CHALMERS
UNIVERSITY OF TECHNOLOGY

Automated semantic grammar generation in dialogue-based task-oriented systems

Master's thesis in COMPLEX ADAPTIVE SYSTEMS

GEORGIOS TRIANTAFYLLOU

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
DIVISION OF VEHICLE ENGINEERING AND AUTONOMOUS SYSTEMS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

MASTER'S THESIS IN COMPLEX ADAPTIVE SYSTEMS

Automated semantic grammar generation in task-oriented dialogue-based systems

GEORGIOS TRIANTAFYLLOU

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2021

Automated semantic grammar generation in task-oriented dialogue-based systems
GEORGIOS TRIANTAFYLLOU

Master's thesis 2021:56
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Chalmers Reproservice
Göteborg, Sweden 2021

Automated semantic grammar generation in task-oriented dialogue-based systems
Master's thesis in Complex Adaptive Systems
GEORGIOS TRIANTAFYLLOU
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology

Abstract

Dialogue-base task-oriented systems are conversational systems that can complete tasks and answer questions. A semantic grammar can be helpful for user input recognition in such systems. Manually creating semantic grammars can be cumbersome for a developer and delay the deployment of the system, thus it would be of great assistance to automate the process. The topic of automated semantic grammar generation has not been deeply explored and current approaches in this field are either semi-automated or lack interpretability and robustness.

In this thesis, the prospect of developing a completely automated semantic grammar generation method was investigated. An additional aim was to create a method that could be employed in different domains without needing any additional modifications. We propose a novel method that utilizes syntactic and relation analysis to infer the semantics of a user input. The method developed is a hybrid approach that comprises statistical methods for the syntactic analysis and a rule-based model for the semantics. The results show that the method is able to generate a fairly accurate and precise semantic grammar. It is also observed that the method is not able to fully analyze all possible pattern cases. Furthermore, the relation analysis has proved to be helpful on finding semantic synonymity amongst different user input patterns. It is plausible to completely automate the creation of a semantic grammar and our findings suggest that hybrid approaches can preform well in this task. Nevertheless, further adjustments should be made in the rule-based model to provide a universal coverage of pattern cases.

Keywords: Semantic grammar, conversational systems, syntactic analysis, semantic analysis, information extraction

Preface

It has been a pleasure to be on such an exciting journey. I started this project with the hopes of automating the generation of semantic grammars and on the way I deepened my knowledge of natural language processing and understanding, as well as semantic analysis.

Acknowledgements

I would like to thank my supervisor, Professor Mattias Wahde, first for his insights and feedback throughout this thesis and second for the opportunity to explore this interesting research topic. I would also like to thank my parents for their continuous support throughout my studies. In addition, I would like to thank my friends for their support. They provided me with valuable social interactions during these COVID-19 times. Lastly, I would like to thank my partner Lynsay for encouraging me and providing linguistic insights throughout my thesis work.

Contents

1 Introduction

Dialogue-base task-oriented systems are conversational systems that can complete tasks and answer questions. These systems have become popular during the last decades due to a growing academic and industrial interest [1]. In order for such systems to be able to achieve high functionality and become useful assets to every day life, they need to hold certain attributes. One of the most important functions of these systems is the ability to understand the user's requests, whether they are in spoken or written language. A dialogue-based task-oriented system must be able to distinguish different user input patterns (questions or requests) and identify the correct response. Dialogue-based systems can utilize natural language understanding techniques to detect the user's intent. Certain systems can analyze the user's input by dividing it into phrases that are mapped onto specific labels based on a semantic grammar [2].

A semantic grammar is a grammar with rules that contain label-phrase pairs [3] and can be created manually by the developer of the system [4]. These label-phrase pairs can include one or more phrases and are labeled based on the context of the system's task domain [3]. An example of a semantic grammar can be observed in Figure 1.1. For the purposes of this thesis, a grammar rule will be referred to as a *semantic category* of the semantic grammar. A semantic grammar can be of great assistance to developers of task-oriented systems as it can help the system analyze a user's input not just based on structural elements, but also the sentence's contextual meaning. Syntactic analysis solely focuses on the structural elements of the sentence, yet a sentence's meaning is not entirely dependent on structural elements. By using a semantic grammar, the system takes into account that the context a sentence is communicated in also effects its meaning. Nevertheless, this kind of grammar might be fragile if subject to different contexts and thus it is recommended for single domain systems [5]. Lastly, a semantic grammar can give the opportunity for an easier development of a task-oriented dialogue-based system.

$$\begin{aligned} \text{[date]} &\longrightarrow \text{on [month\_year] [day\_month]} \\ \text{[month\_year]} &\longrightarrow \text{April} \\ \text{[day\_month]} &\longrightarrow \text{22} \end{aligned}$$

Figure 1.1: *An example of a semantic grammar inspired by [6]. Each line describes a grammar rule, where the left side indicates the label of the rule and the right side indicates the pattern of the rule. The labels '[month_year]' and '[day_month]' refer to words, whereas the label '[date]' refers to a rule that comprises a pattern of words including '[month_year]' and '[day_month]'.*

A dialogue-based system that can utilize a semantic grammar is DAISY [7]. DAISY is a system which consists of dialogue-blocks, or so-called cognitive blocks, that handle user input processing, cognitive processing, and output generation. Each of these parts of the block are represented by specific computational units called input items, cognitive items, and output items. A user input can be expressed in written or spoken language and is assigned to a specific input action based on an input pattern. Input patterns are patterns that DAISY utilizes to map the user's input to an input action. An input item is a computational unit that contains a list of one or more input actions. An input action can either point directly to an output item or a cognitive item. A cognitive item is responsible for completing a task based on the input of the user and can include one or more actions that are called cognitive actions. A cognitive action might also use parts of the user's input, which are represented as dynamic values in the input pattern. Furthermore, when the task has been completed the last cognitive action points to the relative output item. An output item can either be mapped to an input action or a cognitive action and expresses the response of the conversational agent (hereafter: agent) in written and spoken language. A rough scheme of the structure can be observed in Figure 1.2.

Different input patterns that are associated with the same agent response should be assigned to the same input item. A problem often occurring in the development of systems such as DAISY is that, in the case that a plethora of different input patterns should be mapped to the same agent response, manually generating the grammar places a significant burden on the developer. The developer may fail to consider every possible input pattern for a specific input item. A semantic grammar should include complicated nested *semantic categories* that contain every possible input pattern. Therefore, manually generating a semantic grammar could become cumbersome and there is a need for automated methods. This thesis will propose a novel approach for automated generation of a semantic grammar considering the dialogue-based system DAISY, but with the ability for a generalized application to other dialogue-based systems that use a similar semantic grammar.

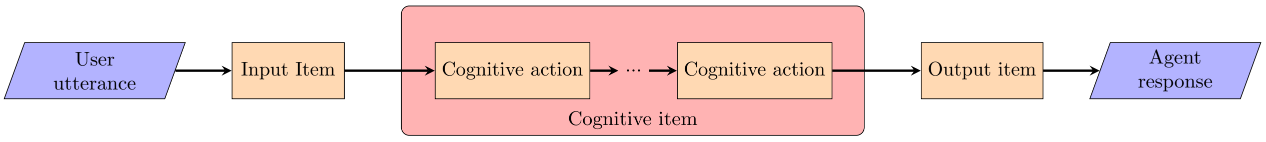


Figure 1.2: *A graph illustrating the structure of DAISY.*

The method will focus on creating a semantic grammar based on a hierarchical ontology, in which a *semantic category* will consist of a label and a set of patterns or words. A *semantic category* is equivalent to a grammar rule. A *semantic word category* is defined as a *semantic category* that contains a label and a set of words. A *semantic phrase category* is defined as a *semantic category* that consists of a label and a set of patterns. These patterns can either be a sequence of semantic *word categories*, a sequence of other *semantic phrase categories* or a combination of both. If a *semantic phrase category* consists of only *semantic word categories* then it is considered a simple *semantic phrase category*, but if it consists of other *semantic phrase categories* it is considered a complex one. A *semantic phrase category* will be referred to as *phrase category*, a *semantic word category* as *word category* and a *semantic category* will also be referred to as *category*. The following set of equations is an example of each case, where the *categories* contain generic labels:

$$\text{[word\_category]} = \text{word\_1, word\_2} \tag{1.1}$$
$$\text{[phrase\_category]} = \text{[word\_category] [word\_category] [word\_category]} \tag{1.2}$$
$$\text{[complex\_phrase\_category]} = \text{[phrase\_category][phrase\_category]} \tag{1.3}$$

These *categories* follow a slightly different way of representing a semantic grammar. Each *category* contains a label enclosed into brackets, but each synonymous word or pattern is divided by a ',' symbol. The '=' symbol is used instead of the '→' symbol to map the *category* with its set of assigned words or patterns. Therefore, the *word category* in Equation 1.1 contains two synonym words and the *phrase categories* in Equations 1.2, 1.3 contain one pattern.

The dataset that was chosen for creating the algorithm was the (6) dialog bAbI tasks (bAbI) [8], which is an unannotated restaurant domain dataset for dialogue-based task-oriented systems provided by the Facebook research team and will be referred to as the bAbI dataset. A small unannotated dataset was manually created to assess the robustness of the method. This dataset will be referred to as the rental dataset, because it is based on the vehicle rental domain. The method utilizes a statistical part of speech tagger and a constituency parser to analyze the syntax of a sentence. Then, heuristic rules are utilized to create *word* and *phrase categories* for the semantic grammar.

2 Related Work

There have been a few tries to generate semantic grammars using automated methods such as in [6, 9, 10]. The methods that will be examined in this section were found to rely mostly on approaches that include an authoring phase. Some of these represent the concepts of the domain in a form of semantic ontology or system representation network. In [6], there is also an initial set of grammar rules that are created. In cases where the methods need to analyze the sentence syntactically or semantically, there is a use of statistical parsers. By studying these approaches, it is observed that heuristic rules are important for creating semantic grammars. Moreover, there is a research interest on creating semantic grammars based on semantic role labels [9].

Automated semantic role labeling (ASRL) methods such as in [11] do not explicitly refer to automated generation of semantic grammars, but could be used to create one, especially considering the method presented in [9]. It could also be argued that semantic role labeling may not be suitable for a creation of a domain-specific semantic grammar, since it may produce grammars with *semantic categories* that contain generic role labels.

2.1 Growing semantic grammar

In [6], the approach proposed focuses on growing an already established semantic grammar. The method considers user feedback and tries to engage the user on correcting misunderstood utterances to compensate for errors. The method is divided into two parts, the authoring phase, where an initial set of semantic grammar rules is created from the developer based on a domain model (DM) and the run time phase, where grammar is extended through automated mechanisms and corrections from the user.

A DM is an acyclic graph consisting of the semantic concepts of the domain. The nodes of the graph represent concepts in the form of labels and the edges represent a concept to subconcept relation between the connected nodes. These relations are expressed as grammar rules of the semantic grammar. This creates a hierarchical structure of concepts defining the topology of the model. An example can be observed in Figure 2.1.

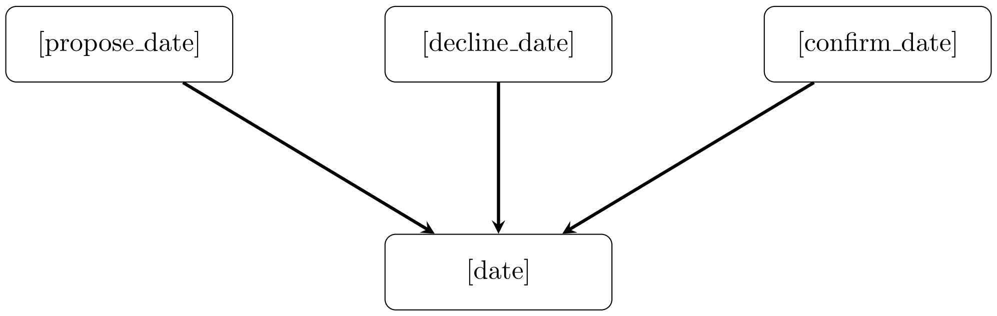


Figure 2.1: *An example of how a concept-subconcept relation is expressed in the hierarchical DM of the method in [6]. Here the subconcept is the label [date] and the nodes connected to it are the concepts in a higher level of the DM*

For the authoring phase, the developer first creates the DM according to the needs of the domain through a so-called domain model editor. Then, a so-called kernel grammar editor prompts the developer to define grammar rules for the kernel grammar that will be used during the run time phase. A grammar rule derived from the semantic grammar in Figure 1.1 and the DM in Figure 2.1 would be the rule '[propose_date]'$\longleftarrow$"Maybe [date]?". The run time phase involves a learning process, that consists of three parts namely, hypothesis formation and filtering, interaction with end-user and dynamic rule creation.

First, the method hypothesizes the rules for the example sentence by finding all the possible parse trees that are approved by the DM and ranks them based on their likelihood. A parse-tree is defined as a rooted ordered tree that represents the structure of a string based on a context-free grammar (CFG) and will be further explored in Subsection 3.3. The failed parse trees are used to predict missing words either before or after the parsed sequence, so that an incomplete sentence could be related to an existing rule of the grammar. For example, if the word 'April' is not assigned to the concept '[month_year]' as in Figure 1.1 and there exists a known rule '[propose_date]'$\longleftarrow$"Maybe [date]?", then in the sentence "Maybe on April?" the method would try to associate 'April' with '[date]'. Another case is when there exists a known pattern such as "Maybe on April?" and an unknown one such as "I would prefer on April". Given that 'April' is associated with a concept such as '[date]', then the method will try to associate the two sentences in the same rule by hypothesizing that 'Maybe' is similar to "I would prefer" in this context.

In addition, a hidden understanding model (HUM) is used to assist the parse predictions mechanism. The HUM includes a speech-act n-gram, a concept-subconcept hidden markov model (HMM) and a concept-word HMM. An n-gram is a sequence of n words that can be found in a text. An n-gram model is a statistical model that uses a given set of n-words to predict the next most probable word. HMMs are markov chains that can find the behaviour of hidden states in a system, based on the observable states of that system. The speech-act n-gram correlates top-level concepts such as '[salutation]' with certain acts that could be assigned after them, such as a request. The concept-subconcept HMM correlates concepts with their immediate subconcepts in the parse tree. This creates a chain of hierarchical semantic labels. The concept-word HMM correlates the terminal nodes (words) of a parse tree with their immediate parent concept. For example, the pre-terminal concept '[month_year]' of a parse tree is coupled with its daughter terminal word 'April'. The aim of the HUM is to capture the patterns that are observed in the DM and could not be found by the parse predictions or the existing semantic grammar rules.

After the hypotheses have been finalized, the end-user decides the most appropriate one. If the end-user does not consider at least one hypothesis as valid, then the user is requested to provide an appropriate paraphrase of the example utterance. Lastly, the chosen hypothesis or end-user paraphrase induces a new grammar rule that is then added to the grammar. A rough scheme of the method can be observed in Figure 2.2.

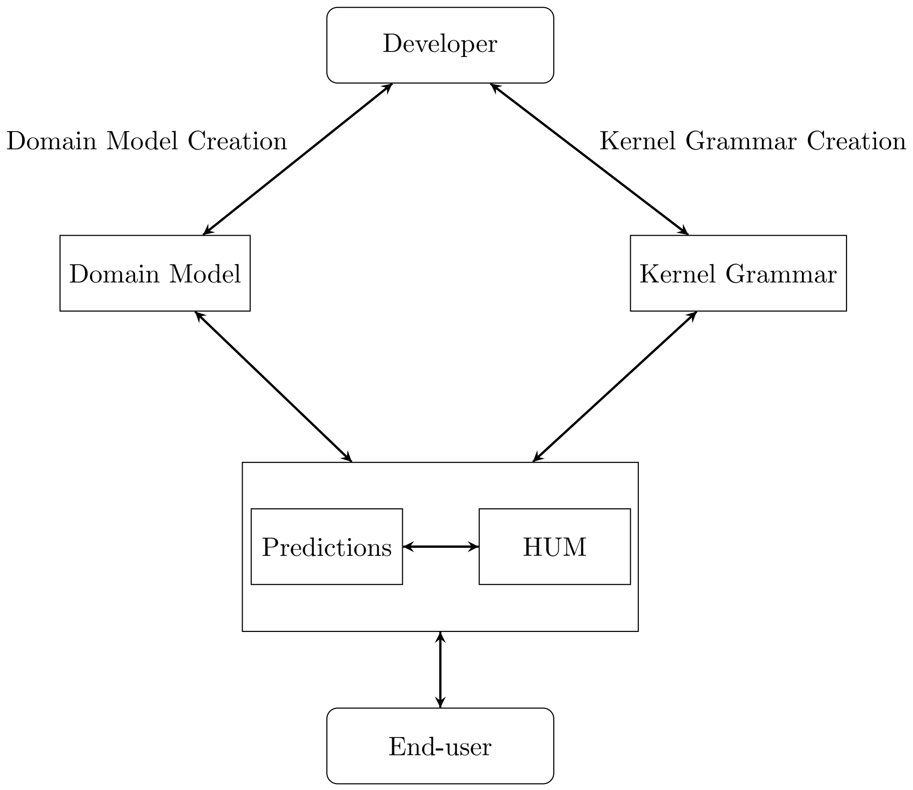


Figure 2.2: *A rough scheme of the method described in [6]*

The goal of the run time phase is to reach universal coverage with the assistance of the user. As it was previously discussed, semantic grammars are domain oriented and are fragile in context-free cases. Therefore, it is easy to understand the choice of using a DM, but there is a substantial amount of author's burden to create the grammar, as well as reliability on the users to extend it. The choice of relying on some sort of assistance from the users should be taken into consideration, but according to the author of this thesis, it should not be the only way of extending the grammar. The reason for that is that the user could create a biased or incorrect grammar that would lead to erroneous predictions and could not be able to generalize well. The scope of the thesis is relevant to the concept that this method proposes, but with the aim to extend on a completely automated approach.

2.2 Semantic representation system

A method for semantic grammar generation, that utilizes heuristic rules for outputting an agent response, is described in [10]. It is notable that this method aspires to create accurate and quick output responses for dialogue-based systems with the use of handwritten training examples. Although the main goal is to generate agent responses, it requires an analysis of utterances. This results in a form of a semantic representation system that could be parallelized to a semantic grammar. For the generation of this semantic system, each training example produces a triplet of type $(u, syntax(u), semantics(u))$, where u corresponds to the training utterance, $syntax(u)$ corresponds to the syntax analysis of the utterance and $semantics(u)$ comprises substrings of the sentence that correspond to semantic categories. The syntax analysis of the sentence is produced by the Charniak parser [12] based on the Penn Treebank constituent format [13].

For the semantic analysis, a semantic representation system is created based on a predefined set of semantic representations. The system uses an attribute-value matrix representation to divide an utterance into a set of variables that correspond to a speech act. These variables are then mapped to so-called frames. A speech act incorporates the semantic parts of an utterance. These include notions such as the actor which is the addresser that expressed the utterance, the addressee, the action which indicates the intent and the semantic content that contains all the other semantic parts of the sentence. These representations must then be linked with the relevant substrings of the sentence that do not need to be consistent with the syntax analysis but need to follow the same tokenization. Moreover, an important feature is that every word of the sentence is assigned to a semantic representation, either by mapping directly the word to a specific semantic representation or through a substring that contains the word. For example, in the case that a user expresses a request such as "I want a car", the user is mapped to the 'speech-act.actor' frame, the agent is mapped to the 'speech-act.addressee' frame, the phrase "I want to" is an action that is mapped to the 'speech-act.action' frame as a request and the phrase "a car" is the value of the request action that is mapped to a 'speech-act.content.value' frame. This approach provides a good example of how creating multi-chunk *semantic categories* could assist agents to better understand user utterances. The advantage of this method is that semantic and syntax analysis are co-utilized to understand the meaning of an utterance. The disadvantage is that a significant author's burden is needed to create the initial semantic representation system.

2.3 Inductive logic programming

In [9], a semantic grammar is created using a supervised inductive logic programming approach. The method primarily focuses on creating semantic grammar rules by analyzing the structure of sentences, semantically and syntactically, but mainly for identifying semantic roles and actions. Inductive logic programming (ILP) is a subfield of symbolic artificial intelligence where some background knowledge theory B combined with positive (E^+) and negative (E^-) examples are used to infer a hypothesis H. A hypothesis h must adhere to specific constraints subject to theory B, and examples E^+ and E. ILP systems mostly consist of two parts, which are hypothesis search and hypothesis selection. The CHILI algorithm utilizes a generalized shift-reduced case-role parser to create the background knowledge theory and induction control rules to produce the hypothesis. The shift-reduce parser maps semantic roles to constituents and connects them through a verb. This is accomplished by striving from the start of the sentence, using a stack and an input buffer containing the original sentence. Then the parser creates action elements that include a verb and the roles that are connected through that verb. For example, the sentence "I bought an apple" would be parsed as:

$$\text{[[bought,pat:[apple,det:an],agt:[I]]]}, \tag{2.1}$$

where pat: and agt: are the role labels and bought is the verb that connects them.

In addition, the actions that result from the general parser are mapped to a clause that includes the arguments of the action and is labeled as an operator clause. An example that is created for the *agt* role, according to [9] is the following:

$$op([S1, S2|SRest], Inp, [SNew|SRest], Inp) := combine(S1, agt, S2, SNew), \tag{2.2}$$

where S1 is the action verb bought, Inp is the input, S2 is pat, SRest is the rest of the stack that is not involved in the action and SNew is the new stack created by the translation process.

This parser produces various analyses of the training sentences, which have to be specialized through the use of heuristic rules. This specialization is partitioned into three stages. In the first one, positive and negative examples for each operator clause are created. The positive ones include the first correct parse of each training example for each operator produced from the general parser. The negative ones include all the other parses. In the second stage, an induction algorithm creates the control rules for each operator. Each control rule comprises of a Horn clause based on the positive examples of the operator. Lastly, the control rules are combined with the output results of the general parsing, by unifying the head of the operator clause created from the general parsing and the head of the control rule for that clause. Then the new condition is added to the original clause body along with any inducted predicates.

The method was assessed through a tourist domain dataset and the results indicate a promising 98% accuracy on the test subset. Despite that the results are promising, the method either creates a correct output or fails, which might be a problem for producing semantic grammars with an optimal coverage. Moreover, this method requires a supervised learning approach, thus limiting the developer in cases that annotated datasets might not be available. Finally, this method focuses on semantic role labeling, thus it might not provide complete coverage of the patterns examined in this thesis.

2.4 Automated semantic role labeling

In [11], one of the first approaches for automated semantic role labeling is presented. This method is a statistical prediction model that infers predictions based on the probabilities of different syntactic features of a sentence. It must be noted that three of the syntactic features proposed in this method have influenced the method of this thesis, namely the governing category feature, the head word feature and the feature related to identifying phrase types.

The statistical analysis is initiated by parsing a sentence with a statistical constituency parser. The first feature of the method focuses on identifying the type of each phrase based on the Penn Treebank format. This features is inspired by the fact that different semantic roles are assigned different phrase types. Consequently, it could be inferred that utilizing phrase types could be useful when creating semantic grammars. A key feature related to this thesis is the notion of a governing category or gov feature. This feature tries to analyze the syntactic relations of the sentence by identifying the subject or object of verbs, which are represented as 'S' (sentence tag) and 'VP' (verb phrase tag). Another feature is the identification of the tree path from the constituent to be labeled up to the target word that invokes the semantic frame. The tree path is created by utilizing a bottom-up approach of analyzing a parse tree. The notion of a head word feature is also used in the method, where the statistical parser assigns a head word to the constituents of the parse tree, that is then used to predict the correct semantic role label. The last two features are the position and voice features. The position feature tries to compensate for incorrect parses and indicates the position of the constituent to be labeled with regard to the predicate of the semantic frame in question. Furthermore, the voice feature assigns a passive or active voice to the verbs of the sentence. These last two features are created specifically for semantic role labeling, thus were not deemed useful for the scope of this thesis.

Concluding, the method predicts the probability for a constituent to be labeled with a specific semantic role according to the distribution of a semantic role given all or a subset of the method's features. The paper explores different ways of combining the features, but mentions that for most combinations there is a trade-off between accuracy and coverage. Based on the principles of this approach, it is overall observed that there exists a strong correlation between syntactic and semantic analysis, as well as to which features could be regarded valuable to the generation of a semantic grammar.

3 Theory

The task of automated generation of a semantic grammar can be described as a natural language process (NLP) problem. The nature of such task requires systems that can analyze sentences from a training dataset or corpus, that could either be open-domain or domain specific. As argued previously, semantic grammars can be prone to problems when used for open-domain systems, but can be helpful for domain-specific ones. Therefore, domain-specific corpora should be used as the optimal type of dataset, when creating semantic grammars for a single task-oriented system.

Syntax analysis of a sentence in NLP systems can be done utilizing part of speech tagging and constituency parsing. Semantic analysis can be assisted by information extraction techniques (name-entity recognition, relation extraction).

3.1 Pre-processing

The first step in an NLP pipeline entails the pre-processing of the data. One pre-processing step is tokenization of a sentence, where the sentence is split into lexical units such as words or punctuation marks. This is an important step in NLP pipelines such as [14, 15]. Repeating lexical units, that are mainly produced from spoken language, should be removed in order to reduce errors [2]. Punctuation marks that are non-lexical units, such as commas, is argued to positively affect the parsing efficacy of a parser [16, 17]. Punctuation marks do not greatly improve the performance of a parser for simple sentences, whereas in longer more complex sentences it is shown to be of utmost importance [18]. Nevertheless, removing punctuation marks from the beginning or end of a sentence is recommended as it does not have an effect on the score of the parsed tree [19]. Lemmatization is another pre-processing step that will be included in the method of thesis and is described as the process where variations of a word, a verb or noun, are transformed into the basic form of that word.

For example, the word ‘born’ would translate to ‘bear’. The next step for syntactic analysis of an utterance includes the assignment of part of speech tags to each tokenized word.

3.2 Part of speech tagging

Part of speech (POS) tagging is the assignment of the most probable POS tag to a lexical unit for the current sentence’s context based on data from annotated corpora. These tags can vary between tagged corpora, but POS tagging models and parsers use specific category tags based on the part of speech and the grammatical form of the word. For example, according to the Penn Treebank POS tag system [13], the verb ‘eat’ would be assigned to the POS tag ‘VB’, which is an abbreviation for ‘verb base’, but the word ‘eaten’ would be assigned to the past tense tag ‘VBZ’. There exists a universal POS tagset defined in [20] that is broadly accepted and comprises 17 POS tag categories that are shown in Table 3.1.

Table 3.1: Universal POS tag table including the tag, the name and example words for every part of speech defined in [20].

Tag	**Part of speech**	**Examples**
ADJ	adjective	cheap, expensive, quick
ADP	preposition/postposition	in, for
PRON	pronoun	I, you, they
ADV	adverb	slowly, immediately
NOUN	noun	table, car, food
VERB	verb	eat, search, want
NUM	numeral	one, 2, IV
AUX	auxiliary verb	can, may, will
SCONJ	subordinating conjunction	if, unless, because
INTJ	interjection	hey, ouch, wow
CCONJ	coordinated conjunction	and, or
PUNCT	punctuation	‘.’, !, ‘,’
DET	determiner	a, this
SYM	symbol	+, =, $
PROPN	proper noun	John, Paris, Audi
PART	particle	to
X	unspecified POS	pfgh, jkml

Other corpora such as the Brown Corpus [21] and the Penn Treebank corpus [13] use more complex tags to encompass verb tenses and noun grammatical forms (singular, plural). For the purposes of this thesis, the universal POS tag system will be used. POS tagging is an important step that needs to be implemented before constituency parsing. This is due to the fact that constituency parsers rely on accurate POS tags to produce possible constituent combinations by mapping the sequential POS tagged lexical units to rules provided from the CFG of the system.

The task of tagging a word is ambiguous [22] and is the reason that probabilistic approaches have been introduced to solve the problem. A word could either be a noun or a verb depending on the context and its position in the sentence, thus it would be obscure to determine with 100% confidence the tag of the word in question without any other knowledge of the surrounding words. An example of how a word could be assigned two different POS tags based on the context of the sentence can be observed in Figure 3.1.

POS tagging approaches involve statistical models such as n-gram models and HMMs, and deep learning methods such as recurrent neural networks (RNN)s [22]. It has been observed that methods that include these approaches reach over 97% accuracy [23]. POS tagging can also be done using handwritten rules [24], an approach that was followed in the earliest attempts to tackle this NLP task, but can be assumed to be dated for this task.

PRON	VERB	PART	VERB	DET	NOUN
I	want	to	buy	a	book
PRON	VERB	PART	VERB	DET	NOUN
I	want	to	book	a	table

Figure 3.1: *An example of POS tag ambiguity. In this figure the word 'book' has different POS tags in each of the example sentences. Based on the context of each sentence it could be a verb or a noun.*

3.2.1 N-grams

N-gram models can be utilized to assign a POS tag to a word. An n-gram model examines a POS tagged corpus to find all the n-grams that contain the word to be tagged in the n spot of the sequence. Then the frequency of these n-grams is calculated and the n-gram with the highest frequency is used to predict the most probable POS tag for the word in question. For example, if a unigram tagger is used and the word 'rent' is more frequently found to be verb then the tag 'VERB' would be assigned to it. This would evidently not be effective for a tagger, thus it is recommended to use an n-gram tagger. A bigram tagger is an n-gram tagger that examines sequences of two words, thus considering only the previous adjacent word to the word that needs to be tagged. For example, a bigram tagger for the word 'rent' will find bigrams such as 'to rent' and 'the rent'. It has been observed that bigrams yield the best results in comparison to trigrams or n-grams, which are becoming more computational expensive, as the n increases, without providing any benefit [14]. N-grams could be also referred to as lexical based models, since they rely on assigning tags based solely on the frequencies observed in the tagged corpora.

3.2.2 Hidden markov models

An approach that incorporates bigrams, but is more efficient than a bigram tagger, is the use of HMMs. In the case of POS tagging, HMMs include two components, the probability of a tag to be observed after another specific tag and the probability that a given POS tag would be linked to a specific word. Additionally, HMMs incorporate a decoding feature, which in POS tagging is implemented by finding the most probable n-tag sequence for a series of n-words. HMMs utilize bigrams by assuming that the probability of a tag depends only on the previous tag. Another assumption made is that words are independent from the other words in the sequence. Nevertheless, HMMs are prone to errors when new words, pronouns and acronyms, are introduced [22].

3.2.3 Conditional random fields

Conditional random fields (CRF)s are statistical discriminative undirected graph models that are based on log-linear models [25]. In POS tagging, linear chain CRFs are used by assigning a probability to every tag sequence based on the word sequence to be tagged. These probabilities are called global features and are accompanied by weights. These global features are further split into a sum of local features. Every local feature probability is calculated based on the current tag, the previous tag, the word sequence and the position of the current tag. This model is similar to an HMM, with the difference that it can consider different features of the word sequence and it also resembles logistic regression on a word sequence scale [22].

3.2.4 Deep learning

Lastly, deep learning methods have experienced a tremendous interest during the last decades and especially in the NLP community through state-of-the-art research based on RNNs. POS tagging has also been affected by that research boom. State-of-the-art methods utilize bi-directional long term short term (Bi-LSTM) RNNs coupled with CRFs [26], with one of the methods with the highest accuracy combining a Bi-LTSM-CRF with contextual string embedding [27].

3.3 Constituency parsing

Constituency parsing can either be done using parse trees or RNNs [22]. In terms of constituency parsing, parse trees can also be defined as rooted ordered trees that concatenate a sentence into phrases or part-of-speech lexical units. These are represented as nodes in the parse tree. Each node that contains phrases that can be further concatenated is called a non-terminal node, whereas nodes with lexical units are called terminal nodes. A graphical representation of how this parsing method works is illustrated in Figure 3.2. The scope of phrase tags that will be considered in this thesis can be found in Table 3.2.

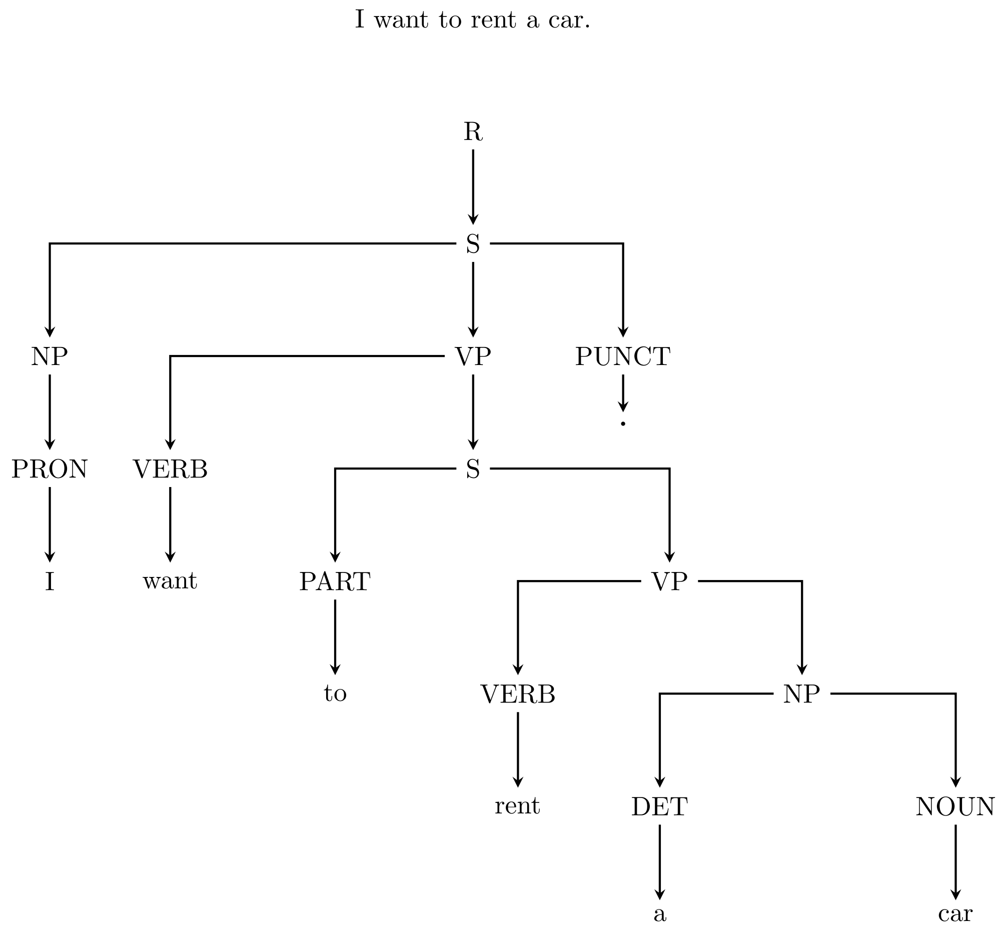


Figure 3.2: *An example of constituency parsing, produced by the Stanford CoreNLP [15] parser, for the sentence "I want to rent a car."*

Table 3.2: Phrase tags based on the Penn Treebank format.

Tag	**Description**
ADJP	Adjective Phrase
ADVP	Adverb Phrase
VP	Verb Phrase
NP	Noun Phrase
INTJ	Interjection phrase

Constituency parsing algorithms analyze sentences based on a grammar. The type of grammars that will be focused in this thesis are the CFGs.

CFGs are grammars that contain production rules of type :

$$A- > \alpha, \tag{3.1}$$

where A = a non-terminal symbol and α is a string of terminal and/or non-terminals.

A CFG comprises of a set $G = \{N, \Sigma, R, S\}$, where N is a finite set of non-terminal symbols (noun phrases, verb phrases), Σ is a set of terminal symbols (words), R is a finite set of production rules of the type in Equation 3.1 and S is the start or root symbol (sentence) in a parse tree.

It is expected that the larger the sentence is the more combinations of constituents can be produced, thus the more possible trees can be created. This creates another problem that could be solved by using a probabilistic context-free grammar (PCFG) and subsequently probabilistic parse trees and algorithms. A PCFG is defined as a CFG with rules that are assigned probabilities. The probabilities can be hard-coded based on the intuition of the developer or the linguist that is responsible to design the PCFG. A PCFG derived from the parse tree in Figure 3.2 is illustrated in Table 3.3.

Table 3.3: An example of a PCFG with rules derived from the parse tree in Figure 3.2 and probabilities assigned arbitrary. The table is divided into phrase rules on the first level and part of speech tag rules on the second.

Rule	**Probability**
S $\longrightarrow$ NP VP PUNCT	0.6
S $\longrightarrow$ PART VP	0.4
NP $\longrightarrow$ *PRON*	0.7
NP $\longrightarrow$ DET NOUN	0.3
VP $\longrightarrow$ VERB S	0.2
VP $\longrightarrow$ VERB NP	0.8
PRON $\longrightarrow$ *I*	0.5
VERB $\longrightarrow$ *want*	0.8
VERB $\longrightarrow$ *rent*	0.2
NOUN $\longrightarrow$ *car*	0.3
DET $\longrightarrow$ *a*	0.7
PART $\longrightarrow$ *to*	0.4
PUNCT $\longrightarrow$.	0.5

Another approach that is implemented by many parsers is the use of a treebank. A treebank is an annotated corpus with POS and constituency tags. An annotated corpus is represented by parse trees for every sentence in the corpus, from which the production rules for the grammar are created. With this approach, the probability of each rule is found by calculating the maximum likelihood of that rule, which is the amount of times the rule is found in the corpus, divided by the amount of times the left side of the production rule is found in a rule that is produced by the parse trees of the corpora.

One of the first and most influential approaches in constituency parsing is the Cocke–Younger–Kasami (CYK) algorithm [28]. The CYK algorithm is a chart bottom-up dynamic programming algorithm for constituency parsing that requires the CFG to be in a Chomsky normal form (CNF). A grammar is said to be in CNF, when all production rules consists of a non terminal symbol in the left side and a terminal symbol or two non-terminal symbols on the right side. The CYK algorithm is combined with a Viterbi approach to find the most probable parse tree when a PCFG is used [14, 22]. The probability of a parse tree is found by multiplying the probabilities of all the production rules it comprises.

There are a number of probabilistic parsers that do not rely on a CNF grammar and have produced results with high accuracy. The Charniak parser, in [29], described a maximum entropy probabilistic model for constituency parsing, with an updated version in [12] proposing a coarse-to-fine method for finding the n-best parse trees. This later version also uses a reranker, which introduces a process that finds the k-best parse trees and the parse tree with the best F-score is selected. An improved version of that parser implements a lexicalized PCFG that produces higher F-score. This method produces parse trees, where a head word is identified for every constituent and assigned as an extra label, along with its POS tag. This head word is subsequently assigned as a label for the sub-constituents derived from the initial constituent [30]. The process of finding the head word of a sentence, constituent or phrase is described as lexicalization.

3.4 Semantic analysis

Semantic analysis refers to the natural language understanding (NLU) task of interpreting the meaning or context of a text. This can be achieved by analyzing the structure of the text through syntactic analysis and information extraction techniques. The semantics of a domain can be expressed by a model consisting of an ontology of entities as described in Section 2.1 or logic forms as in Section 2.3. Domains that are represented by ontologies contain relations that connect the entities of the domain. Information extraction techniques such as named-entity recognition (NER) and relation extraction can be of great assistance to determine semantic aspects of a text.

NER is an information extraction technique that refers to the task of identifying named-entities such as locations and people in unannotated sentences. A NER can utilize statistical models such as CRFs and rule-based models to identify named-entities [31]. The statistical NER models are trained based on named-entity tagged corpora. The NERs that use rule-based models are flexible and can include either already established rules or custom ones developed for the needs of a specific domain [15]. Relation extraction is an information extraction technique that refers to the task of extracting relations between domain specific named-entities, entities that are created based on co-reference resolution, meaning entities that refer to the same entity in a complex sentence or entities that are part of a domain ontology. These relations are created based on a central verb that connects these two entities. Moreover, these relations are described using a metalanguage that is called Resource, Description Framework in the form of triplets. For example, in the sentence "I want a car." the relation resulting from that sentence would be the RDF triplet want(I,car). The entities in this triplet would be [*I*] and [*car*] in the *want* relation.

There are 5 recommended method categories to extract relations, namely using patterns, supervised machine learning, supervised using seed patterns, distant learning and unsupervised [22]. The approach of using patterns can be described as methods where hand-annotated rules of patterns are created and based on those rules the relative relations are extracted. One of the earliest approaches can be found in [32], where lexicon and syntactic patterns were defined based on hyponyms.

The approach of supervised machine learning involves the use of a trained classifier that decides whether two found entities are related. These classifiers usually involve RNN models, decision trees or regression models. For supervised methods with seed patterns, seed patterns or seed relation triples are used as a base of creating new relation patterns. Based on the seed relations, the method will try to find entities in the corpus or dataset and create new generalized patterns. Then, these new patterns are used to investigate new relations between other entities. Due to the fact that this method can result in erroneous patterns that would result in incorrect relations, a confidence value such as the one described in [33] can be defined in order to validate the relations created.

In terms of distant learning, an annotated relations corpus is used to create a large number of seed relations. The tuples of entities in the seed relations are used to find identical tuples in unannotated texts that would result in the creation of new relations. Then these new relations would be used as training data for a classifier that would find new patterns. This method is argued to combine the advantages of purely supervised machine learning and supervised with seed patterns techniques [22]. Finally, there are unsupervised methods such as in [34] that utilize syntactic and lexical constraints combined with POS and entity taggers. An important part of this method involves heuristic rules related to identification of main verb and noun phrases. This approach also utilizes confidence values to determine the validity of the relations created.

4 Method

The method of the thesis can be described as a hybrid approach of statistical models and a rule-based model. It comprises syntactic and semantic analysis of dialogues suited for task-oriented conversational systems. The method aims to analyze each user utterance in a set of dialogues contained in a dataset and create *word* and *phrase categories* based on that analysis. This chapter gives a summary of the steps of the method, and then establishes the basic principles behind each step. Furthermore, the datasets used are described in depth and an evaluation process is defined. The method analyzes each dataset by splitting each dialogue in turns that correspond to a user utterance and an agent response. In cases where a user utterance is not present for that turn, for example if an agent responds two times to a user's utterance, then the last valid user utterance is considered as part of the current turn. During each turn only the user's utterance is analyzed. The method can only be applied to datasets containing dialogues with turns of user input that correspond to one sentence.

The first step for the analysis is the pre-processing step that must be taken for every user utterance and is described by the scheme in Figure 4.4. The pre-processing involves augmentation of an utterance by trimming punctuation marks that do not affect the syntactic analysis and capitalization of words in caseless datasets. Caseless datasets are datasets that do not have proper capitalization for lexical units like proper nouns. True casing a sentence means that the systems will try to capitalize words that are named-entities or need capitalization for some other reason. The capitalization is accomplished by identifying named-entities through the usage of a NER annotator trained for caseless sentences. NER annotators can be trained to identify named-entities in caseless sentences, but they are not always accurate. The named-entities found are also used in the semantic category creation phase. The next step involves the syntactic analysis of each utterance, which is described by the scheme in Figure 4.5. The user's utterance is processed by identifying the POS tags of each word in the sentence by a statistical tagger and then parsing it by means of a statistical constituency parser. This results in the creation of a parse tree that will then be used to create *word* and *phrase categories.* The use of POS tags will not only be required by the constituency parser but also by the semantic analysis. In order to be able to conduct the semantic analysis there is a need to translate the parse tree into a class object. After the translation of the parse tree into a class object, the method proceeds with the semantic analysis.

The core function of this analysis is to utilize linguistics to create patterns that would then be assigned semantic labels. Through each parsed utterance the method generates *word* and *phrase categories.* This part of the method is implemented with the use of an "if-then" rule-based system according to rules that have been established by the author through common syntax norms and observations. The system that is proposed aims to generate noun *phrase categories* with rules that focus on identifying nouns that would be then used as head words for the *phrase categories* created. Nevertheless, this method can also create non-noun *phrase categories* under specific circumstances that will be described in Subsection 4.4.2.

In addition to the initial creation of the *categories*, the method also aims to find *phrase categories* that, depending on the context or domain of the dataset in question, are similar. In order to accomplish that, information extraction techniques are employed to find *phrase categories* that might be semantically common, without necessarily having lexically and syntactic synonymous patterns. The core thought behind this feature is the fact that semantic synonymity does not imply syntactic synonymity. The task of identifying semantically synonymous *phrase categories* is accomplished by defining a new concept of a relation entity and combine it with unsupervised relation extraction. The method creates a list of relations and secondarily identifies similarities between the relations of the list in order to find semantically synonymous *phrase categories.* A scheme of the overall pipeline can be observed in Figure 4.1.

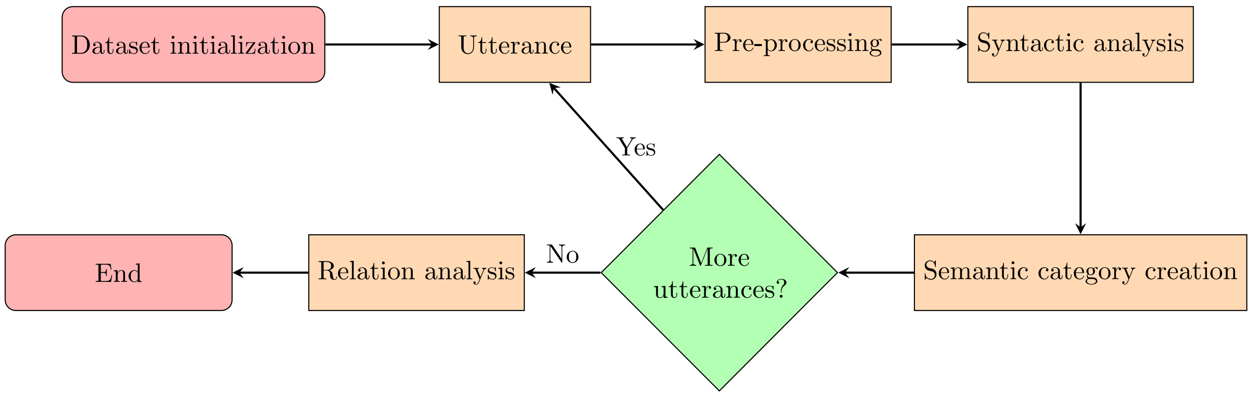


Figure 4.1: *The overall scheme of the method. The dataset is first initialized in a way that it can be processed. Each user utterance is analyzed by sequential pre-processing, syntactic analysis and semantic category creation. If there are not any utterances left, the last step is to analyze the relations created from the analyzed user utterances.*

For the method implementation, the *Stanford CoreNLP* pipeline [15] is utilized. This package comprises various NLP tools, such as a tokenizer, a lemmatizer, a statistical POS tagger, a statistical constituency parser and a NER annotator. The package is coded in *Java* but it can be accessed through *C#*, *Python* and various implementations in other programming languages. For the method, the *C#* implementation [35] of the pipeline will be used. This implementation is compatible with Microsoft's *.NET* framework, as it translates the *Java* code into a *C#* usable version of the pipeline with the help of *IKVM.NET*. *IKVM.NET* is a *Java* implementation for Microsoft's *.NET* framework, which includes a *Java Virtual Machine* created in *.NET*, a *.NET* implementation of *Java* libraries and tools that assist in interchangeability of project code written in *Java* and under *.NET* [36]. The reason that it was decided to work with *.NET* is that it will enable the combination of the project with DAISY, that is also implemented in *.NET*. Moreover, *.NET* enables access to various libraries that could be useful for web and windows application development, as well as a robust way of creating an application programming interface (API), in case it would be needed to incorporate the method in other projects. Lastly, *Stanford CoreNLP* provides the opportunity to use pre-trained models for all of the tools included in the library. These models can be used for cased and caseless datasets.

4.1 Dataset

The datasets that can be used with the method must adhere to a specific format. The method is created for unannotated dataset suited for dialogue-based systems. This means that the dataset must contain dialogues with turns and each dialogue must be separated from the previous and next dialogue by a new line. Each line of the dialogue must include a turn that comprises one user utterance and one agent response separated by a horizontal tabulation. Moreover, the method is programmed to read a dataset through a '.txt' type of file and store it to a one dimensional string array that contains one turn of a dialogue at each cell. The method is not constrained to work only with '.txt' type of files, thus if a developer wishes to use a dataset that is stored in a different type of file, they must implement their own way of reading that dataset and store it in the one dimensional string array that is used by the method.

The data used for this thesis were the first task subset of the bAbI dataset and the rental dataset that was manually created by the author. The subset of the bAbI dataset is a caseless dataset with dialogues for restaurant reservations and is artificially generated to simulate spoken language interaction between a user and an agent (or so-called bot). This dataset contains interactions with differences in the structure of the patterns but with a small set of words. The dialogues contain interactions where the user would express most of the information needed for the restaurant reservation during the first turn of the dialogue, thus creating long and complicated sentences, such as in Figure 4.2. Furthermore, the subset of the dataset that is used to develop the method included two types of non-analyzable utterances that can also be observed in Figure 4.2. The user responses that contained the tag '<SILENCE>' define that the agent responded two times in a row and the agent responses that contain the tag 'api_call' to define that the agent conducted an API call instead of responding to the user. The turns that contain these two tags are not analyzed nor used by the method.

1 User: hello   Agent: hello what can i help you with today

2 User: can you book a table with french cuisine for eight people in a cheap price range   Agent: i'm on it

3 User: <SILENCE>   Agent: where should it be

4 User: in madrid   Agent: ok let me look into some options for you

5 User: <SILENCE>   Agent: api_call french madrid eight cheap

Figure 4.2: *An example dialogue from the bAbI dataset. Each line defines a turn and contains a prefix index following a user utterance and an agent response. The '<SILENCE>' tag defines the absence of a user utterance for that turn and the 'api_call' tag indicates that the agent response is part of an API call process. The user utterance in line 2 contains a complicated request with a lot of information resulting in the agent inquiring only about the place of the booking.*

The rental dataset is a cased dataset with dialogues for renting a vehicle. This dataset contains simple interactions but with a more diverse vocabulary, such as in Figure 4.3. The choice of introducing a more diverse vocabulary was made so that the method would be able to produce more *word categories* and further evaluate that part of the method. Lastly, the use of a second dataset was also introduced for evaluating the overall performance of the method in different domains, thus to understand to what extent the method could be used with the same performance regardless of the domain.

1 User: Hey   Agent: Hey

2 User: I want to rent a car   Agent: What type of car?

3 User: A cheap car   Agent: What colour would you prefer?

4 User: red   Agent: For how many days?

5 User: For four days   Agent: Ok, I found a red Toyota Yaris for 30 euros per day

6 User: <SILENCE>   Agent: Would that be ok?

7 User: Yes   Agent: Ok, your booking is ready

Figure 4.3: *An example dialogue from the rental dataset. The structure is the same as in Figure 4.2, with the difference that an 'api_call' tag is not used. In this dialogue, the user's rental request is a simple sentence resulting in the agent asking far more questions to the user in order to acquire all the necessary information.*

4.2 Sentence augmentation

As was previously discussed, there are some pre-processing steps that are needed before running a sentence through the constituency parser. A sentence can be augmented so that the syntactic analysis is optimized and errors that are created by statistical models or any similar supervised approach can be minimized. Since the method also relies on correct POS tags and constituency parsing for creating *categories*, it could be inferred that sentence augmentation is also detrimental for the overall performance of the method.

Pre-processing for the method firstly involves trimming periods and commas from the utterances. Since the utterances are considered to be distinct sentences, periods are trimmed from the end of the utterances if present. If a semantic grammar is used for systems that communicate only by verbal means, then creating *phrase categories* with commas or periods would hinder the ability of the system to recognize the user's utterance in cases that the speech recognizer is not able to create and correctly allocate those marks. This could also be the case even in text communicative systems where the semantic grammar may not adhere to the sometimes vague positioning of commas by the users. Other punctuation marks are not included in the scope of this thesis, thus they are not trimmed during this step.

Furthermore, the sentence augmentation process for the method includes true casing caseless datasets. In caseless datasets, the first problem that must be handled is the ambiguity of words' POS tags. A caseless dataset could cause an increase of erroneously tagged words, that would otherwise be avoided. In the case of the word 'spanish', the capitalization of the word is important to differentiate between an erroneous and a correct POS tag. In a cased dataset the word 'spanish' will be assigned an adjective POS tag, but the word 'Spanish' would be assigned a proper noun POS tag, whereas in caseless datasets this will also depend on the surrounding words and the training data of the tagger model. In other cases, where proper nouns must be recognized, the POS tagger that is being used might not have been trained to recognize the word when not capitalized, resulting in a proper noun tagged as noun.

The method tries to solve the true casing problem by utilizing a NER model. The method uses the caseless model and the NER annotator package provided by the *Stanford CoreNLP* pipeline. Due to the fact that for this step it is not required to use a sophisticated NER annotator, a three class caseless pre-trained NER model combined with the statistical part of the NER annotator is used. The statistical part of the annotator consists of a series of trained CRFs based on the method described in [31] and the pre-trained model of the classes 'PERSON', 'LOCATION', 'ORGANIZATION'. The utterance is parsed through the NER annotator, where the named-entities are identified and stored in a named-entity list. In caseless utterances, the list is used to capitalize all the named-entities found in the utterance. Moreover, this list is used in the semantic analysis for *phrase* and *word category* generation. A rough scheme of the pre-processing part of the method can be observed in Figure 4.4.

In order to parse the sentence through the POS tag annotator, the utterance needs to be tokenized. This is

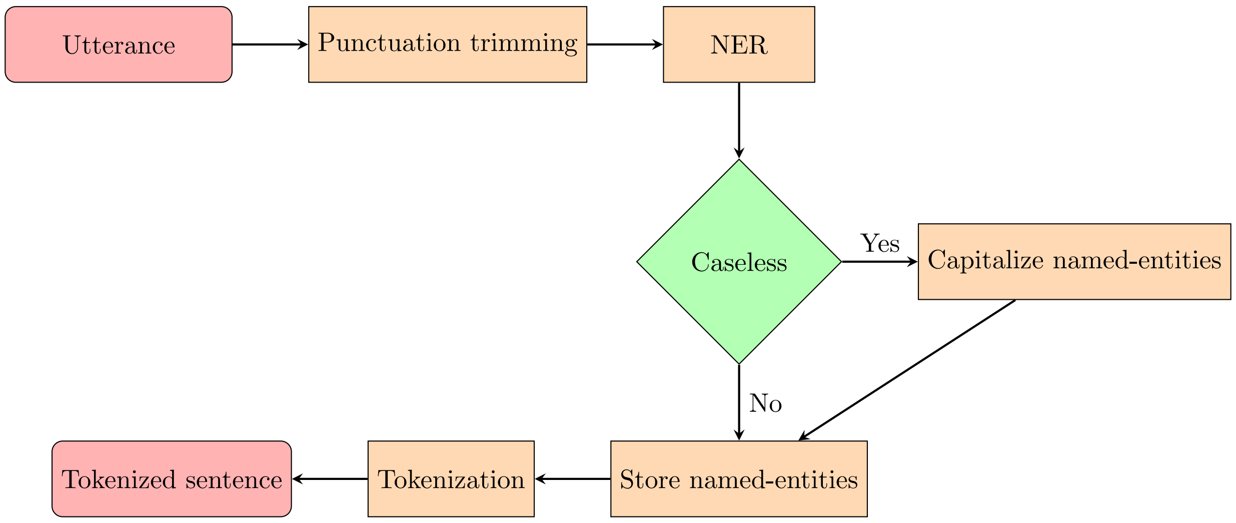


Figure 4.4: *A rough scheme of the pre-processing phase.*

achieved by using the tokenize annotator provided in the *Stanford CoreNLP* package. This tokenizer follows the Penn Treebank standard of tokenizing words. This involves splitting the utterance into a list of lexical units. For example, the sentence "I want to rent a car" would be split into the list ($'I'$, $'want'$, $'to'$, $'rent'$, $'a'$, $'car'$). Even though tokenizing a sentence might seem like a self-explanatory task, there are special cases that need to be considered. There are various options in regard to handling special cases during tokenization when using the *Stanford CoreNLP* tokenizer. The options that were deemed as most related to the scope of the thesis are described in Table 4.1. For the purposes of this thesis, the default values for the tokenizer are chosen. This means that in cases of hyphened words such as 'semi-supervised', the word would be split in three tokens namely ($'semi'$, $'-'$, $'supervised'$) and in cases where two words are connected with a slash character such as "Swedish/Italian" the words would be tokenized as ($'Swedish'$, $'/'$, $'Italian'$). In addition, cases with assimilation such as the word 'wanna', are also tokenized as ($'wan'$, $'na'$).

Table 4.1: Options provided for the tokenizer in the *Stanford CoreNLP* toolkit, which were deemed notable.

Variables	**Type**	**Description**
splitAssimilations	bool	If true, tokenize assimilations.
splitHyphenated	bool	If true, tokenize hyphenated words as several tokens.
splitForwardSlash	bool	If true, tokenize segments of slashed tokens separately

4.3 Syntactic analysis

The next step in the method involves the syntactic analysis steps, which are described by the scheme in Figure 4.5. The maximum entropy POS tagger of the *Stanford CoreNLP* library, that is described in [37, 38] will be utilized to assign POS tags to the tokenized sentence. The model that is used to annotate the lexical units of an utterance is a pre-trained model suited for cased datasets. Furthermore, the model is created using the Wall Street Journal (WSJ) sections 0-18 of the Penn Treebank corpus as a training dataset. This model is recommended for more general cases of datasets, thus aligning better with the aim of the method, which is to aspire for universal coverage of datasets.

The model follows the Penn Treebank POS tag format, whereas the method uses the universal POS tag system that is introduced in Section 3.2. The CoreNLP package does not provide a way of directly translating POS tagged words from the Penn Treebank format to the universal POS tag system, but this is possible through the translation of a parse tree. Therefore the translation to universal POS tags is achieved after the sentence has been parsed through the constituency parser.

The following step would be to parse the tokenized and POS tagged list of lexical units with a constituency parser. The constituency parser is an implementation of the CYK parser, described in Section 3.3, provided in the *Stanford CoreNLP* toolkit. The parser utilizes the english unlexicalized PCFG model provided in the *Stanford CoreNLP* toolkit. This provides a fast and accurate way of parsing a sentence without the need of lexicalization. The PCFG is trained on the sections 2–21 of the WSJ section of the Penn Treebank based on an unlexicalized approach described in [39]. The decision was made due to the simplicity and speed of this approach. Despite that the method utilizes an unlexicalized PCFG model, it is argued that lexicalization is useful in semantic analysis [39], thus the method implements its own approach of lexicalization during the semantic category creation phase that will be explained in Section 4.4.2.

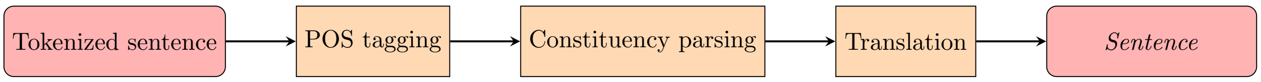
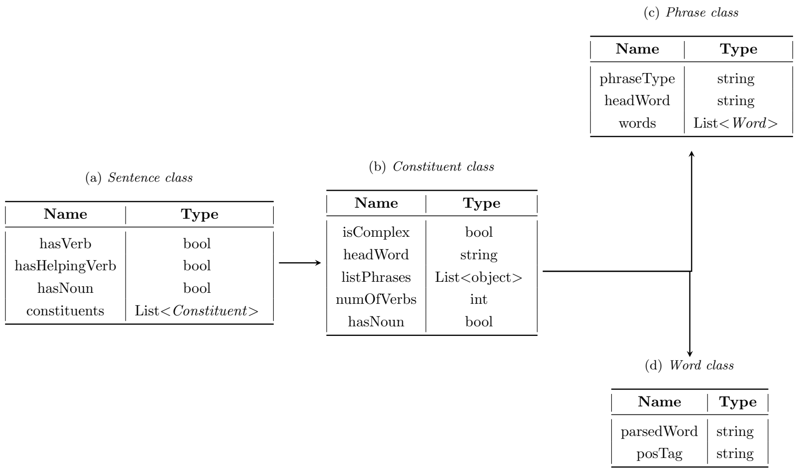


Figure 4.5: *A rough scheme of the syntactic analysis phase including the translation step. The tokenized sentence is tagged by the POS tagger. Then the constituency parser produces the sentence's parse tree and the translation step transforms the parse tree to a Sentence class object.*

After the tokenized tagged list of lexical units has been parsed, a parse tree is created enclosing all the syntactic information of the sentence represented by an object of a class named *Tree* that is implemented in the *Stanford CoreNLP* toolkit. Although the class *Tree* provides a lot of useful functions such as finding the subtrees of a parse tree or the depth of a tree, it is not flexible in regard to the data manipulation that was needed in the semantic category creation phase. A class called *Sentence* is created for each user utterance, that can provide a more flexible data structure of the annotated constituents and phrases. The *Sentence* class comprises variables that can provide access to syntactic information about the sentence being processed, such as whether the sentence contains nouns (the 'hasNoun' variable), verb (the 'hasVerb' variable) or if it has more that one verb (the 'hasHelpingVerb' variable), namely an assisting verb. The variables of the class *Sentence* can be found in the Figure 4.6. Furthermore, the *Sentence* object includes a list of class objects named *Constituent*, that contain the variables found in Figure 4.6. The *Constituent* class contains variables that show if the constituent is complex (the 'isComplex' variable), which is used to determine if the *phrase category* created for this constituent is complex or not. Other variables are the 'headWord' variable that stores a temporary head word that is used in the translation step, the 'numOfVerbs' variable that stores the number of verbs in the constituent and the 'hasNoun' variable that defines if the constituent has a noun. Additionally, it can be observed that the *Constituent* class contains a list of objects. These objects can either be instances of a class named *Phrase* or a class named *Word*. The *Phrase* class contains a variable named 'phraseType', which defines the type of the phrase such as noun ('NP') or verb ('VP') phrase, a 'headWord' that has the same functionality as in the *Constituent* class and a list of class objects named *Word*. The *Word* class contains a 'parsedWord' variable that defines the word for that object and a 'posTag' variable that contains the POS tag of the word. In that way, the resulted *Tree* object of the constituency parser is translated into a class object that encloses all the syntactic information such as constituents, phrases and words in a simple scheme of inheritance found in Figure 4.6, where each class object contains a list of a one-level simpler subpart of the object.

In Figure 4.6, it can be observed that the *Constituent* class's list of phrases is of type 'object', which means it can either be a *Phrase* or a *Word* object. The reason for a distinction between words that are grouped into a list and individual words is that the semantic analysis relies on noun phrases. During the translation process, all noun phrases are grouped into a list of *Word* objects. A noun phrase at this step of the process is described as a list of preceding words including determinants, adjectives and descriptive words of the head noun. Any other words that are not part of this group of words, but are closely related to the head noun, are considered as preceding or exceeding words. These words are included as individual *Word* objects in the *Constituent* list of '*Phrase*' (the quotes are used, because evidently the list does not strictly contain *Phrase* objects but also *Word* objects). In order to better understand the structure of the *Sentence* class and its subclasses the sentence "I want to rent a car" will be examined. The parse tree that corresponds to that sentence is described in Figure 3.2. This representation does not comply with the method's approach and the *Sentence* object is augmented for such cases.

The main part of these changes include the words that exist before the main verb of a sentence. Every word before the main verb of a sentence is grouped under one constituent. A main verb can be described as the verb that is involved in the relation between the subject and the object of the sentence. Task-oriented systems can be expected to handle simple phrases that only contain nouns, adjectives or interjections. For example, a user



(a) *Sentence class*

Name	Type
hasVerb	bool
hasHelpingVerb	bool
hasNoun	bool
constituents	List<*Constituent*>

(b) *Constituent class*

Name	Type
isComplex	bool
headWord	string
listPhrases	List<object>
numOfVerbs	int
hasNoun	bool

(c) *Phrase class*

Name	Type
phraseType	string
headWord	string
words	List< *Word*>

(d) *Word class*

Name	Type
parsedWord	string
posTag	string

Figure 4.6: *An inheritance scheme for the Sentence class that is produced after the translation of the parse tree.*

could answer to the question "How many days?" of the agent with a simple 'four' or "four days". The most complicated sentences that are expected to be handled by a dialogue-based task-oriented system would include a main verb accompanied by another verb that has an assisting role, such as an auxiliary verb or a verb of type 'want' that is followed by the particle 'to' before the main verb. Furthermore, they would include the notion of a subject, usually the user, represented by a pronoun of type 'I' or 'We' and an object that is related to the purpose of the task-oriented system. For example, if the system is a vehicle renting system then the object of the sentence would be a noun of type vehicle, such as a car. In the case of the sentence in Figure 3.2, the main verb is the verb 'rent', the assisting verb is the verb 'want', the subject is the pronoun 'I' and the object is the noun 'car'.

The translation process results in the creation of a *Sentence* object that includes two constituents. The first constituent includes the words $('I', 'want', 'to')$, which are before the main verb 'rent', in the form of *Word* objects. The list of subparts of the second constituent contains one *Word* object that represents the verb 'rent' and one *Phrase* object that contains the rest of the constituent. This *Phrase* object contains an ordered list of the words $('a', 'car')$ in the form of *Word* objects. The translation process for the sentence "I want to rent a car" can be observed in Figure 4.7.

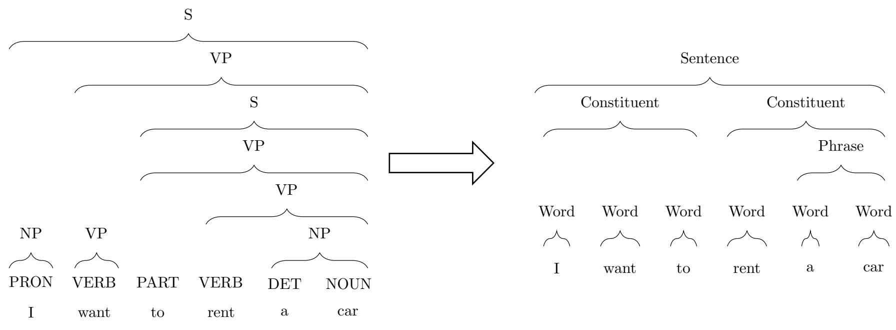


Figure 4.7: *An example of the translation process from a parse tree to a Sentence object. The figure depicts on the left side of the arrow the rough scheme of the parse tree created after the constituency parsing of the sentence "I want to rent a car" and on the right side the translated version of the parse tree.*

The sentence "I want to rent a cheap car for four days" is described by the same initial structure as before, with the difference that here the second constituent now comprises two *Phrase* objects and one *Word* object. More specifically, the two *Phrase* objects correspond to the lists $('a', 'cheap', 'car')$ and $('four', 'days')$, where the words 'car' and 'days' are considered to be the head nouns. The word 'for' is stored as an individual *Word* object between the two *Phrase* objects. The numeral 'four' is also considered to be closely related to the head noun 'days', thus it is included in the *Phrase* object of the noun 'days'. The *Sentence* object created for the sentence "I want to rent a cheap car for four days" is depicted in Figure 4.8.

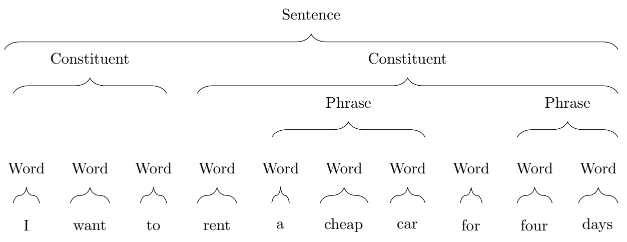


Figure 4.8: *The resulted Sentence object for the sentence "I want to rent a sports car for four days". The structure is similar with the one in Figure 4.7 with the difference that here the second constituent contains two phrases. The word 'for' is not part of the second Phrase object.*

4.4 Semantic category creation

Following the syntactic analysis of the sentence, the resulted *Sentence* object is analyzed by the rule based model. The system creates two different types of *categories*, namely *word categories* and *phrase categories*, that are stored in two lists. While the method parses the sentence, a relation is also created for that sentence and is stored in a relation list, that will be used for the relation analysis phase of the method. Every relation is described by a class object named *Relation* and consists of a left side, a verb and a right side. These terms are described by the variables in Figure 4.9 in the *Relation* class. The relation extraction and analysis will be discussed further in Subsection 4.5. The process for the semantic category creation is described by the following steps:

- Initialize a *Relation* object for the sentence.
- Parse every word of the sentence one by one.
- For every word parsed, create a *word category* and add it to a temporary *word category* list.
- If main verb found, assign the verb to the verb variable of the initialized *Relation* object.
- Store the temporary *word category* pattern as a *phrase category* and add it to the *phrase category* list.
- Assign the *phrase category*'s label to the left side of the initialized relation.
- Continue parsing the sentence.
- Create *phrase categories* for the remaining part of the sentence and add them to the *phrase category* list.
- Create one *phrase category* enclosing all the *phrase categories* found in the previous step.
- When the parsing has ended, add the resulted *phrase category*'s label from the previous step to the right side of the *Relation* object.

The result of the semantic category creation phase for the sentence "I want to rent a car" can be observed in Figure 4.13a. The method creates two *phrase categories* and six *word categories.* A different case with generic labels and a complex sentence is the sentence "May I rent a car for four days?". In Figure 4.13b, instead of the verb *phrase category* '[VP_want]' of the example in Figure 4.13a, the 'subject' phrase of the sentence has the generic label '[AUX_PRON]'. Additionally, the noun *phrase category* '[NP_car]' is complemented by one more simple *phrase category* '[NP_day]' and a complex noun *phrase category* '[COMPLEX_NP_car]'.

As it was previously noted, every *phrase* and *word category* comprises a label and a list of words or *word category* patterns. The structure of the classes representing the two *categories* can be found in Figure 4.9. The label creation for a *category* is important as it can provide helpful semantic information regarding the *category* and thus assist the developer to optimally use the semantic grammar. When a *category* is used in a conversational system the right use could become cumbersome if the label is not self-explanatory. It is not optimal to create *categories* with labels where the developer has to continuously validate the meaning of them or use them in a wrong way, thus hindering the process of developing a conversational system. In order to avoid that, the goal is to create *categories* with self-explanatory labels. The details regarding the creation of these *categories* will be elaborated in the following subsections.

Name	Type
label	string
singleWords	List <string>

(a) *Word Category*

Name	Type
label	string
singleWords	List<List<string>>

(b) *Phrase Category*

Name	Type
leftSide	string
verb	string
rightSide	string

(c) *Relation*

Figure 4.9: *The structure of the classes representing the word category class, the phrase category class and the relation class. Each row contains the name and the type of the variable. The phrase category on the right contains a variable 'patterns' that is only populated by strings of type word category 'label'.*

4.4.1 Word category

A *word category* is described as a *category* that contains a label and a set of words. The label of *word category* represents the semantic meaning of that *category.* This set of words includes semantically synonymous words or words that are in different forms of the same word. For the creation of each *category* a specific process is followed, although it differs depending on the part of speech of each word examined. The semantic synonymity of a *word category*'s set depends on the domain of the dataset and the sentence context they were used in. It is also necessary to group together words that are different forms of the same word. For example, the plural and noun form of a noun or the past and present tense form of a verb should be grouped in the same *word category.* The *word category* creation is divided into label creation and word list generation. Lastly, punctuation marks that are not deleted by the sentence augmentation pre-processing step, such as hyphens or slashes, are not handled by the method during the *word category* creation step.

The POS tag of the word is considered first for the creation of the label. If a word is a noun, verb or an adverb then the label that is assigned to the *word category* is the lemmatized version of the word. For example, if the word is 'days', then the *word category* label will be '[day]'. For adjectives the label assigned to the *category* is created by finding the noun that is subject to the adjective. Then the label created would be assigned the prefix 'TYPE_' followed by the noun that was found to be described by that adjective. Therefore, in the phrase "cheap restaurant" the *word category* created for 'cheap' would be '[TYPE_restaurant]'. In cases of nouns that are used as adjectives, such as in the phrase "family car", the label is created with the same way, thus resulting in the label '[TYPE_car]' for the word 'family'.

User: I want to rent a car

Agent: What type of car?

User: convertible

Figure 4.10: *A dialogue where the user's response contains an adjective word that describes the noun 'car' and the previous agent responses contain the keyword 'type' and the noun 'car'.*

In the case that there is not a noun present in the sentence and that sentence contains only one word, the method considers the previous agent response to find a possible noun. This is derived by the fact that in many situations like the dialogue in Figure 4.10, the user does not need to use the noun word, as the agent has provided the necessary information and established the context for that noun. Therefore, the agent asks the user what type of car would they want to rent and the user responds by providing the adjective 'convertible'. In the same scenario, if the agent was replaced by a human, then the human would understand that the response of the user is the adjective for the noun 'car'. In Figure 4.10, the sentence includes the keyword 'type', which from a human's perspective designates that the response of the user would be an adjective for the noun 'car'. In similar scenarios, a common way of formulating the question in 4.10 would be to use a keyword followed by the noun. For example, the question could be formulated as "What kind of car?". This also inspired the creation of a set of keywords, namely 'preference', 'type', 'kind', 'sort'. Apart from the word 'preference' the other words were selected as synonyms of the word 'type'. These words are stored in a list during the initialization phase of the method. The steps followed to identify possible nouns in this cases are :

- Find keywords included in the predefined set of 'type-like' words.

- Search for a noun following that keyword.
- If found, create a *word category* with label '[TYPE_noun]', where noun is the noun found.

In cases where either a keyword is found and a noun cannot be found or a keyword cannot be found, the method examines the previous user utterance. If the utterance is a sentence that includes a verb and a main object (head noun) then that noun is used to create the *word category* of that adjective. An example of that case can be observed in Figure 4.11, where the head noun of the initial user utterance is the word 'car', resulting in the creation of the *word category* '[TYPE_car]'. The agent's response does not include a noun following the keyword 'type', but the method used the initial user utterance to find the noun appropriate for the adjective 'convertible'.

User: I want to rent a car

Agent: What type?

User: convertible

Figure 4.11: *A dialogue, where the user's response contains an adjective word that describes the noun 'car' and the previous agent utterance does not include the noun.*

These rules also apply to cases of nouns or proper nouns that could be used to describe a noun. For example, in Figure 4.12, the proper noun 'BMW' that corresponds to the name of a car company will be assigned to the *word category* with label '[TYPE_brand]'. If a noun is still not found and the word is a descriptive noun or adjective, then the label for the *word category* would be the lemmatized version of that word.

User: I want to rent a cheap sports car

Agent: What type of brand?

User: BMW

Figure 4.12: *A dialogue where the user's response contains only a proper noun that is descriptive of the noun 'brand'.*

For proper nouns, except from the case described above, the stored named-entity list from the pre-processing step is examined. If the proper noun is found in the named-entity list, then the label of the *word category* for that proper noun would be the type of the named-entity. For example, the city 'Paris', if found by the statistical NER annotator, would be assigned to a *word category* with label '[LOCATION]'. Additionally, in a case where a proper noun is not found by the NER annotator nor is found to be describing a noun, it is assigned to a *word category* with label 'PROPN' (the POS tag for proper nouns).

For all the other words, their POS tags are used as the label for the *word category* they will be assigned to. This means that the *word categories* for these words would have labels equal to their POS tag, similar to the case of unidentified proper nouns. For example, a word with POS tag 'INTJ' will be assigned to the *word category* with label '[INTJ]'.

A *word category* contains a list of words that were found to be semantically related enough as to be included in the same *category* . The way that these lists are generated resembles the way the labels were created. For nouns, verbs and adverbs the method's approach is to include the words that are different forms of the same word. The reason for not including synonymous words is the problem of word sense ambiguity, that could result in words that are not semantically synonymous being grouped in the same *word category.* For example, a dictionary-like corpus such as WordNet [40] contains sets of synonymous words. If WordNet was utilized by the method, the words 'tabular array' and 'table' would be grouped in the same *category.*

As introduced in the label creation part, the adjectives are grouped under the same *word category* of the noun that they were found to describe. The same applies for proper nouns and nouns that are used to describe nouns. Therefore, if the words 'cheap', 'family' and 'BMW' were used to describe the noun 'car', then they would be grouped into the same *word category* with label '[TYPE_car]'. The verbs and adverbs are grouped the same way as the nouns and all the other types of words are grouped under the same *word category* according to their POS tag. Therefore, the auxiliary verbs 'can', 'would', 'may' will be grouped under the *word category* '[AUX]' or interjections such as 'hey', 'yes', 'greetings' would be grouped under the *word category* '[INTJ]'.

4.4.2 Phrase category

The main goal of the method is to create noun *phrase categories*, where a noun is present. The *phrase categories* consist of a label and a list of *word category* patterns. These patterns consist of *word categories* that are created by the lexical units found in the sentences. In the case of a *phrase category*, a different approach is incorporated. A pattern is assigned to a certain *phrase category* based on the head noun of the phrase. Furthermore, if a *phrase category* is complicated, it is additionally needed to identify if the pattern consists of other simple patterns.

The label creation process for *phrase categories* is different than the one in *word categories*. The type of labels created by the method are divided into verb, noun and generic pattern labels. The noun *categories* which are the key aspect of the method can have two types of labels depending on whether they are simple noun *phrase categories* or complex. Despite that, both are created based on the lexicalization of the phrase. The lexicalization of a noun phrase is accomplished by analyzing the *Phrase* object, for that phrase, and finding the last noun in its list of *Word* objects. Simple *phrase categories* have a label with the prefix 'NP_' following the noun of that phrase and complex *phrase categories* have a label with prefix 'COMPLEX_' following the head noun of the phrase. The head noun of a complex phrase is the object of the sentence that through the method is defined as the head noun of the first noun phrase that is included in the complex phrase pattern list. For example, the phrase "a cheap car" will result in the creation of a simple noun phrase with label '[NP_car]', whereas the phrase "an expensive car for four days" will result in two simple noun *phrase categories* with labels '[NP_car]', '[NP_day]' and one complex noun phrase with label '[COMPLEX_NP_car]'. The head noun for this complex phrase is the noun 'car'. This choice is not arbitrary and it is derived from that fact that in dialogue-based task-oriented system the nature of the sentences are of type 'subject-verb-object'. Therefore in a sentence such as "I want to book a table for two people", the subject is the pronoun 'I' and the object is the noun 'table', which is the immediate noun after the verb 'book'. By generalizing this example, the same assumption can be made for phrases that contain the subject and object of a sentence. Therefore, it is established that these sentences contain 'subject' and 'object' phrases. The 'subject' phrase of the sentence would be the phrase "I want to", the verb would be the verb 'book' and the object phrase of the sentence would be the phrase "a table for two people". Following the logic of the previous example, the object phrase "a table for two people" would be mapped to a complex noun *phrase category* with label '[COMPLEX_NP_table]', as the word 'table' is the object of the sentence.

Another type of *phrase categories* that the method creates are verb *phrase categories*. A verb *phrase category* is created from phrases that contain an assisting verb in a sentence and usually belong to the left side of the main verb of the sentence. These *phrase categories* have labels with prefix 'VP_' followed by the assisting verb. For example, the phrase "I want to" would be mapped to a verb *phrase category* with label '[VP_want]'. Considering these examples, a verb phrase contains a set of words located near the subject of the sentence. Therefore, these verb phrases are considered to be the 'subject' phrases of a sentence.

In the case that neither a noun nor an assisting verb is present in a phrase, two subcases are considered. The first subcase considers patterns of words that would be regarded as incomplete user utterances, such as "in Paris", or 'subject' phrases. The second subcase considers phrases that contain descriptive words that could be assigned a noun phrase label. For the first subcase, the method creates a label equal to the jointed labels of the created pattern for that phrase. For the sentence "May I rent a car", the 'subject' phrase of the sentence is the phrase "May I". The method will create a *phrase category* with label '[AUX_PRON]', because the *word category* pattern for this phrase contains two *word categories* with labels '[AUX]' for the auxiliary verb 'May' and '[PRON]' for the pronoun 'I'. Therefore, the method does not treat auxiliary verbs as assisting verbs.

For the second subcase, there might be a descriptive word that hints the presence of a noun. Following the same logic as with the label creation for descriptive words, the appropriate noun is found and then the phrase is assigned a label with prefix 'NP_' following the noun word. For example, in the Figure 4.10, the user's response to the agent's question is an individual word that could only be assigned to a *word category*. The question that arises in this situation is whether a *phrase category* could be derived from that single word response. In cases of numerals, adjectives and descriptive proper nouns this could help the semantic grammar to be able to comprehend more vague responses of users. The method tries to find a noun that could be used to combine the user's response by searching the previous agent's question and, if not found, also search the previous user utterance as it was described in the Section 4.4.1. The result for the user's utterance 'convertible' in Figure 4.10 would be a *phrase category* with label '[NP_car]'. This process is implemented on utterances that contain only one word, whereas if a descriptive word is found in a complex noun phrase, then the word is only added to the pattern list.

Regarding the pattern list generation for the *phrase categories*, a different grouping approach from the *word category* creation step is followed. A *phrase category* contains *word category* patterns that were found to have the same label. Moreover, in certain complex noun *phrase categories* the patterns may contain individual *word categories*. For example, the phrase "a table for four" would result in a complex *phrase category* with the pattern "[NP_table] [ADP] [NUM]" containing one simple *phrase category* and two *word categories*.

I want to rent a car

⇩

Phrase Categories :

[VP_want] = [PRON] [want] [PART]

[NP_car] = [DET] [car]

Word Categories :

[PRON]	=	I	[rent]	=	rent
[want]	=	want	[DET]	=	a
[PART]	=	to	[car]	=	car

;

(a) *An example of a grammar produced by a simple sentence with a 'subject' phrase that contains an assisting verb.*

May I rent a sports car for four days

⇩

Phrase Categories :

[AUX_PRON] = [AUX] [PRON]

[NP_car] = [DET] [TYPE_car] [car]

[NP_day] = [ADP] [NUM] [day]

[COMPLEX_NP_car] = [NP_car] [NP_day]

Word Categories :

[AUX]	=	May	[car]	=	car
[PRON]	=	I	[ADP]	=	for
[rent]	=	rent	[NUM]	=	four
[DET]	=	a	[day]	=	days
[TYPE_car]	=	sports			

(b) *An example of a grammar produced by a complex sentence. The semantic grammar contains an adjective, a generic labeled phrase category and a complex noun phrase category.*

Figure 4.13: *The outputs of the semantic category creation phase for a simple and a complex sentence.*

4.5 Relation analysis

The semantic category creation phase can generate *phrase categories* that are semantically synonymous. Since these phrase categories convey the same meaning, their patterns should be grouped under the same *phrase category*. This is achieved by utilizing relation analysis. After every utterance has been parsed through the method, the relation list is analyzed in order to find synonymous *phrase categories* that can be grouped together. For the purposes of this thesis, relations will be defined with a different type of entity. The entities of the relations are phrase entities instead of individual word entities. A relation comprises a left side, a right side and a verb. A relation is added to the relation list only if all of its components are not empty strings. This means that each side of the relation should contain a 'subject' and an 'object'. Each relation is defined as a triplet of type (left side, verb, right side) instead of the format verb(entity, entity) that was described in Section 3.4. For example, the sentence "I want to rent a car", will result in two *phrase categories* '[VP_want]', '[NP_car]'. The relation extracted from that sentence will be '([VP_want], rent, [NP_car])'.

The method parses the relation list two times in order to find synonymous *phrase categories*. During the first parse, the goal is to find generic labeled *phrase categories* that are semantically synonymous to a verb or noun *phrase category*. This is achieved by examining each relation first for the left side and then for the right side. This is explained by the following steps for every sentence and for each side of the relation. These steps are considered only for phrases that do not contain a 'NP_' or 'VP_' prefix:

- Find the first relation in the relation list with the same verb and different left (right) side (depending on the relation side that is examined) that has a phrase label that contains 'NP_' or 'VP_'.

- If found, replace the label relation's left (right) side with the label of the *phrase category* in the left side of the relation found in the previous step.

- Add the pattern of the *phrase category* with label equal to the one replaced in the previous step, to the set of the *phrase category* that corresponds to the replacement label in the previous step. Delete the replaced generic labeled *phrase category*.
- If there does not exist a relation with the same verb, find a relation with the same right(left) side that has a left (right) side phrase label that contains 'NP_' or 'VP_' .
- If found, replace the label of the relation's left (right) side with the label of the *phrase category* in the left side of the relation found in the previous step.
- Add the pattern of the *phrase category* with label equal to the one replaced in the previous step, to the set of the *phrase category* that corresponds to the replacement label in the previous step. Delete the replaced generic labeled *phrase category*.

Table 4.2: A relation list example before the relation analysis phase. This relation list is the result produced by the method for the rental dataset described in Section 4.1. The relation list contains generic labeled *phrase categories*, such as '[AUX_PRON]' and semantically synonymous *phrase categories* such as '[VP_want]' and '[VP_like]'.

Left Side	**Verb**	**Right Side**
[VP_want]	rent	[NP_car]
[VP_like]	rent	[NP_car]
[AUX_PRON]	rent	[COMPLEX_NP_motorcycle]
[VP_like]	book	[COMPLEX_NP_car]
[VP_want]	rent	[COMPLEX_NP_car]
[VP_be]	rent	[COMPLEX_NP_car]
[VP_like]	rent	[COMPLEX_NP_car]
[AUX_PRON]	rent	[COMPLEX_NP_car]

Table 4.3: The relation list after the first parse step of the relation analysis phase. The generic labeled *phrase categories* have been replaced by the equivalent verb *phrase categories* due to similar verb or right side noun *phrase categories*.

Left Side	**Verb**	**Right Side**
[VP_want]	rent	[NP_car]
[VP_like]	rent	[NP_car]
[VP_want]	rent	[COMPLEX_NP_motorcycle]
[VP_like]	book	[COMPLEX_NP_car]
[VP_want]	rent	[COMPLEX_NP_car]
[VP_be]	rent	[COMPLEX_NP_car]
[VP_like]	rent	[COMPLEX_NP_car]
[VP_want]	rent	[COMPLEX_NP_car]

The second parse involves detection of semantically synonymous noun or verb *phrase categories*. This is achieved similarly to the first parse, but the search is not confined to only generic labeled *phrase categories*. Each side of every relation is examined with the following steps :

- Find all the relations with the same verb and different left(right) side as the currently examined relation and store them in a *same verb* relation list.
- Find the frequency of the current relation's left(right) side label in the relation list.
- Find the frequency of the left(right) side labels of all the *same verb* relations.
- For each pair of current relation's left(right) side label and *same verb* relation's left (right) side label compare their frequencies :
  - Add the patterns of the *phrase category* that correspond to the label with the lower frequency to the *phrase category* that corresponds to the other label, but only the patterns that do not already exist in the other *phrase category.*
  - Delete the *phrase category* with the lower frequency.
  - Replace the labels of all the left (right) side labels in the relation list with the same verb that belong to the deleted *phrase category*, with the label of the other *category.*
  - If the label with the lower frequency belongs to the currently examined relation then proceed to the next relation.

After the relation analysis step, the patterns of generic labeled *phrase categories* are grouped under verb or noun *phrase categories* that were found to be semantically synonymous with. Moreover, *phrase categories* are grouped based on semantic synonymity in order to achieve a better coverage of the possible patterns under the same *phrase category.* After the first parse the relation list depicted in Table 4.2 takes the form in Table 4.3, where there are no generic labeled *phrase categories* in the list. The *phrase category* '[VP_want]' now contains the pattern of the generic labeled *phrase category* '[AUX_PRON]'. After the second parse, the relation list now has the form in Table 4.4. The *phrase categories* '[VP_want]', '[VP_be]' and '[VP_like]' are grouped into the same *category* '[VP_want]'. This means that all the patterns in '[VP_be]' and '[VP_like]' are now part of the '[VP_want]' *phrase category.*

Table 4.4: The relation list after the second parse step of the relation analysis phase. Every left side *phrase category* has been grouped under the '[VP_want]', due to similar verbs or right side *phrase categories.*

Left Side	**Verb**	**Right Side**
[VP_want]	rent	[NP_car]
[VP_want]	rent	[NP_car]
[VP_want]	rent	[COMPLEX_NP_motorcycle]
[VP_want]	book	[COMPLEX_NP_car]
[VP_want]	rent	[COMPLEX_NP_car]
[VP_want]	rent	[COMPLEX_NP_car]
[VP_want]	rent	[COMPLEX_NP_car]
[VP_want]	rent	[COMPLEX_NP_car]

4.6 Evaluation

For the evaluation of the method a semantic grammar was manually created by the author. The manually generated semantic grammar (MGG) is considered as the ground truth for the experiments and follows the same thought process as the method, meaning that the semantic grammar is created based on the semantic category creation phase, as well as the relation analysis phase mentioned in the previous sections. The evaluation is implemented by comparing the labels and set of words or patterns (sets) of the *categories* between the MGG and the semantic grammar created by the method. A *category* of the MGG is defined as an *expected category.*

First, the *categories* with correct labels are found and the accuracy and precision of the method's ability to create correct *category* labels is examined. The metrics are called acc_{CL} and prec_{CL}. The acc_{CL} is defined as the number of correct labels divided by the number of *categories* found by the method and expressed as:

$$\text{acc}_{CL} = \frac{L}{T}, \tag{4.1}$$

where L = number of found *categories* with correct labels, T = total number of *categories* found by the method. The prec_{CL} is defined as the number of correct labels divided by the number of *expected categories* and expressed as:

$$\text{prec}_{CL} = \frac{L}{N}, \tag{4.2}$$

where L = number of found *categories* with correct labels, N = total number of *expected categories.*

Second, the *categories* with correct sets are found and the accuracy and precision of the method's ability to create correct sets is examined. A *category* has a correct set if the set contains the same words or patterns with an MGG's *category* set. Moreover, it is not mandatory for a *category* to have the same label as the ground truth in order to be considered correct for this comparison. The metrics are named acc_S and prec_S. The acc_S is defined as the number of *categories* with correct sets divided by the number of *categories* found by the method and expressed as:

$$\text{acc}_S = \frac{S}{T}, \tag{4.3}$$

where S = number of found *categories* with correct sets, T = total number of *categories* found by the method. The prec_S is defined as the number of *categories* with correct sets divided by the number of the *expected categories* and expressed as:

$$\text{prec}_S = \frac{S}{N}, \tag{4.4}$$

where S = number of found *categories* with correct sets, N = total number of *expected categories.*

Lastly, the overall performance is examined by finding the method's *categories* with correct labels and the correctly allocated subsets. A subset represents a word in cases of *word categories* and a pattern in cases of *phrase categories.* A subset that is grouped in the correct *category* set is defined as correctly allocated. The metrics for the overall performance are named $\text{acc}_{\text{total}}$ and $\text{prec}_{\text{total}}$. The $\text{acc}_{\text{total}}$ is defined as:

$$\text{acc}_{\text{total}} = \frac{\sum_{i=1}^{T} t_i H_i}{T}, \tag{4.5}$$

where t_i = the value t for the i_{th} *category* that is 0 for a *category* with a wrong label and 1 otherwise, T = total number of *categories* found by the method, H_i = the average number of correctly allocated subsets found for the i_{th} *category* set. The H_i is defined as :

$$H_i = \frac{\sum_{j=1}^{Q} h_{ij}}{Q}, \tag{4.6}$$

where h_{ij} = the value h for the j_{th} subset of the i_{th} *category* that is 1 if the subset belongs to the *category* set and 0 otherwise, Q = total number of subsets found for the i_{th} *category.* The $\text{prec}_{\text{total}}$ is defined as:

$$\text{prec}_{\text{total}} = \frac{\sum_{i=1}^{T} t_i G_i}{N}, \tag{4.7}$$

where t_i = the value t for the i_{th} *category* that is 0 for a *category* with a wrong label and 1 otherwise, T = total number of *categories* found by the method, G_i = the average number of the correctly allocated subsets found for the set of the i_{th} *expected category,* N = total number of *expected categories.* The G_i is defined as:

$$G_i = \frac{\sum_{j=1}^{Q} h_{ij}}{D}, \tag{4.8}$$

where h_{ij} = the value h for the j_{th} subset of the i_{th} *category* that is 1 if the subset is correctly allocated to the *category* set and 0 otherwise, Q = total number of subsets in the i_{th} *category,* D = total number of subsets for the i_{th} *expected category.*

5 Results and discussion

The evaluation of the method was conducted by creating a semantic grammar based on a sample of dialogues from the bAbI dataset and two semantic grammars based on the rental dataset. Both of the datasets contained fifteen dialogues. The results include quantitative data regarding the *categories* created and the performance of the method. The evaluation of the method, as described in Section 4.6, was based on a MGG, where the *categories* created by the method were compared to the *categories* from the MGG. There were three types of comparisons, namely *category* label, set and subset comparison. The evaluation process aimed to understand the ability of the method to create *categories* with correct labels and sets. An important aspect of the evaluation process was the identification of the user input patterns that resulted in the method failing, as by finding these exceptions the rule-based model can be extended and keep evolving with the aim of reaching universal coverage regardless of the domain of the dataset. Overall the method found most of the *category* subsets for both datasets. The performance of the method was considerably better for *word categories* than for *phrase categories*. *Phrase categories* were produced less precisely in the presence of user input that deviated from the expected types of patterns.

5.1 BAbI dataset

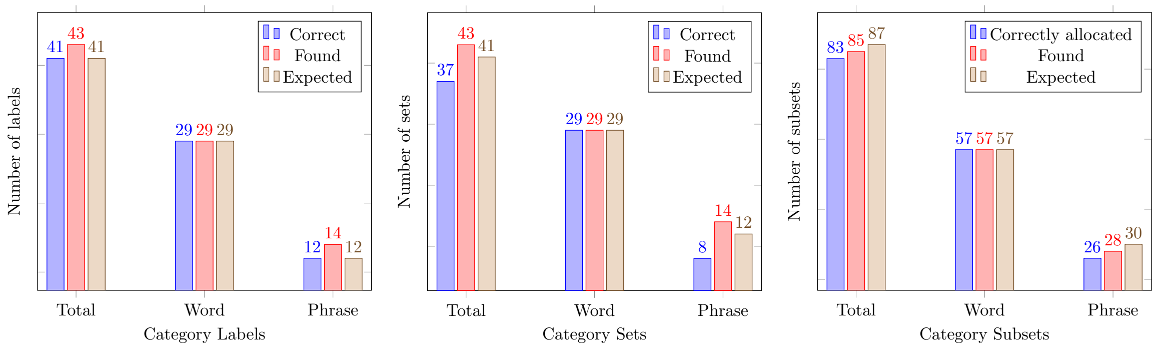


Figure 5.1: *Data obtained from the bAbI dataset. Each figure contains quantitative data about the number of correct labels, sets and correctly allocated subsets respectively.*

The bAbI dataset was employed as the "training" dataset for the method, thus the translation process and the rule-based model were curated for the type of dialogues found in the dataset, such as in Figure 4.2. The bAbI dataset's particularity was that it contained dialogues with patterns that resulted in complex *phrase categories* with different combinations of simple *phrase categories*. For example, the user inputs "may i have a table in a moderate price range with british cuisine for two in bombay" and "can you book a table with french cuisine for eight people in a cheap price range" resulted in the assignment of the patterns '[NP_table] [NP_range] [NP_cuisine] [ADP] [NUM] [ADP] [LOCATION]' and '[NP_table] [NP_cuisine] [NP_people] [NP_range]' to the complex *phrase category* with label '[COMPLEX_NP_table]'. Consequently the sample dialogues of the bAbI dataset produced a '[COMPLEX_NP_table]' *phrase category* with nine different patterns that can be observed in Figure 5.2. These patterns are populated by a few simple noun *phrase categories* arranged in different orders. It was expected for the method to preform well on the bAbI dataset, but the results indicate that the method was able to group patterns that contained the same head word under the correct complex *phrase category*. Furthermore, the method was able to distinguish between the different *phrase categories* present in a complex phrase and identify the semantics of both the whole phrase and the subphrases by creating complex *phrase categories* with 100% accurate labels.

It was inferred from the results in Figure 5.1 that the method was able to find the correct labels and sets for all word categories expected to be produced by the dataset. Moreover, the method was able to create a fair number of *phrase categories* with correct labels, but produces some *phrase categories* with wrong *category* sets. The number of correct subsets for the *phrase categories*, based on Figure 5.1, was notably higher than the correct sets. This indicates that while the method was not able to create completely correct *category* sets, the number of erroneously allocated subsets was low. This also explains the difference between the acc_S and $\text{acc}_{\text{total}}$ of the *phrase categories* in Table 5.2. The difference between acc_{CL} and prec_{CL} in Table 5.2 shows

[COMPLEX_NP_table] = [NP_table][NP_range][NP_cuisine][ADP][NUM][ADP][LOCATION],

[NP_table][NP_cuisine], [NP_table][ADP][NUM][NP_food],

[NP_table][ADP][LOCATION][ADP][NUM][NP_cuisine][NP_range],

[NP_table][NP_people][NP_range], [NP_table][NP_cuisine][NP_people][NP_range],

[NP_table][NP_food][NP_people][ADP][LOCATION],

[NP_table][ADP][LOCATION][ADP][NUM][NP_cuisine],

[NP_table][NP_range][ADP][LOCATION]

Figure 5.2: *The '[COMPLEX_NP_table]' phrase category created from the analysis of the sample dialogues from the bAbI dataset. The category contains nine different patterns. Some patterns only contain simple phrase categories and some others also contain word categories.*

that although more *phrase categories* than needed were created, the *category* labels that were expected to be produced were all found by the method. This was also validated by the data in Figure 5.1. The lower *phrase category* prec$_S$ in Table 5.2 and the data in Figure 5.1 indicate that the correct *phrase category* sets found were less than the ones expected to be found. Nevertheless, the *phrase category* prec$_{\text{total}}$ was higher than the *prec*$_S$, in Table 5.2, entailing that the method was able to identify correctly all the expected *phrase category* labels and map patterns to the correct *phrase category* at a high rate.

Table 5.1: Accuracy and precision metrics for the total number of *categories* produced by analyzing the bAbI's sample dialogues.

acc$_{CL}$(%)	**acc**$_S$(%)	**acc**$_{\textbf{total}}$(%)	**prec**$_{CL}$(%)	**prec**$_S$(%)	**prec**$_{\textbf{total}}$(%)
95.3	86	95.3	100	90.2	94.5

Table 5.2: Accuracy and precision metrics for *word* and *phrase categories* produced by analyzing the bAbI's sample dialogues.

	acc$_{CL}$(%)	**acc**$_S$(%)	**acc**$_{\textbf{total}}$(%)	**prec**$_{CL}$(%)	**prec**$_S$(%)	**prec**$_{\textbf{total}}$(%)
Words	100	100	100	100	100	100
Phrases	85.7	57.1	85.7	100	66.7	86.8

5.2 Rental dataset

The handwritten rental dataset was employed as the "test" dataset; hence it contained cases of user input that the rule-based model of the method could not fully analyze. This created the need to understand the extent of the robustness of the rule-based model and estimate how other types of patterns of user input would affect the overall performance of the method. For the purposes of evaluating the method's ability to handle both seen and unseen types of patterns, two datasets were created based on the original handwritten dataset. One dataset contained only dialogues with already examined types of patterns and will be defined as rental$_H$ and the second dataset contained dialogues that also contained unseen types of patterns and will be defined as rental$_{\text{UH}}$. The difference between the two subdatasets was that the rental$_H$ contains dialogues such as in Figure 4.10, whereas the rental$_{\text{UH}}$ also contains dialogues such as in Figure 5.3. The goal was to observe the effect of these slightly deviated patterns on the resulted *categories*. The performance difference for the *phrase categories* between Figure 5.5 and Figure 5.6 indicate that even small deviations of expected patterns affect the results of the method. For example, the analysis of the dialogue in Figure 4.10 resulted in the correct *categories* observed in Figure 5.4a. On the contrary, the analysis of the dialogue in Figure 5.3 resulted in the *categories* observed in Figure 5.4b, where the word 'coupe' was assigned to a wrong noun *word category* with label '[coupe]' and the phrase "Maybe a coupe" was erroneously assigned to a noun *phrase category* with label '[NP_coupe]'.

User: I want to rent a car

Agent: What type of car?

User: Maybe a coupe

Figure 5.3: *An example dialogue of the rental$_{UH}$ dataset, where the phrase "Maybe a coupe" cannot be fully analyzed by the method.*

[TYPE_car] = coupe

[NP_car] = [TYPE_car]

(a) *The resulted categories from analyzing the last user utterance of the dialogue in Figure 4.10. The method allocated the word 'coupe' at the correct word category and correctly assigned the created word category at the '[NP_car]' phrase category.*

[NP_coupe] = [maybe] [DET] [coupe]

[coupe] = coupe

[maybe] = Maybe   [DET] = a

(b) *The resulted categories from analyzing the last user utterance of the dialogue in Figure 5.3. The method fails to assign the word 'coupe' to the '[TYPE_car]' word category and identify the phrase "Maybe a coupe" as a '[NP_car]' phrase category pattern.*

Figure 5.4: *Example of resulted categories generated from the dialogues in Figure 4.10 and Figure 5.3.*

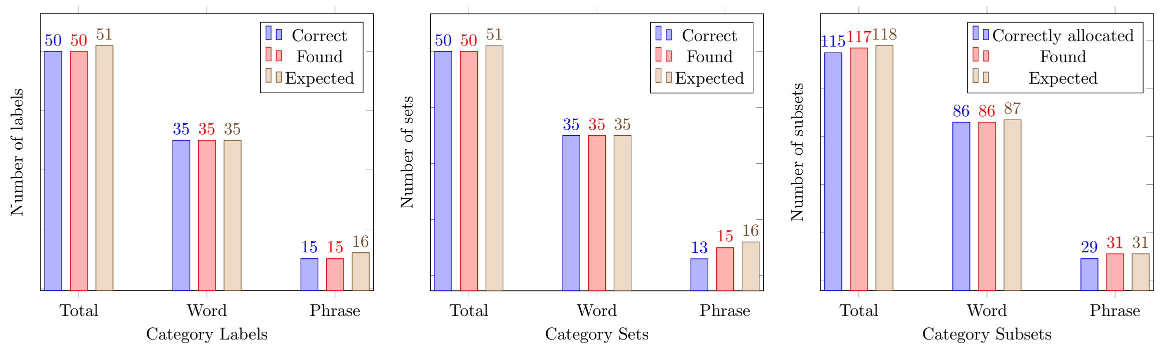


Figure 5.5: *Data obtained from the rental$_H$ dataset. Each figure contains quantitative data about the number of correct labels, sets and correctly allocated subsets respectively.*

In Table 5.3, the performance of the method for rental$_H$ was significantly higher than in rental$_{\text{UH}}$. All the *categories* produced for rental$_H$ were accurate, whereas there were few *category* sets that were not correct. The higher acc$_{\text{total}}$ indicates that the lower acc$_S$ was due to a small number of incorrect *category* subsets. This was further validated from Figure 5.5, where the number of incorrectly allocated subsets was significantly lower than the correct ones. Therefore, the method failed to group all the subsets for a few *categories* but was able to correctly identify almost all of the subsets. Despite that all the *categories* produced have correct labels, the lower precision metrics show that the method was not able to find all the *categories* that were expected to be found.

Table 5.3: Accuracy and precision metrics for the total number of *categories* produced by analyzing the two variations of the rental dataset.

	acc$_{CL}$(%)	**acc**$_S$(%)	**acc**$_{\textbf{total}}$(%)	**prec**$_{CL}$(%)	**prec**$_S$(%)	**prec**$_{\textbf{total}}$(%)
rental$_H$	100	94	98.8	98	92.2	97.7
rental$_{\text{UH}}$	84.7	69.5	83.75	98	80.4	93.7

Regarding the rental$_{\text{UH}}$ the method shows a relatively worse performance, which was expected. Although the method creates almost all the *expected categories*, it additionally produces unnecessary ones. The ability of the method to create correct *phrase categories* was lower in the presence of unseen patterns, whereas for the *word categories* the performance was more stable.

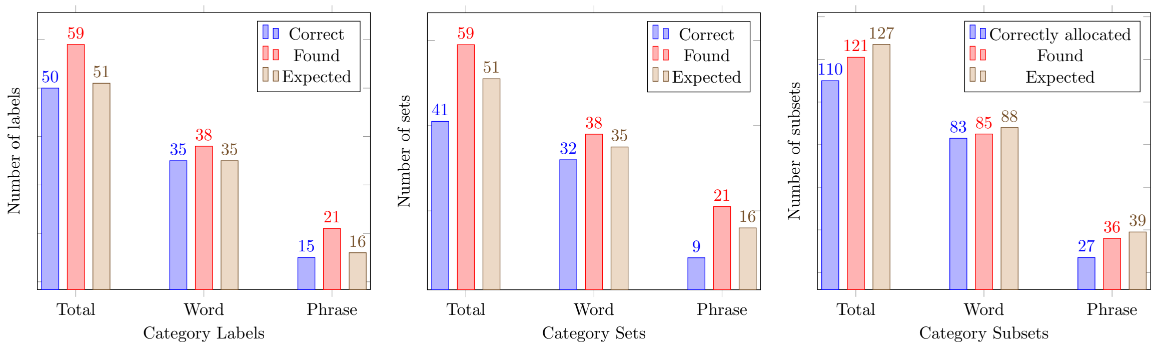


Figure 5.6: *Data obtained from the rental$_{UH}$ dataset. Each figure contains quantitative data about the number of correct labels, sets and correctly allocated subsets respectively.*

Table 5.4: Accuracy and precision metrics for *word* and *phrase categories* produced by analyzing the rental dataset. The index 'H' corresponds to the rental$_H$ dataset and the 'UH' to the rental$_{\text{UH}}$ dataset.

	acc$_{CL}$(%)	**acc**$_S$(%)	**acc**$_{\textbf{total}}$(%)	**prec**$_{CL}$(%)	**prec**$_S$(%)	**prec**$_{\textbf{total}}$(%)
Word$_H$	100	97.1	100	100	97.1	99.5
Phrases$_H$	100	86.7	96.1	93.8	81.3	89.6
Word$_{\text{UH}}$	92.1	84.2	94.7	100	91.4	97.3
Phrases$_{\text{UH}}$	71.4	42.9	96.1	93.75	56.3	82.8

5.3 Analysis

The method follows a hybrid approach for automated semantic grammar generation. The results of the thesis highlight that such an approach is able to produce both accurate and precise *semantic categories* at a high rate. This indicates that hybrid methods can both provide interpretability and performance without the need of intense manual labour from a developer. Furthermore, the relation analysis step provides additional benefits for understanding the semantics of a sentence. It is important to note that the two datasets used for the evaluation were morphologically different. The bAbI dataset contains dialogues with long sentences that include combinations of phrase patterns with little vocabulary variety, whereas the handwritten rental dataset contains the opposite type of dialogues. The method is able to handle both datasets with complicated sentences and datasets with a variety of descriptive words.

The creation of *word categories* for non-descriptive words relies mostly on the statistical POS tagger and the descriptive ones on the rule-based model of the method. The results indicate erroneous *word categories* can manifest in two cases. The first case involves wrongly assigned POS tags that depend on the performance of the POS tagger, which results in *word categories* with incomplete sets. The second one is derived from utterances with unseen pattern cases and results in the creation of more *word categories* than expected. One case observed is the presence of descriptive words, in the *rental$_{UH}$* dataset, that were not allocated to the correct *word category*. These descriptive words were part of an utterance with a small pattern that included a determiner or an interjection. By failing to understand the correct *category* of these words, the method was not able to map the pattern to the expected noun *phrase category*, thus resulting in the creation of wrong *phrase categories*. The method produces flexible results in terms of descriptive noun words. If a descriptive noun word is also found in a non-descriptive context, then the appropriate noun *word category* will be created. Additionally, named-entities that demand more sophisticated NER annotators and models such as duration and dates are not identified by the method and result in erroneous *word categories*.

Phrase categories are mostly derived from the semantic category creation phase of the method but also rely on the translation process. The semantic category creation phase needs a *Sentence* object that contains correctly allocated constituents and phrases in order to generate semantically meaningful *categories*. The translation of the parse tree is tied to two cases of analyzing and understanding a sentence. The first case involves the syntactic analysis of a user's input that contains simple patterns such as a noun phrase or a pattern of few words that produces a generic label. The second entails the handling of user input that is of type 'subject-verb-object'. This could be described as the most common scenario of initializing a request in a dialogue-based task-oriented system and the first case usually is derived from sequential responses of the user to questions of the agent. It can be assumed that there is no need for handling sophisticated conversations, if the user can provide all the needed information through single-word responses. Therefore, the method can be assumed to cover the necessary cases of user-agent interaction and consequently is able to successfully extract the semantics of such cases.

The method successfully groups noun phrases and allocates the correct head noun in complicated noun phrases. Complicated noun phrases could contain subpatterns that correspond to inferrable noun *phrase categories*. It can be argued that by not replacing those subpatterns with their equivalent noun *phrase categories*, the performance of the method could be hindered. This may be an issue if those patterns are individually mapped to a specific information slot that the agent needs or if that pattern could not be extracted from other instances of the dataset. In addition to the erroneous *phrase categories* that could be produced due to wrongly associated descriptive words, the method cannot extract semantically meaningful information from cases of nominal phrases with more than one non-adjective descriptive word. For example, a phrase with two consecutive descriptive nouns. It is important to note that this case occurs due to the morphology of the English language which allows such an exception. As English is the lingua franca of the modern world, such cases should be considered for developing semantic grammars. Lastly, there are a few cases where generic labeled 'subject' phrases could not be semantically decyphered by the relation analysis, due to absence of other relations with similar verbs or 'object' phrases. This problem might be a limitation even for humans in certain scenarios, but still the ability of extracting the semantics of such phrases should be further examined. It should also be noted that assimilated words were not present in the datasets examined, thus their effect on the method's performance was not investigated.

Since the method does not provide a universal coverage, there are types of patterns that the method cannot fully analyze, thus the rule-based model (translation, semantic category creation and relation analysis) will require additional curation. Nevertheless, the ability of the method to be able to reach universal coverage without solely relying on a training dataset provides an alternative in situations where labeled training samples are not available or complicated semantic meanings cannot be perceived by exclusively supervised methods. Lastly, the use of an unlexicalized constituency parser seems to not negatively affect the results and the use of other parsers such as a shift reduce parser did not alter the results of the method. This is due to the fact that the highest level of pattern complexity observed in the datasets were simple sentences. It can be assumed that these kind of patterns are generally expected to be analyzed by dialogue-based task-oriented systems, but in case of datasets with more complicated patterns the use of more sophisticated parsers should be examined.

6 Conclusion

This thesis examined dialogue-based task-oriented systems that rely on semantic grammars to understand user input. The manual generation of these grammars can become cumbersome and delay deployment of such systems. This highlights that there is a need for accurate automated methods. Similar algorithms are either domain specific, demanding a significant author's burden, focus mostly on semantic role labeling or lack in interpretability. This thesis aimed to develop a method that would be as automated as possible without compromising performance, interpretability and robustness. In order to achieve that, the method involved a series of preprocessing and analysis steps that tried to imitate a human-like way of processing a sentence. A combination of statistical tools, semantic rules and correlation of extracted analysis seems to work well for the problem of automated semantic grammar generation. The rule-based model of the method does not provide a universal coverage and this is why there are types of patterns that are not fully analyzed by the method. Nevertheless, the successful results highlight the importance and promising nature of hybrid methods for semantic analysis.

Considering the current limitations of the method, future research will focus on optimizing the syntactic analysis phase and enhancing the capability of the method to decypher more sophisticated and conversational dialogues. The first goal will comprise examining better trained statistical models to minimize erroneous POS tags and deep learning approaches, such as RNNs, as a better alternative compared to statistical methods. For the second goal, the future work will aim to enrich the rule-based part of the method for the purposes of expanding its case handling capabilities.

References

[1] Hongshen Chen et al. "A Survey on Dialogue Systems: Recent Advances and New Frontiers". *ACM SIGKDD Explorations Newsletter* **19** (Nov. 2017). DOI: `10.1145/3166054.3166058`.

[2] Sunil Issar and Wayne Ward. "CMLPs robust spoken language understanding system". Jan. 1993.

[3] Wayne Ward et al. "Speech Understanding in Open Tasks". *Proceedings of the Workshop on Speech and Natural Language*. HLT '91. Harriman, New York: Association for Computational Linguistics, 1992, pp. 78–83. ISBN: 1558602720. DOI: `10.3115/1075527.1075545`. URL: `https://doi.org/10.3115/1075527.1075545`.

[4] Michael McTear. "Chapter 9 - The Role of Spoken Dialogue in User–Environment Interaction". *Human-Centric Interfaces for Ambient Intelligence*. Ed. by Hamid Aghajan et al. Oxford: Academic Press, 2010, pp. 225–254. ISBN: 978-0-12-374708-2. DOI: `https://doi.org/10.1016/B978-0-12-374708-2.00009-7`. URL: `http://www.sciencedirect.com/science/article/pii/B9780123747082000097`.

[5] Aaron Radzinski. *"Introduction Into Semantic Modelling for Natural Language Processing"*. URL: `https://dzone.com/articles/introduction-into-semantic-modelling-for-natural-l`.

[6] Marsal Gavaldà and Alex Waibel. "Growing Semantic Grammars". *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 1*. ACL '98/COLING '98. Montreal, Quebec, Canada: Association for Computational Linguistics, 1998, pp. 451–456. DOI: `10.3115/980845.980922`. URL: `https://doi.org/10.3115/980845.980922`.

[7] Mattias Wahde. "A Dialogue manager for task-oriented agents based on dialogue building-blocks and generic cognitive processing". *2019 IEEE International Symposium on INnovations in Intelligent SysTems and Applications (INISTA)*. 2019, pp. 1–8. DOI: `10.1109/INISTA.2019.8778354`.

[8] Antoine Bordes et al. *"Learning End-to-End Goal-Oriented Dialog"*. 2017. arXiv: `1605.07683 [cs.CL]`.

[9] John M. Zelle and Raymond J. Mooney. "Learning Semantic Grammars with Constructive Inductive Logic Programming". *AAAI*. 1993, pp. 817–822. URL: `http://www.aaai.org/Library/AAAI/1993/aaai93-122.php`.

[10] David Devault et al. "Making grammar-based generation easier to deploy in dialogue systems". *Appears in SIGdial* (Jan. 2008). DOI: `10.3115/1622064.1622102`.

[11] Daniel Gildea and Daniel Jurafsky. "Automatic Labeling of Semantic Roles". *Comput. Linguist.* **28**.3 (Sept. 2002), 245–288. ISSN: 0891-2017. DOI: `10.1162/089120102760275983`. URL: `https://doi.org/10.1162/089120102760275983`.

[12] Eugene Charniak and Mark Johnson. "Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking". *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*. Ann Arbor, Michigan: Association for Computational Linguistics, June 2005, pp. 173–180. DOI: `10.3115/1219840.1219862`. URL: `https://www.aclweb.org/anthology/P05-1022`.

[13] Ann Taylor et al. "The Penn Treebank: An overview" (Jan. 2003). DOI: `10.1007/978-94-010-0201-1_1`.

[14] Steven Bird et al. *"Natural Language Processing with Python"*. 1st. O'Reilly Media, Inc., 2009. ISBN: 0596516495.

[15] Christopher D. Manning et al. "The Stanford CoreNLP Natural Language Processing Toolkit". *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Baltimore, Maryland: Association for Computational Linguistics, June 2014, pp. 55–60. DOI: `10.3115/v1/P14-5010`. URL: `https://www.aclweb.org/anthology/P14-5010`.

[16] Brian Roark. "Robust Probabilistic Predictive Syntactic Processing: Motivations, Models, and Applications" (June 2001).

[17] Donald Engel et al. "Parsing and Disfluency Placement". *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*. EMNLP '02. USA: Association for Computational Linguistics, 2002, pp. 49–54. DOI: `10.3115/1118693.1118700`. URL: `https://doi.org/10.3115/1118693.1118700`.

[18] Bernard E. M. Jones. "Exploring the Role of Punctuation in Parsing Natural Text". *Proceedings of the 15th Conference on Computational Linguistics - Volume 1*. COLING '94. Kyoto, Japan: Association for Computational Linguistics, 1994, pp. 421–425. DOI: `10.3115/991886.991960`. URL: `https://doi.org/10.3115/991886.991960`.

[19] Daniel M. Bikel. "Intricacies of Collins' Parsing Model". *Computational Linguistics* **30**.4 (2004), 479–511. DOI: `10.1162/0891201042544929`. URL: `https://www.aclweb.org/anthology/J04-4004`.

[20] Joakim Nivre et al. "Universal Dependencies v1: A Multilingual Treebank Collection". *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. Portorož, Slovenia: European Language Resources Association (ELRA), May 2016, pp. 1659–1666. URL: `https://www.aclweb.org/anthology/L16-1262`.

[21] W. Nelson Francis and Henry Kucera. *"Brown Corpus Manual"*. Tech. rep. Department of Linguistics, Brown University, Providence, Rhode Island, US, 1979. URL: `http://icame.uib.no/brown/bcm.html`.

[22] Daniel Jurafsky and James H. Martin. *"Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition"*. 1st. USA: Prentice Hall PTR, 2000. ISBN: 0130950696.

[23] Christopher D. Manning. "Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics?" *Computational Linguistics and Intelligent Text Processing*. Ed. by Alexander F. Gelbukh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 171–189. ISBN: 978-3-642-19400-9.

[24] Fred Karlsson et al. *"Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text"*. Jan. 1995.

[25] John D. Lafferty et al. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 282–289. ISBN: 1558607781.

[26] Zhiheng Huang et al. *"Bidirectional LSTM-CRF Models for Sequence Tagging"*. 2015. arXiv: `1508.01991 [cs.CL]`.

[27] Alan Akbik et al. "Contextual String Embeddings for Sequence Labeling". *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, Aug. 2018, pp. 1638–1649. URL: `https://www.aclweb.org/anthology/C18-1139`.

[28] Itiroo Sakai. "Syntax in universal translation". *1961 International Conference on Machine Translation of Languages and Applied Language Analysis*. Teddington,England. II. London: Her Majesty's Stationery Office, 1962, pp. 593–608.

[29] Eugene Charniak. "A Maximum-Entropy-Inspired Parser". *1st Meeting of the North American Chapter of the Association for Computational Linguistics*. 2000. URL: `https://www.aclweb.org/anthology/A00-2018`.

[30] Matthew Lease et al. "A Look at Parsing and Its Applications". Vol. 2. Jan. 2006.

[31] Jenny R. Finkel et al. "Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling". *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*. Ann Arbor, Michigan: Association for Computational Linguistics, June 2005, pp. 363–370. DOI: `10.3115/1219840.1219885`. URL: `https://www.aclweb.org/anthology/P05-1045`.

[32] Marti A. Hearst. "Automatic Acquisition of Hyponyms from Large Text Corpora". *COLING 1992 Volume 2: The 15th International Conference on Computational Linguistics*. 1992. URL: `https://www.aclweb.org/anthology/C92-2082`.

[33] Ellen Riloff and Rosie Jones. "Learning Dictionaries for Information Extraction by Multi-Level Bootstrapping." *AAAI/IAAI*. Ed. by Jim Hendler and Devika Subramanian. AAAI Press / The MIT Press, 1999, pp. 474–479. ISBN: 0-262-51106-1. URL: `http://dblp.uni-trier.de/db/conf/aaai/aaai99.html#RiloffJ99`.

[34] Anthony Fader et al. "Identifying Relations for Open Information Extraction". *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Edinburgh, Scotland, UK.: Association for Computational Linguistics, July 2011, pp. 1535–1545. URL: `https://www.aclweb.org/anthology/D11-1142`.

[35] Sergey Tihon. *"Stanford NLP for .NET"*. URL: `sergey-tihon.github.io/stanford.nlp.net/`.

[36] Jeroen Frijters. *"IKVM.NET"*. URL: `http://www.ikvm.net/index.html`.

[37] Kristina Toutanova and Christopher D. Manning. "Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger". EMNLP '00. Hong Kong: Association for Computational Linguistics, 2000, pp. 63–70. DOI: `10.3115/1117794.1117802`. URL: `https://doi.org/10.3115/1117794.1117802`.

[38] Kristina Toutanova et al. "Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network". *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*. 2003, pp. 252–259. URL: `https://www.aclweb.org/anthology/N03-1033`.

[39] Dan Klein and Christopher D. Manning. “Accurate Unlexicalized Parsing”. *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*. ACL ’03. Sapporo, Japan: Association for Computational Linguistics, 2003, pp. 423–430. DOI: 10.3115/1075096.1075150. URL: https://doi.org/10.3115/1075096.1075150.

[40] George A. Miller. “WordNet: A Lexical Database for English”. *Commun. ACM* **38**.11 (Nov. 1995), 39–41. ISSN: 0001-0782. DOI: 10.1145/219717.219748. URL: https://doi.org/10.1145/219717.219748.