goal	0	**10**	0
spice	0	0	**10**

Table 4.2: Flavoring Roles Result

Neutralizing The 10-fold cross-validations for this action core resulted in a F_1 score of 91% for the senses and 100% for the action roles (see Table 4.3 and 4.4). The 91% can be explained because purine has two synsets which are representing a base in WordNet. Since there is no sense for purine defined in the training set, these two synsets get the same similarities between the concepts in the trained MLN. This results that the WSCP solver determines two possible solutions and therefore it has to pick a solution randomly. In this evaluation, it picked the solution which was not in the test set.

The most interesting part of the evaluation is the results of the action roles inference. The results show that the MLN is able to identify the base or acid just by applying the knowledge of the taxonomy. It does not require any syntactic information to infer the correct roles.

Prediction/Ground Truth	aluminum_hydroxide.n.01	calcium_hydroxide.n.01	imidazole.n.01	magnesium_hydroxide.n.01	maleic_acid.n.01	melamine.n.01	neutralize.v.06	oxalacetic_acid.n.01	oxalic_acid.n.01	oxyacid.n.01	pantothenic_acid.n.01	pectic_acid.n.01	permanganic_acid.n.01	phthalic_acid.n.01	picric_acid.n.01	potash.n.01	purine.n.01	purine.n.02	pyridine.n.01	pyrimidine.n.01	pyruvic_acid.n.01	sodium_hydroxide.n.01
aluminum_hydroxide.n.01	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
calcium_hydroxide.n.01	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
imidazole.n.01	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
magnesium_hydroxide.n.01	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
maleic_acid.n.01	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
melamine.n.01	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
neutralize.v.06	0	0	0	0	0	0	**10**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
oxalacetic_acid.n.01	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
oxalic_acid.n.01	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
oxyacid.n.01	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0
pantothenic_acid.n.01	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0
pectic_acid.n.01	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
permanganic_acid.n.01	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
phthalic_acid.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0
picric_acid.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
potash.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0
purine.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0
purine.n.02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	**0**	0	0	0	0
pyridine.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0
pyrimidine.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0
pyruvic_acid.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0
sodium_hydroxide.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**

Table 4.3: Neutralizing Senses Result

Prediction/Ground Truth	acid	action_verb	base
acid	**10**	0	0
action_verb	0	**10**	0
base	0	0	**10**

Table 4.4: Neutralizing Roles Result

Flipping The cross-validation achieved a F_1 of 89% for the senses (see Table 4.5) and 100% for the action roles (see Table 4.6). The object "spareribs" has two synsets which are of the food domain. So the same argument applies which hold true for the 'Neutralizing' action core and therefore the WCSP solver guessed again the synset which was not defined in the test set.

Prediction/Ground Truth	barbecued_spareribs.n.01	bean_curd.n.01	fish_slice.n.01	flip.v.08	meatball.n.01	nan.n.04	omelet_pan.n.01	pancake_turner.n.01	potato_pancake.n.01	potato_skin.n.01	rissole.n.01	saucepan.n.01	seafood.n.01	sparerib.n.01	spork.n.01	table_knife.n.01	vegetable.n.01	waffle.n.01
barbecued_spareribs.n.01	**0**	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0
bean_curd.n.01	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
fish_slice.n.01	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
flip.v.08	0	0	0	**10**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
meatball.n.01	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
nan.n.04	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0
omelet_pan.n.01	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0
pancake_turner.n.01	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
potato_pancake.n.01	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
potato_skin.n.01	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0
rissole.n.01	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
saucepan.n.01	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0
seafood.n.01	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0
sparerib.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0
spork.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0
table_knife.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0
vegetable.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0
waffle.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**

Table 4.5: Flipping Senses Result

Prediction/Ground Truth	action_verb	obj_to_be_flipped	utensil
action_verb	**10**	0	0
obj_to_be_flipped	0	**10**	0
utensil	0	0	**6**

Table 4.6: Flipping Roles Result

Storing During the 'Storing' MLN evaluation we achieved a F_1 of 85% for the senses and 100% for the action roles. The confusion matrices are depicted in Table 4.7 and 4.7. During the senses evaluation the MLN inferred the wrong senses for "duck" and "dip". It inferred the senses duck.n.04[1] and dip.n.02[2]. This time the explanation is imbalanced data (see Section 2.5.4). For this action core, the MLN learned multiple domains for the ***obj_to_be_stored*** role. It distinguishes between foods, vessels and containers. During the cross-validation it occurs an unbalancing between these domains if for example "duck" is removed from the training set. Since there is no sister terms or other kinds of meat in the training set, this slight unbalancing is enough to provoke a wrong inference for this word. We verified this behavior by training the MLN with the complete training set and tried some additional sister terms of the previously wrong inferred words. The evaluation showed that a balanced trained MLN is able to infer the correct senses.

[1]Synset description for duck.n.04 in WordNet is "a heavy cotton fabric of plain weave; used for clothing and tents"

[2]Synset description for dip.n.02 in WordNet is "(physics) the angle that a magnetic needle makes with the plane of the horizon"

Prediction/Ground Truth	barrel.n.02	basket.n.01	berry.n.01	bowl.n.01	bucket.n.01	chest.n.02	coffee.n.01	crate.n.01	cup.n.01	dip.n.02	dip.n.04	dish.n.01	dough.n.01	drawer.n.01	drum.n.04	duck.n.03	duck.n.04	loft.n.02	lumber_room.n.01	mead.n.03	mushroom.n.05	pitcher.n.02	seafood.n.01	store.v.02	strudel.n.01	tin.n.02	yogurt.n.01
barrel.n.02	**2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
basket.n.01	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
berry.n.01	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bowl.n.01	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bucket.n.01	0	0	0	0	**2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
chest.n.02	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
coffee.n.01	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
crate.n.01	0	0	0	0	0	0	0	**2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cup.n.01	0	0	0	0	0	0	0	0	**2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
dip.n.02	0	0	0	0	0	0	0	0	0	**0**	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
dip.n.04	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
dish.n.01	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
dough.n.01	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
drawer.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
drum.n.04	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**2**	0	0	0	0	0	0	0	0	0	0	0	0
duck.n.03	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0	0	0	0	0	0	0
duck.n.04	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	**0**	0	0	0	0	0	0	0	0	0	0
loft.n.02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
lumber_room.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0
mead.n.03	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
mushroom.n.05	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0
pitcher.n.02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0
seafood.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0
store.v.02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**20**	0	0	0
strudel.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0
tin.n.02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0
yogurt.n.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**

Table 4.7: Storing Senses Result

Prediction/Ground Truth	action_verb	location	obj_to_be_stored
action_verb	20	0	0
location	0	9	0
obj_to_be_stored	0	0	20

Table 4.8: Storing Roles Result

Conclusion These cross-validations show that the proposed MLNs are suitable to infer the correct senses and roles for sentences which have a similar syntactic structure and describe the same domains as the training sentences. So our conclusion is that these MLNs can be trained with the provided training set and then they can be applied to infer semantic information of sentences which are stored in natural-language documents.

4.2 Evaluation with a Self-created Corpus

In this section we present the evaluation for the action cores 'Flavoring' and 'Neutralizing' by using a self-created corpus. As mentioned in Section 3.1.2, the intention of this corpus is to compensate the lack of the coreference resolver. In addition, we can design this corpus so it contains only sentences which contain action-core-specific knowledge.

The focus of this section is to evaluate if the main features of our solution are working before we apply it on the WikiHow corpus. We want to evaluate if compound nouns will be detected correctly. In addition, we want to verify if the extraction of multiple frames of one sentence can be performed. The last feature which we are intending to evaluate is the information retrieval process.

This section is divided in two subsections, each describes the evaluation of one action core. However, the general evaluation process remains the same for these action cores. First, we present the sentences which we added to our corpus and then we explain what features we want to evaluate with the sentences. Afterwards, we evaluate the extracted information which was inferred by the corresponding MLNs. For the information retrieval evaluation we decided to use the retrieval algorithm which gets an object list and determines the most suitable object of this list to complete the given instruction (see Section 3.4.2.2). We want to evaluate if this approach is capable to capture specific knowledge such as that baked goods are seasoned with sweeteners.

4.2.1 FLAVORING

This section describes the evaluation of the 'Flavoring' action core. Figure 4.1 shows the test sentences which were contained in our self-designed corpus. To avoid a look up of the correct senses and roles, we consider entities which do not appeared in the training set.

```
Flavor the neck and turkey with garlic and cayenne.
Flavor the waffles with honey.
Season the cupcakes with cinnamon.
```

Figure 4.1: Flavoring sentences contained in the self-designed corpus

We developed these sentences to evaluate three features. First, we want to show that our system can handle synonym verbs. In this evaluation the system should map the verb "season" to the 'Flavoring' action core even that the word "season" was not in the training set.

The second feature is that the system should extract multiple frames of a single sentence. A correct implementation should extract 4 frames of the sentence "flavor the neck and turkey with garlic and cayenne" where one frame represent the semantic information "flavor the neck with garlic". The other 3 frames should represent "flavor the neck with cayenne", "flavor the turkey with garlic" and "flavor the turkey".

Since we are not using a probabilistic model to represent and retrieve the extracted knowledge, we have to verify that our knowledge representation and retrieval algorithm is suitable to represent the knowledge of the given corpus. The idea of this corpus is to represent the knowledge that baked goods are seasoned with sweetener and meat with more spicy ingredients.

The first step of this evaluation was to extract the frames of the test sentences. After our information extraction system processed the given corpus, we evaluated the extracted knowledge. The result was that the system extracted 6 frames. The 4 mentioned frames of the first sentence and respectively 1 frame of the second and of the third sentence. All frames represented the 'Flavoring' action core. For each frame the corresponding senses and roles were inferred correct.

To evaluate our information retrieval algorithm, we applied our algorithm on the incomplete instructions "flavor the pancake" and "flavor the steak". The given object list contained the items honey, sugar, sirup, pepper, salt and chili powder. Figure 4.2 shows the completion results.

The structure of Figure 4.2 is build as follows: The synsets represented in the bars are the synsets which are contained in the object list. The synset which is mentioned in the caption of the bar chart represents the ***goal*** action role which is given in the incomplete task. We do not mention the action verb synset since it remains the same for all incomplete tasks. The attached values to the bars represent the confidence score which is determined by our information retrieval algorithm. This score can be interpreted as a representation how suitable the role is to complete the given instruction. A suitable role is a role which completes the given instruction in a way that the instruction can be transformed in an executable plan. The synsets with the highest score are highlighted in green.

We explain this structure by the results for the instruction "flavor the pancake" which are depicted in Figure 4.2 (A). PRAC infers that this instruction activates a 'Flipping' action core with "pancake" having the sense pancake.n.01 and that it is representing the role ***goal***. The information retrieval process suggests that honey is the most suitable role to complete the instruction. Honey gets the highest score because it is explicitly mentioned in the corpus. However, through the taxonomic knowledge the algorithm is able to represent that sugar and sirup are more suitable to flavor a pancake instead of pepper and salt. Pepper and salt get such high confidence scores since the system only include taxonomic knowledge. It does not concern such knowledge that people would probably not eat a pancake which is seasoned with chili powder.

Based on Figure 4.2, the evaluation shows that PRAC is able to retrieve the information that baked goods are seasoned with sweetener and meat with more spicy ingredients.

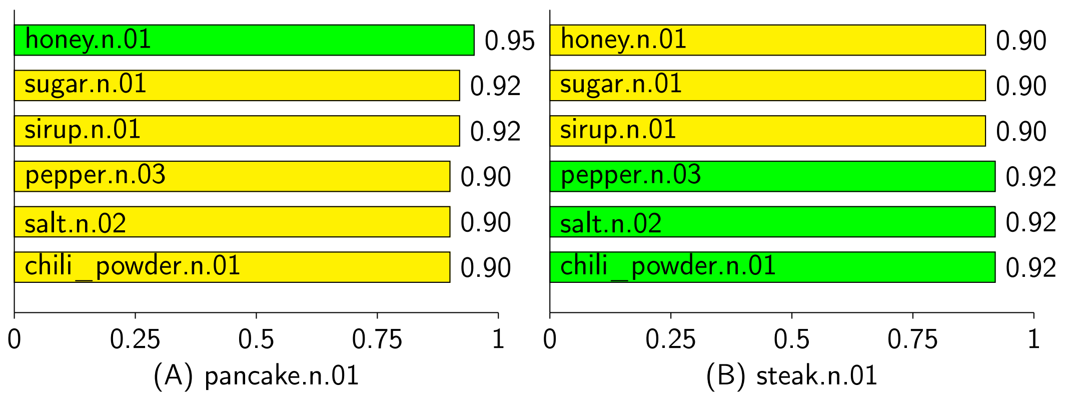


Figure 4.2: Flavoring completion results with a given object list

4.2.2 NEUTRALIZING

The evaluation of the 'Neutralizing' action core shows that our system can handle action cores of other domains and not only of the kitchen domain. Also, we show that our system can perform correct knowledge extraction without using the syntax as evidence. Instead, it is only using the knowledge of the taxonomy. We used the sentences in Figure 4.3 to show these points.

```
Neutralize the titanic acid with glyoxaline.
Neutralize the nitrous acid with cyanuramide.
Neutralize the calcium hydroxide with aqua regia.
Neutralize the potassium hydroxide with malonylurea.
```

Figure 4.3: Neutralizing sentences contained in the self-designed corpus

The following evaluation steps were performed like in the 'Flavoring' evaluation in Section 4.2.1. All frames of the given corpus were extracted correct and the incomplete instruction were "neutralize the bromic acid" and "neutralize the potash". Figure 4.4 shows that the system learned that to neutralize an acid it has to use a base and vice versa. The high value of potash.n.01 in the first query can be explained by the WordNet's design choice that "potassium hydroxide", which appears in the test corpus, maps to the synset potash.n.01.

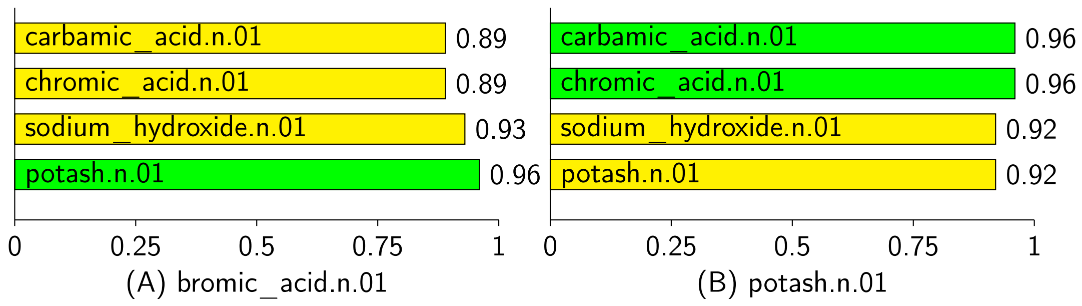


Figure 4.4: Neutralizing completion results with a given object list

4.3 EVALUATION WITH WIKIHOW CORPUS

The evaluation in Section 4.2 shows that our system creates reasonable results on a self-created corpus. To demonstrate that our solution can also perform on real data we use articles from WikiHow. For the evaluation we do not consider all articles from WikiHow. We considered only the articles of the Food-and-Entertaining category. Since our action cores 'Flipping' and 'Storing' represent actions of the kitchen domain, we decided to ignore the articles of the other categories such as Sports-and-Fitness. These articles like "Bake Oatmeal Bread" and "Write a Message on a Cake" describe in multiple sentences how to perform these tasks.

This section is dived in 3 subsections. In the first subsection we present some general statistics about the evaluation corpus. In addition, we evaluate the general extraction and present e.g. the number of extracted frames. Afterwards, we perform the evaluation of the 'Flipping' and 'Storing' action cores by applying our two completion algorithms (see Section 3.4.2.2) on the extracted knowledge of WikiHow corpus. The results are presented in the remaining 2 subsections.

4.3.1 GENERAL EVALUATION

The Tables 4.9 and 4.10 depict how many articles we used for the evaluation and the results of the extraction process.

Measures	Absolute Numbers
Articles	8784
Sentences	58612
Extracted Frames	192181

Table 4.9: Statistics about the subset corpus of WikiHow

Action Core	Absolute Numbers of Frames	Percentage
Unknown	129888	67.6%
Neutralizing	50840	26.5%
Flipping	7187	3.7%
Storing	3827	2%
Flavoring	439	0.2%

Table 4.10: Statistics about the extracted frames

Extracting so many ’Neutralizing’ frames shows that due to the WCSP inference (see Section 3.3.3) we have a high chance of creating the wrong frames. So the next evaluations have to show that even wrong frames exist, the system can still retrieve reasonable objects for the missing roles. Due to the large amount of extracted frames, we are not able to evaluate if all senses are inferred correct. Instead, we evaluate the senses in the frames which we retrieved during the evaluation. We argument that if the extraction system would only infer wrong senses, then we would not able to infer reasonable results since the retrieval algorithms consider the semantic similarity between the given instruction and the sentences in the corpus.

4.3.2 FLIPPING

Instruction Completion through Retrieving Semantic Similar Frames Table 4.10 shows that there are 7187 frames which activated the ’Flipping’ action core. However, not all will be considered for instruction completion due to the constraints which we added to the retrieval algorithm. From the 7187 frames only 211 contained all three ’Flipping’ action roles. This result shows that a coreference resolver would improve the retrieval results since resolving of pronouns would allow inferring all roles from a sentence like "flip it with a spatula". At its current state, our information retrieval system infers only the action verb and the utensil from the given sentence.

Of the 211 frames only 17 passed the verb similarity check. As a remainder, this constraint defines that if the WUP similarity between the action verb of the instruction and the extracted frame is below 85%, then this frame should be revoked.

These 17 frames were used to complete the instructions "flip the pancake" and "flip a steak". Table 4.11 shows the 5 highest scored frames for "flip the pancake" and Table 4.12 for "flip a steak".

Confidence Score	Sentence	Action Roles	Senses
97%	If youre skillful, you can flip the crepe with a quick action of the wrist and no spatula.	Action Verb	flip.v.08
		Obj_to_be_flipped	crape.n.01
		utensil	wrist.n.01
93%	To flip a crumpet, remove the ring using tongs, and then flip the crumpet with a spatula.	Action Verb	flip.v.08
		Obj_to_be_flipped	crumpet.n.01
		utensil	spatula.n.01
92%	Turn the pancake with the help of turner and let it cook on the other side till get brown spots.	Action Verb	turn.v.01
		Obj_to_be_flipped	pancake.n.01
		utensil	turner.n.08
87%	Do not try to flip food with a skillet that is too heavy for you to easily control.	Action Verb	flip.v.08
		Obj_to_be_flipped	food.n.02
		utensil	frying_pan.n.01
77%	Flip the steak over and repeat steps 1-3 with the other side.	Action Verb	flip.v.08
		Obj_to_be_flipped	steak.n.01
		utensil	side.n.05

Table 4.11: The 5 highest scored frames for "flip a pancake"

Confidence Score	Sentence	Action Roles	Senses
100%	Flip the steak over and repeat steps 1-3 with the other side.	Action Verb	flip.v.08
		Obj_to_be_flipped	steak.n.01
		utensil	side.n.05
87%	Do not try to flip food with a skillet that is too heavy for you to easily control.	Action Verb	flip.v.08
		Obj_to_be_flipped	food.n.02
		utensil	frying_pan.n.01
84%	With great_care, invert the pan over the plate, then holding them firmly, turn the plate and pan over.	Action Verb	turn.v.01
		Obj_to_be_flipped	plate.n.07
		utensil	great_care.n.01
83%	Flip over each piece of chicken with a fork and wait approximately 4-6 minutes for the other side to whiten and cook through.	Action Verb	flip.v.08
		Obj_to_be_flipped	chicken.n.01
		utensil	fork.n.01
77%	To flip a crumpet, remove the ring using tongs, and then flip the crumpet with a spatula.	Action Verb	flip.v.08
		Obj_to_be_flipped	crumpet.n.01
		utensil	spatula.n.01

Table 4.12: The 5 highest scored frames for "flip a steak"

The first thing to recognize is that all extracted frames are semantically similar to the given instructions. These frames describe in general how to flip a specific food. In addition, the inferred senses for the roles turn out to be suitable. Some exceptions are "wrist" and "side" but these words are of domains which are not represented in the 'Flipping' MLN. An interesting result is also the resulted sense for the *obj_to_be_flipped* in the sentence "with great care, invert the pan over the plate, then holding them firmly, turn the plate and pan over" in Table 4.12. This synset represents an object of the food domain. However, in the context of this sentence "plate" represents a dish. This examples shows that for the inference process there are use cases where relational knowledge is important to consider.

These frames contain some roles which are suitable to complete the instruction so that the robotic agent can perform it. For the "flip a pancake" example the objects "spatula" and "turner" are suitable. Table 4.12 shows that a "fork" and a "spatula" can be used to flip a steak. Unfortunately, these frames are not the highest score frames. One flaw of this completion algorithm is that the proposed missing roles are not evaluated how suitable they are to complete the instruction. One approach to solve this problem could be to implement a common sense knowledge such as that the robot needs an instrument to perform a 'Flipping' task. However, since it requires a lot of effort to design such a common sense knowledge, we added this task to our future research agenda. An additional task for our future research is to consider handling negative sentences such as "do not try to flip food with a skillet that is too heavy for you to easily control". Handling negative sentences is not quite as simple. The extracted sentences are directed to human readers. Since we are intending that a robot uses this knowledge, the robot has to infer if such constraints apply to it as well.

In conclusion, we can say that PRAC would provide reasonable results to complete the given instructions. For the "flip a pancake" example, the robot would revoke the role "wrist.n.01" since it is not an object which can be grabbed and operated by the agent. However, the spatula is suitable to perform the given task. A similar argumentation can be given for the "flip a steak" example. The role "side.n.05" would be possibly revoked. However, an interesting scenario to consider is the handling of the role "frying_pan.n.01". It would depend if the plan describes how to flip an object with the pan in which this object is contained. If the system would revoke the frying pan as a suitable role, then the role "fork.n.01" would be suggested and this object can be used to flip the steak.

Instruction Completion through Inferring Missing Roles of an Object List For this evaluation we consider an object list containing tongs, spatula, fork, bowl, plate and coffee maker to complete the instruction "flip a pancake", "flip a steak", "flip a coin", "flip a patty", "flip an omelet" and "flip the bacon". Figure 4.5 shows the results for the 6 queries with the given object list.

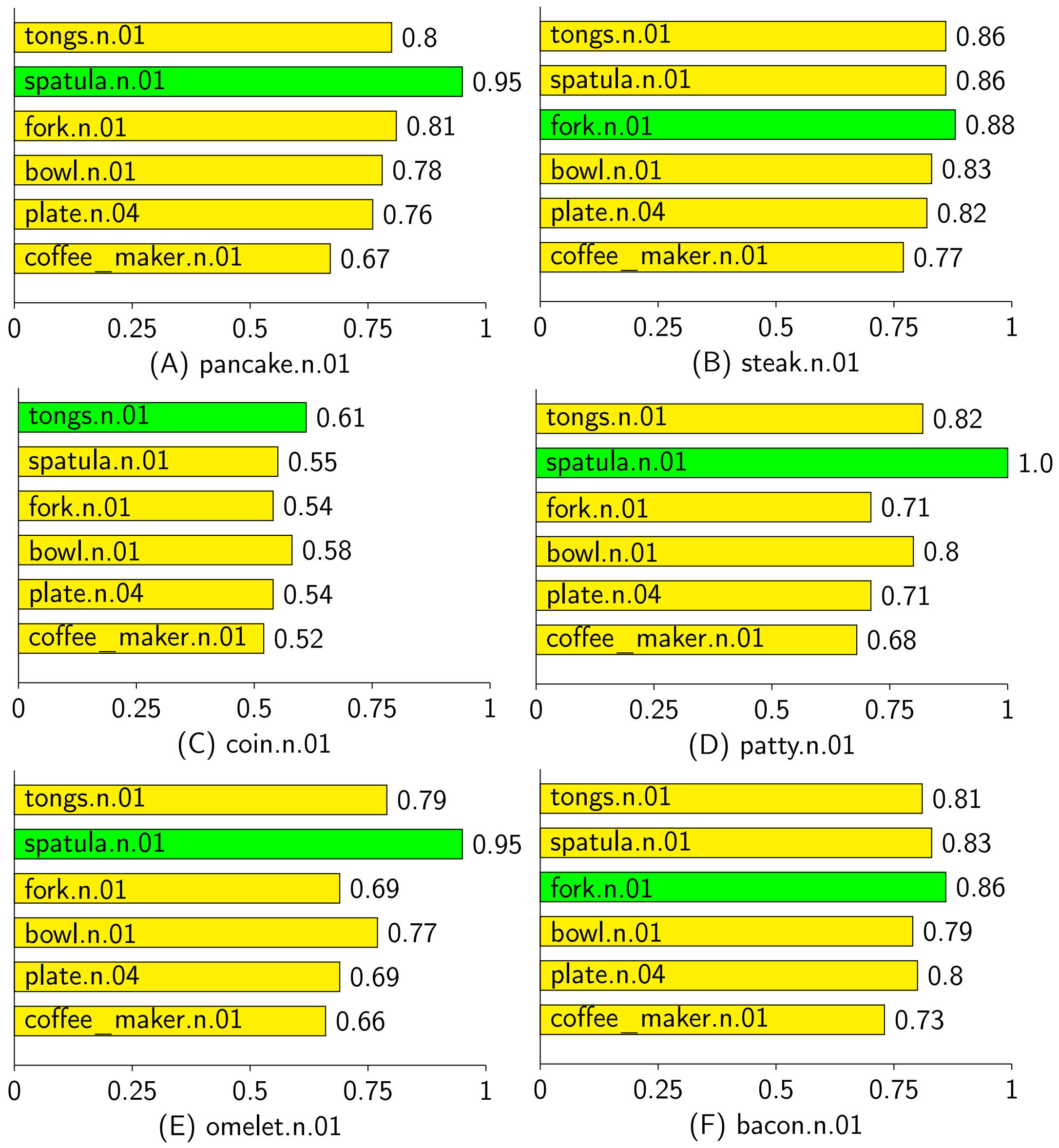


Figure 4.5: Flipping query results with a given object list

The evaluation shows that for every query, PRAC returns reasonable results. Every object with the highest confidence score represents an object which can be used to flip the given objects. Interesting is the result of the "flip a coin" example. Since there is no mention of a coin or a similar object in the corpus, the system illustrates this by scoring these objects lower compared to the other queries.

These results show also a problematic impact of wrong frames stored in the knowledge base. In some query results the object "bowl.n.01" is scored with a high confidence value. This can be explained by the frame with the sentence "flip the steak over and repeat steps 1-3 with the other side" (see Table 4.12). It turns out that the WUP similarity between side.n.05[3] and bowl.n.01 is 0.65. To handle this prob-

[3]Synset description for side.n.05 in WordNet is: "An extended outer surface of an object"

lem there are multiple possibilities. The first one could be to remove wrong frames from the knowledge base. Obviously, evaluating thousands of frames is highly time consuming. A second option could be to use a different taxonomy to calculate different similarities. Setting the confidence limit over 90% can solve that problem too but this can exclude some correct results (see e.g. Figure 4.5 (F)). This observation shows also the difficulty to set a confidence limit. The difficulty is that a limit which is set to high can revoke some reasonable solutions to complete instruction. However, setting the confidence score limit to low can cause that the robot operates with objects which are not suitable to perform the instruction. We mention in Section 5.2 how we are intending to tackle this challenge in our future research.

4.3.3 STORING

Instruction Completion through Retrieving Semantic Similar Frames From the extracted 3827 'Storing' frames, 209 remained which contain all three action roles and 109 which remained after the verb similarity check. Unfortunately, no frame of the 109 describes where to store containers or other kitchen utensils. Due to this result, the evaluation considers only food objects in the incomplete tasks.

These 109 frames were used to complete the instructions "store the sirup" and "store the turkey". Table 4.13 shows the 5 highest scored frames for "store the sirup" and Table 4.14 for "store the turkey".

Confidence Score	Sentence	Action Roles	Senses
89%	By taking time to store basil in the fall, you can enjoy the fresh basil flavors throughout the year.	Action Verb	store.v.02
		Obj_to_be_stored	basil.n.03
		location	descent.n.05
89%	Store the mango chutney in an airtight container for up to a week in the refrigerator.	Action Verb	store.v.02
		Obj_to_be_stored	chutney.n.01
		location	container.n.01
89%	Store the teriyaki sauce in the refrigerator.	Action Verb	store.v.02
		Obj_to_be_stored	sauce.n.01
		location	refrigerator.n.01
84%	Make sure to keep the ingredients in place and that the rice sticks together.	Action Verb	keep.v.03
		Obj_to_be_stored	ingredient.n.03
		location	topographic_point.n.01
83%	You can keep the sauce for up to 3 weeks in the fridge.	Action Verb	keep.v.03
		Obj_to_be_stored	sauce.n.01
		location	electric_refrigerator.n.01

Table 4.13: The 5 highest scored frames for "store the sirup"

Confidence Score	Sentence	Action Roles	Senses
84%	Store the food in the pantry, fridge or freezer.	Action Verb	store.v.02
		Obj_to_be_stored	food.n.02
		location	pantry.n.01
84%	Store the food in the pantry, fridge or freezer.	Action Verb	store.v.02
		Obj_to_be_stored	food.n.02
		location	deep-freeze.n.01
84%	Store the food in the pantry, fridge or freezer.	Action Verb	store.v.02
		Obj_to_be_stored	food.n.02
		location	electric_refrigerator.n.01
83%	Remember to store the ribs in your refrigerator and only remove them an hour before smoking so that they are at room_temperature when it is time to begin cooking.	Action Verb	store.v.02
		Obj_to_be_stored	rib.n.03
		location	refrigerator.n.01
83%	Keep the horsemeat in correct conditions.	Action Verb	keep.v.03
		Obj_to_be_stored	horsemeat.n.01
		location	condition.n.01

Table 4.14: The 5 highest scored frames for "store the turkey"

The results show that the most of the extracted frames are semantically similar to the given instructions. The only exception is the frame in Table 4.13 representing "make sure to keep the ingredients in place". In Table 4.13, the wrong inferred senses for "fall" can be explained that our 'Storing' MLN does not contain the domain "season".

The Corpus Analyzer (see Section 3.1.2.1) did not capture the electronic devices such as "fridge", "freezer" and "refrigerator". This can be explained due to the low occurrence of unique words in the corpus and therefore this domain got a lower score compared to the other proposed hypernyms (see Table 3.6). During the frame extraction, the MLNs inferred the correct senses because the devices activate only one synset in WordNet. In the future, an additional feature for the Corpus Analyzer can be that it performs a second analysis without the constraint to consider only unique words. Having two different analysis might provide a better overview about the represented domains in the corpus.

The results show that the robotic agent would be able to complete the instruction "store the sirup" successfully. However, the results for "store the turkey" can be also used to complete the instruction but they require additional evaluation. The frames representing "store the food in the pantry, fridge or freezer" are the highest scored frames. However, they represent a more general situation compared to the frame "store the ribs in your refrigerator". For our example it would be suitable to rate "store the ribs in your refrigerator" frame higher compared to the other 3. We decided to use the WUP similarity to avoid such scenarios (see Section 3.4.1). In this scenario "turkey.n.04" and "rib.n.03" are not sister terms and therefore the similarity between "turkey.n.04" and its hypernym "food.n.02" is higher than the similarity between "turkey.n.04" and "rib.n.03". To solve this scenario there are multiple options. The first option could be to perform an evaluation that the turkey can be stored in a fridge and in a freezer but not in the pantry. An addition option could be to consider a taxonomy where turkey and rib are sister terms.

Instruction Completion through Inferring Missing Roles of an Object List For this evaluation we consider an object list containing oven, fridge, jar, coffee maker, fork, blender and bowl to complete the instruction "store the sirup", "store the paprika", "store the eggs", "store the coffee", "store the turkey" and "store the pepper". Figure 4.6 shows the results for the 6 queries with the given object list.

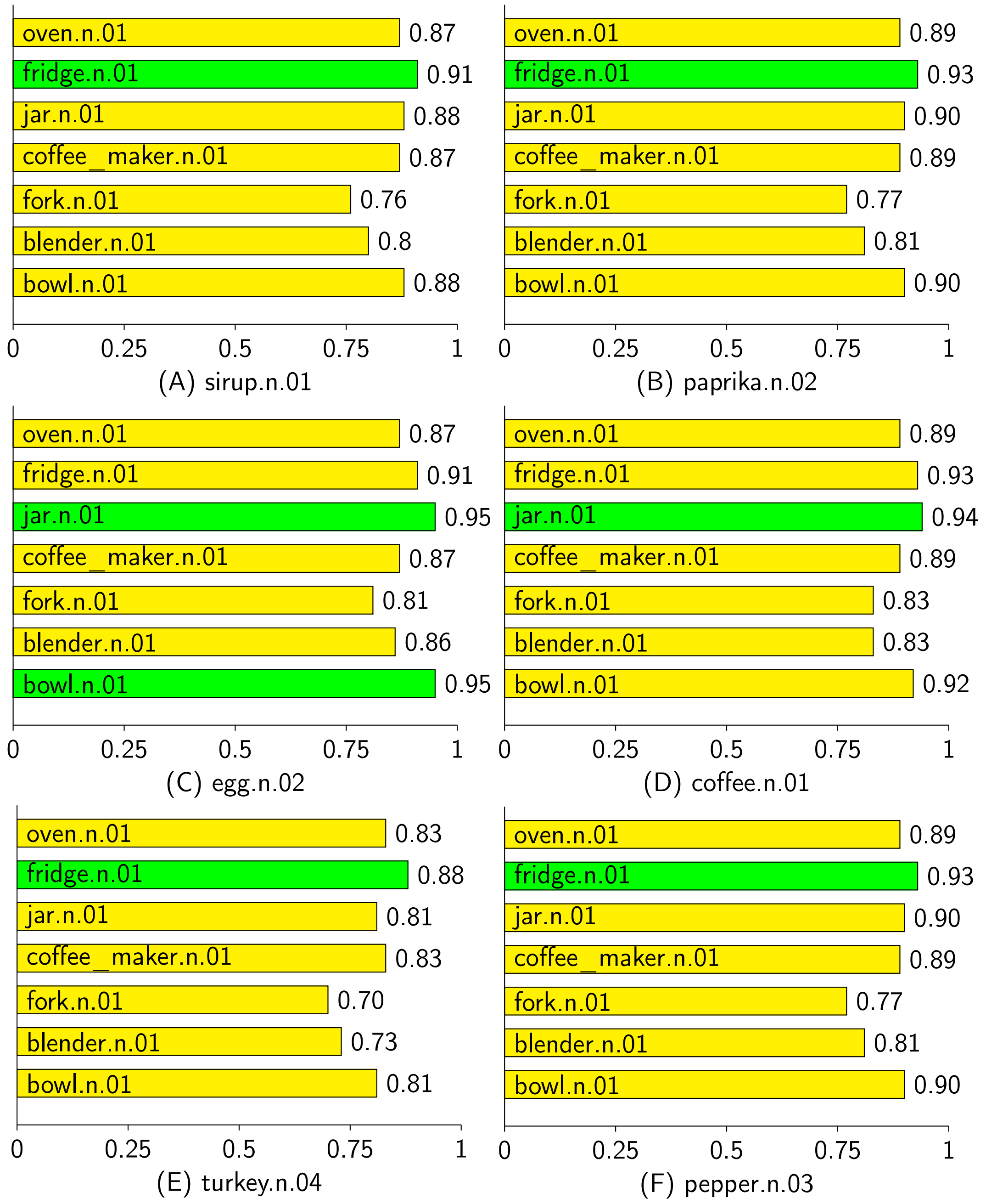


Figure 4.6: Storing completion results with a given object list

The evaluation shows that for every query, PRAC returns reasonable results. Every object with the highest confidence score represents an object which can be used to store the given object. The result for pepper in Figure 4.6 (F) can be explained by the fact that in the WikiHow corpus there is no information provided where to store spices. So the retrieval algorithm determined the confidence score for the completion "store the pepper in the fridge" by considering the extracted frame "store your onions in the freezer". Additionally, the evaluation shows that for some action cores it might be better to use a different taxonomy. For instance, WordNet does not differentiate the kitchen electronic by their functionality. This is the reason why "coffee maker" gets such high confidence values since it is similar to "fridge" which is contained in the frames stored in the knowledge base. Also, WordNet categorizes "fork" and "cup" as tablewares. This is the reason why "fork" gets a high confidence score in Figure 4.6 (D). The algorithm determines that the stored frame "keep some soda or water in your cup at all times" is the semantically most similar frame to the completion "store the coffee in a fork".

CHAPTER five

CONCLUSION

5.1 SUMMARY AND CONTRIBUTIONS

In this thesis we presented an information extraction system which uses fuzzy MLNs to extract semantic knowledge of natural-language documents. Our system differs in the way that it does not use a text-based knowledge representation to store the extracted knowledge. Instead, it annotates the extracted sentences with semantic information. Using the extracted knowledge, PRAC is able to infer the missing roles and to score the results based on the semantic. Our knowledge representation and information retrieval algorithm are not using probabilistic models such as MLNs. Instead, we use JSON for the knowledge representations and simple arithmetics for our retrieval algorithm. Our information retrieval algorithms scale well on large knowledge bases. We presented two algorithms which are able to complete instruction based on the sentences stored in the knowledge base. The worst case running time for our first retrieval algorithm is linear and depends on the number of frames stored in the database. The second algorithm requires an exponential running time but it depends only on the number of missing roles and possible objects which should be used to complete the instruction. In the practice this approach can be tractable if number of objects in the list is kept small.

We evaluated our information extraction system and algorithms on a self-created corpus and on real articles from WikiHow. We showed that our MLNs infer the correct senses and the roles on unseen objects. Our information retrieval algorithms provided reasonable results for the given incomplete instructions.

In conclusion, we can say that we developed a solution which overcomes the scaling problem of joint probability distributions and it is still able to complete robot instructions.

At the end of this section we want to give a recommendation how to utilize the presented contributions such that the robotic agent is able to complete instruction without having a high risk of causing accidents in its environment. This recommendation assumes that the WordNet taxonomy is used to represent the objects. In its current state, we recommend to use the completion algorithm which considers the objects in the robotic agent's environment. The benefits of the algorithm are that the objects in the environment will be considered to solve the task and also it is able to capture the similarity between the objects in the environment and the objects in the corpus. The completion algorithm which retrieves semantic similar frames to complete the instructions is capable to provide suitable missing roles too but it is required that the mentioned roles exist in the environment of the robot. In addition, we would recommend to set the confidence limit to 90%. The high confidence limit revokes possible frames which are to abstract to provide a correct solution. An abstract frame can be a frame like "store the food in the pantry". Such a frame is not suitable to describe e.g. where to store meat. Also, a high confidence limit reduces the risk of causing accidents in the real word.

In practice this approach should not require high computational effort. Currently, PRAC contains action cores which have a maximum number of 5 roles. So a natural-language instruction which actives this action core can only have a maximum number of 3 missing roles. Even tough the object list would contain 100 objects, a modern computer system is capable to perform the algorithm without effort.

5.2 Future Work

The future work should be concentrating on improving the natural-language processing and action role inference.

With an improved natural-language processing our system will be able to extract more information. Also, the chance of having wrong frames will decrease. One part to improve is the parsing process. We stated out that even the used Stanford Parser is not perfect. To implement an information extraction system, which should perform on an open-domain, has to have a great parser. Our proposed solutions to handle the imperative sentences can have a negative impact on other sentences. A possibility could be to try Deep Learning parsers.

During the evaluation, we showed that many frames cannot be considered for the missing role inference due to our constraint that the frames in the knowledge base have to contain all action roles and missing roles such as the given instruction. Some roles in the corpus cannot be inferred due to the missing coreference resolution. At the end of the thesis, the development of a coreference resolver has been started and the intention is to improve our information extraction with it.

There are some additional natural-language tasks, such as handling passive sentences and negative sentences. To handle passive sentences, we intend to transform the passive sentences to active sentences and not design them in the MLNs. This will reduce the complexity for the training and inference process. Handling negative sentences requires additional inference processes. Consider the sentence "do not try to flip food with a skillet that is too heavy for you to easily control". The extracted sentences are directed to human readers. Since we are intending that a robot uses this knowledge, the robot has to infer if such constraints apply to it as well. Some other features to improve are the compound noun processing and the inference of expressions such as "piece of chicken" to "chicken".

The second component to improve is the action role inference. Currently, it costs much time to add the support of new action cores. The creating process of training data and to keep this set balanced consumes the most time. It would be great to speed up the process by creating this training set automatically. This could be achieved by improving the Corpus Analyzer or use other unsupervised algorithms.

Besides the training set creation, it is necessary to improve components regarding Markov logic networks. For instance, developing an incremental learning algorithm or improving the inference algorithms to create a probability distribution.

During the evaluation, we mentioned that it would be good to add an evaluation of the proposed missing roles. This can be achieved by implementing a common sense knowledge or at least consider action-specific knowledge, such as that the utensil of a "Flipping" action core has to be of the domain "kitchen utensil".

For this thesis we defined action cores having only three action roles. It would be interesting to evaluate the information retrieval algorithm with querying for multiple missing roles. Another additional evaluation could be to use a different taxonomy than WordNet. A more sufficient approach to evaluate the inferred senses of the extracted frames will be also good to have.

At the moment, we have to set the confidence limit for each action core manually. A possible approach to determine this limit by the system could be to represent this problem as a regression problem. Then, the system can learn by experience how low the confidence level can be set.

Currently, our knowledge base representation is designed for one use case only. For instance, it cannot be applied to determine the most suitable WikiHow article to perform a specific task. There are two approaches to handle this problem. One solution can be to create a single knowledge representation which suits multiple use cases. The first approach has the benefit that the developer has to maintain only one database. A disadvantage can be a slow performance on large data sets since the knowledge representation is not designed specific for the use case. However, the second solution considers multiple knowledge bases, each designed to speed up the queries for each use case.

At last, we propose two additional features. The first feature is to add an interactive mode where the system can ask the user to provide some answers if the system recognizes that the missing information is not contained in the knowledge base. The second feature is to use the knowledge base to apply aggregate statistics to create a distribution for example what kind of objects are stored in the fridge or pantry. PRISMATIC provides some algorithms for such tasks.

REFERENCES

[1] Root-Mean Square-Arithmetic Mean-Geometric Mean-Harmonic mean Inequality. https://artofproblemsolving.com/wiki/index.php?title=Root-Mean_Square-Arithmetic_Mean-Geometric_Mean-Harmonic_mean_Inequality, . Accessed: 2016-05-27.

[2] Harmonic mean. https://artofproblemsolving.com/wiki/index.php?title=Harmonic_mean, . Accessed: 2016-07-10.

[3] WordNet Similarity. http://search.cpan.org/dist/WordNet-Similarity/lib/WordNet/Similarity/path.pm. Accessed: 2016-04-01.

[4] Alphabetical list of part-of-speech tags used in the Penn Treebank Project. https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html. Accessed: 2016-05-27.

[5] Stanford Parser FAQ. http://nlp.stanford.edu/software/parser-faq.shtml#z. Accessed: 2015-12-27.

[6] What is WordNet? http://wordnet.princeton.edu/wordnet/. Accessed: 2015-12-27.

[7] G. Bezhanishvili and L. S. Moss. Undecidability of first-order logic.

[8] S. R. Branavan, H. Chen, L. S. Zettlemoyer, and R. Barzilay. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 82–90. Association for Computational Linguistics, 2009.

[9] R. Bunescu and R. Mooney. Statistical relational learning for natural language information extraction.

[10] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, and T. M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3, 2010.

[11] K.-U. Carstensen, C. Ebert, C. Ebert, S. Jekat, R. Klabunde, and H. Langer. *Computerlinguistik und Sprachtechnologie: Eine Einführung*. Spektrum, Heidelberg, 3 edition, 2009.

[12] M. Collins. Probabilistic context-free grammars (pcfgs). 2013. URL http://u.cs.biu.ac.il/~89-680/collins-pcfgs.pdf. Accessed: 2015-12-23.

[13] M.-C. De Marneffe and C. D. Manning. Stanford typed dependencies manual. Technical report, 2008.

[14] L. De Raedt and K. Kersting. Statistical relational learning.

[15] O. Etzioni, M. Banko, S. Soderland, and D. S. Weld. Open information extraction from the web. *Communications of the ACM*, 51(12):68–74, 2008.

[16] J. Fan, A. Kalyanpur, D. Gondek, and D. A. Ferrucci. Automatic knowledge extraction from documents. *IBM Journal of Research and Development*, 56(3.4): 5–1, 2012.

[17] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79, 2010.

[18] C. Goutte and E. Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *European Conference on Information Retrieval*, pages 345–359. Springer, 2005.

[19] D. Jain, P. Maier, and G. Wylezich. Markov logic as a modelling language for weighted constraint satisfaction problems. *Constraint Modelling and Reformulation (ModRef'09)*, page 60, 2009.

[20] D. Klein and C. D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics, 2003.

[21] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[22] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox. Learning to parse natural language commands to a robot control system. In *Experimental Robotics*, pages 403–415. Springer, 2013.

[23] M. C. McCord, J. W. Murdock, and B. K. Boguraev. Deep parsing in watson. *IBM Journal of Research and Development*, 56(3.4):3–1, 2012.

[24] D. K. Misra, K. Tao, P. Liang, and A. Saxena. Environment-driven lexicon induction for high-level instructions.

[25] D. Nyga and M. Beetz. Everything robots always wanted to know about housework (but were afraid to ask). In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vilamoura, Portugal, October, 7–12 2012.

[26] D. Nyga and M. Beetz. Cloud-based Probabilistic Knowledge Services for Instruction Interpretation. In *International Symposium of Robotics Research (ISRR)*, Sestri Levante (Genoa), Italy, 2015.

[27] D. Nyga and M. Beetz. Reasoning about unmodelled concepts-incorporating class taxonomies in probabilistic relational models. *arXiv preprint arXiv:1504.05411*, 2015.

[28] S. Rajasurya, T. Muralidharan, S. Devi, and S. Swamynathan. Semantic information retrieval using ontology in university domain. *arXiv preprint arXiv:1207.5745*, 2012.

[29] M. Richardson and P. Domingos. Markov logic networks. *Machine learning*, 62 (1-2):107–136, 2006.

[30] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009. ISBN 0136042597, 9780136042594.

[31] S. A. Tellex, T. F. Kollar, S. R. Dickerson, M. R. Walter, A. Banerjee, S. Teller, and N. Roy. Understanding natural language commands for robotic navigation and mobile manipulation. 2011.

[32] Z. Wu and M. Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32Nd Annual Meeting on Association for Computational Linguistics*, ACL '94, pages 133–138, Stroudsburg, PA, USA, 1994. Association for Computational Linguistics. doi: 10.3115/981732.981751. URL `http://dx.doi.org/10.3115/981732.981751`.