ZÁPADOČESKÁ
UNIVERZITA
V PLZNI

Stochastic Semantic Parsing

PhD Study Report

Miloslav Konopík

Technical Report No. DCSE/TR-2006-01
May, 2006

Distribution: Public

Abstract

The recent achievements in the area of automatic speech recognition started the development of speech enabled applications. Currently it is beginning to be insufficient to merely recognize an utterance. The applications are demanding to understand the meaning of the utterance. Semantic analysis (or semantic parsing) is a part of the natural language understanding process. The goal of semantic analysis is to represent what the subject intended to say.

The thesis summarizes aspects of semantic analysis with emphasis on the stochastic approach to semantic analysis. Fundamental stochastic models along with the training and evaluation of these models are explained in details. Since, the performance of the system is significantly influenced by the way of preprocessing, it is also described in the thesis.

Copies of this report are available on
http://www.kiv.zcu.cz/publications/
or by surface mail on request sent to the following address:

University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitní 8
30614 Pilsen
Czech Republic

Contents

Chapter 1

Introduction

Speech is the most natural way of human communication. Therefore, there is an effort to incorporate speech control into human-computer interfaces. However, nobody likes the idea of remembering a large amount of specific commands. Hence, the ability of natural language understanding is crucial for many speech-enabled computer systems.

Natural Language Understanding (NLU) is a process whereby a computer algorithm extracts the meaning of an utterance and embeds the meaning in the computer model of the world. Semantic analysis is the first step (apart from the preprocessing) of the NLU process. The goal of semantic analysis is to represent what the subject intended to say in a way that would facilitate the process of contextual interpretation (reasoning about the meaning of the utterance).

In this thesis, we concentrate on the spoken language dialogue systems (this work is a part of the project of City Information Dialogue (CID) system [Mou04]). A semantic analyzer of a spoken language system must be able to deal with spontaneous speech effects such as unconstrained formulations, ill formed expressions, repairs, false starts and unknown words. Due to grammatical problems of spoken input, syntax should not play significant role during utterance processing. The CID corpus is in Czech language. Therefore, the issue of Czech language is considered in the summary of some methods.

Recent trends in the area of Natural Language Processing (NLP) are heading towards making all the processes of NLP stochastic. This thesis follows this trend, therefore it is focused on stochastic semantic parsing (section 3.3). Parsing means taking an input and producing some sort of structure for it. In this context, the meaning of word parsing is similar to word analysis, parsing however signifies that the result of semantic analysis is structured. Indeed, a tree structured output is typical for stochastic semantic parsers.

Although stochastic semantic parsing is the main topic of this thesis, the semantic analysis based on expert knowledge is discussed too (section 3.2). Many of the concepts of stochastic semantic parsing stem from semantic analysis based on expert knowledge. Semantic analysis based on expert knowledge is also more effective in some cases (see e.g. sec. 2.5).

1.1 Problem definition

This section specifies the problem of semantic parsing by the definition of the input that enters into the semantic parsing algorithm and the output that results from the algorithm.

Input: the orthographic transcription of an utterance. The form of the transcription can be either the most likely transcription of the utterance or even better a word lattice. Prosody or some nonverbal features may be included as well. Stochastic semantic analysis methods in particular profit from the presence of other information sources (prosody, etc.).

Output: the context-independent meaning of the utterance in a suitable meaning representation. The requirements about the output of semantic analysis are stated in the section 3.1. In short, the result of semantic parsing is required to support the contextual interpretation that follows the process of semantic analysis.

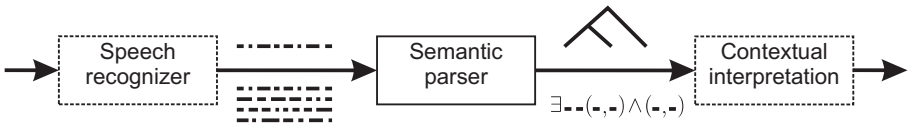

Figure 1.1. The interaction of the semantic analysis system with other modules. The semantic parser input is indicated as the most likely utterance transcription (top) or a word lattice (bottom). The output is either a tree (top) or a logic representation (bottom).

Chapter 2

Preprocessing

From the description of many understanding-related systems, it follows that the success of a NLP system depends on details. The preprocessing is one of the details that can significantly affect the performance of the whole system.

This chapter describes several possible ways of preprocessing which can be used during the utterance understanding process. Morphological tagging (section 2.1) is necessary for syntactic analysis (explained in section 3.2.1) because syntactic analysis does not build the syntactic tree on words but on morphological tags. When syntactic analysis is not used at all during the understanding process (frequent case), morphological tags are still being used as another knowledge source.

Lemmatization (section 2.2) is sometimes being referred to as simple morphological processing because it just transforms a word form to the basic form of a word. It does not examine morphological categories of a word. Lemmatization is useful especially for inflectionally rich languages (Czech, Russian, ...) because it decreases the number of various word forms in the lexicon.

Word sense disambiguation (section 2.3) helps to decrease the ambiguity in an utterance by distinguishing the sense of a word that was intended to be used by the speaker. Lower ambiguity means higher performance of the system.

Hierarchic structures (section 2.4) describe relations that hold among words in a language. The knowledge of the relations can increase the generalization ability of an understanding model.

Pre-parsing (section 2.5) uses different types of parsers to reliably determine the structure of isolated parts of an utterance.

2.1 Morphological Tagging

The basic unit of a sentence is usually assumed to be the word. The meaning of a sentence, though, is composed of words, the word itself is also a product of more primitive parts. The study of morphology concerns the construction of words from more basic components called *morphemes*. For example, the word "unbelievably" consists of the following morphemes: "un", "believe", "able" and "ly". Morphemes themselves are not very useful in the utterance understanding process, however, useful information can be derived from them. The useful information consists of

- lemma
- set of morphological categories

A *lemma* is the base form of a word. For example the word "flies" has the corresponding lemma "fly". The lemma is also called the canonical form of a word or the dictionary headword. The lemma is used in various applications, for example if it is needed to reduce the dictionary or to reduce the model size, etc.

A *set of morphological categories* describes morphological properties of words, such as: part-of-speech, number, gender, person, tense, voice, aspect, negation, degree of comparison, etc. For example the word "flies" has the following values of morphological categories: verb, singular number, 3rd person, present tense.

A *morphological tagger* (or simply *tagger*) is a programm that automatically assigns morphological *tags* to corresponding words. A tag is the label used to denote particular values of morphological categories in short. For example we can have tag "VBZ" to denote the following values of morphological categories "Verb, 3rd person singular present". Given this symbolism the tagger should assign the tag "VBZ" to the word "flies".

The tags together with the explanation are stored in a so called *tagset*. For example, the tag "VBZ" and its explanation was borrowed from the tagset called the "The Penn Treebank tagset" (described in [Mar93]).

The example in fig. 2.1 illustrates the process of morphological tagging. The words in the example are tagged with tags from the Penn Treebank tagset. The tags that were used in the example are listed in table 2.1. For the complete list, see [Mar93].

A variety of statistical morphological taggers were developed during last two decades. One of the best taggers is based on hidden Markov models[1]. It

[1]For more details, see very good tutorial on hidden Markov models in [Rab89]

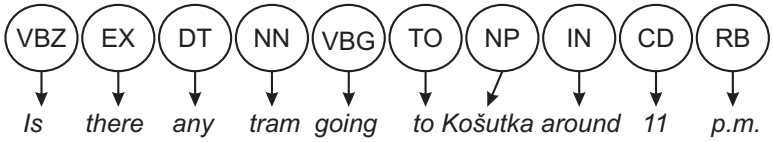

Figure 2.1. Example of morphological tagging.

Tag	Explanation
CD	Cardinal number
DT	Determiner
EX	Existential there
IN	Preposition or subordinating conjunction
NN	Noun, singular or mass
NP	Proper noun, singular
RB	Adverb
TO	to
VBG	Verb, gerund or present participle
VBZ	Verb, 3rd person singular present

Table 2.1. Penn Tree bank tagset example

was claimed that taggers based on Markov models can not compete with other current approaches (mentioned later in this thesis). However, it is shown in [Tho00] that the tagger based on Markov models performs at least as well as other taggers. The success of TnT (Trigrams'n'Tags) tagger [Tho00] is based on the precise solution of "small" details. The choice of smoothing of a trigram model (TnT uses linear interpolation[2]), handling of unknown words (TnT uses suffix analysis, more in [Tho00]) and other details guarantee good results for this tagger.

The transformation based tagger, described in [Bri95] is interesting on theoretical grounds. It does not use any statistical model and yet it performs very good. During training it learns transformations that improve the error rate. There are two components to a transformation: a rewrite rule and a triggering environment. An example of a rewrite rule for morphological tagging is: "Change the tag from modal to noun" and an example of triggering environment is "The preceding word is a determiner". The rules are learned automatically in iterations. Every iteration, the rule that decreases the error rate the most is selected. During tagging the rule is applied whenever the triggering condition is met. Rules have to be applied in the order that they were learned in.

[2]Linear interpolation for n-gram models is described in [MS01] – section 6.3.3

Other influential taggers include the Maximum Entropy tagger [Rtn99] and the Feature-based [HH98] tagger. Both these taggers use a log-linear model. The Feature-based tagger is targeted to inflectionally rich languages (such as Czech, Russian, etc). Because of the rich morphology, it uses a 15-position morphological tag (two positions are reserved) where every position denotes a morphological category (therefore the tags consist of 13 values of morphological categories). Every category is predicted separately in this approach.

The ultimate taggers are the so called voting taggers. A voting tagger consists of several other taggers. During tagging, every tagger "votes" a tag (or more tags) for a word. The tag that was contained in the most tagger outputs is voted and assigned to corresponding word.

The state-of-the-art taggers are now able to reach the human tagging performance. Please note that morphological tagging is often bewilderingly called the *part-of-speech tagging*. Morphological tagging is a more appropriate term, since other morphological categories than part-of-speech category are tagged.

2.2 Lemmatization

The goal of the lemmatization process is to find the basic form of each word (as was explained in the previous section) in an utterance. Lemmatization is usually performed together with morphological tagging. When tagger selects a tag it also selects the corresponding lemma.

Another option is to use the fact that lemmas are not as much ambiguous as morphological tags. Lemmas are then simply found in a table that contains the word forms (e.g. flies) and corresponding lemmas (e.g. fly).

2.3 Word Sense Disambiguation

Ambiguity is a big problem because it decreases the performance of syntactic and semantic parsers (parsers are explained later in this thesis). Word Sense Disambiguation (WSD) helps to decrease the ambiguity in the input by disambiguating the senses of words. The task of WSD is to examine words in context and specify exactly which sense of each word is being used. From now we will call the *target* word the word to be disambiguated.

For illustration, we can borrow an example from [JM00]. Consider the situation where we need to disambiguate the target word "bass" in the sentence: "An electric guitar and **bass** player stand off to one side, not really

part of the scene, just as a sort of nod to gringo expectations perhaps". The target word "bass" has (at least) two meanings. It can mean a tone of low pitch or a fish. The WSD algorithm has to determine which sense is the correct one.

In this thesis, we concentrate on machine learning approaches to WSD. The systems based on machine learning are trained to assign words of an utterance to one of a fixed number of senses. These systems classify words according to the context in which the words are embedded. In the following text we describe the *naive Bayes classifier* approach. For encoding the context we use the so called *co-occurrence feature vector*.

The co-occurrence feature vector consists of integers. Every position in the vector relates to a word and the integer value signify the number of times the word occurs in a region surrounding the target word. The region is most often defined as a fixed size window with the target word at the center. The words in the set are usually the most frequent words from a collection of sentences containing the target word. For example[3] the most frequent words from collection of sentences with the word "bass" could be: [fishing, big, sound, player, fly, rod, pound, double, runs, playing, guitar, band]. Using these words as features in the co-occurrence feature vector and using the window size of 10, the example sentence in the beginning can be represented by the following vector: [0,0,0,1,0,0,0,0,0,0,1,0]. This vector says that words "sound" and "guitar" are used once in the vicinity of the word "bass" in the example sentence.

The naive Bayes classifier approach to WSD tries to choose the most probable sense $\hat{s}$ given the co-occurrence feature vector V:

$$\hat{s} = \arg\max_{s \in S} P(s|V) \tag{2.1}$$

where S denotes the set of senses appropriate for the target word that is associated with vector V.

It would be difficult to collect statistic for the equation 2.1 directly so the Bayes' rule is applied (equation 2.2). The naive Bayes approach naively assumes that the features are independent of one another. Making this assumption we can estimate the probability $P(V|s)$ by the product of the probabilities of its individual features (eq. 2.3). Finally, since $P(V)$ is the

[3]This example is again borrowed from [JM00] and is based on a real corpus.

same for all possible senses, it can be ignored (eq. 2.4).

$$\hat{s} = \arg\max_{s \in S} \frac{P(V|s)P(s)}{P(V)} \tag{2.2}$$

$$\hat{s} \approx \arg\max_{s \in S} \frac{\left(\prod_j P(v_j|s)\right) P(s)}{P(V)} \tag{2.3}$$

$$\hat{s} \approx \arg\max_{s \in S} P(s) \prod_j P(v_j|s) \tag{2.4}$$

The classification then consists in the computation of equation 2.4 that selects the most probable sense $\hat{s}$ given the context vector V.

Given the equation 2.4, training a naive Bayes classifier consists of counting the number of times a word from the most frequent set occurs in the vicinity of a target word:

$$P(v_j|s) \approx \frac{count_{VS}(v_j, s)}{\sum_i count_{VS}(v_i, s)} = \frac{count_{VS}(v_j, s)}{count_S(s)} \tag{2.5}$$

The naive Bayes classifier presented here was chosen for demonstration purposes. WSD is a classification task, therefore any classification algorithm (stochastic supervised, stochastic unsupervised, symbolic, ...) can be used. A comprehensive list of WSD methods can be found in [NJ98].

One way to solve the WSD problem is to adapt a morphological tagging (sec. 2.1) algorithm to use word sense instead of morphological categories. The algorithms for morphological tagging, however, are not too appropriate for WSD task. To obtain reasonable results a wider context has to be used, but morphological taggers use a close context to predict morphological categories.

2.4 Hierarchical structures

The research in computational linguistic shows that it is useful to organize words of a language in a kind of hierarchy. The hierarchy describes lexical relations that hold between words. One example of a lexical relation can be the relation that associates a term with the more general expression of the term (for instance the term "trolleybus" to the term "public transport"). This relation is called the *hypernym* relation. There are other relations, for example hyponym relation (it is opposite to hypernym relation), part-of and has-part relation (describes parts of wholes and vice versa), antonym (relates words with opposite meaning), etc.

The usage of such hierarchic relations is wide. In the semantic analysis task, hierarchical relations can be used to help the semantic analysis to generalize. For example, if the semantic analysis is unable to find a rule to parse a phrase it can search for a more general term. The rule with the more general term is more likely to be present in the semantic analysis system. For example, if the system has no rule to parse the phrase "trolleybus goes from Slovany", the phrase "⟨PUBLIC TRANSPORT⟩ goes from Slovany" is more likely to be present in the system.

The most widely used lexical database for English is called *WordNet* [Fel98]. Each node of WordNet hierarchy consists of a *synset* of words with identical (or close to identical) meaning. The relations between synsets include the ones described formerly in this section and many others (organized separately for nouns, verbs and others parts-of-speech). The similar project is being developed at the MUNI[4] and is described e.g. in [PS04]. This project is targeted at Czech language.

2.5 Pre-parsing

This section is concerned with the problem that some phrases could be processed more easily by a different algorithm than the one that is used as the main approach to understanding utterances. For example, a machine learning based parser (see sec. 3.3) can be used as the main algorithm for parsing utterances. Although a machine learning based parser is able to reliably parse variety of natural language phrases, it has problems with very complicated phrases. Such phrases can be parsed by a dedicated parser based on a small set of hand written rules. The complicated phrases are therefore pre-parsed by the specialized parser. The main parser then works just with tokens which represent the complicated phrases.

Typical examples of complicated phrases are time and date phrases. The specialized parser can be used to translate times and dates into symbolic representation of time or date. Such a specialized parser, for instance, translates the phrase from "twenty five to six" into the "5:35" token.

Other problematic phrases are proper names (e.g. Slovany or Košutka are proper names and denote districts of the city Pilsen). Here, we can use a simple dictionary look-up algorithm to identify proper name phrases. Proper name phrases are then replaced by the token (e.g. token ⟨District⟩; 'tram goes to Slovany" → "tram goes to ⟨District⟩"). Of course, the problem of proper names is in fact slightly more complicated. The proper names can also consist of more then one word. The recognition algorithm moreover

[4]Masaryk University in Brno

does not produce capitalized output. Despite of these problems an improved look-up algorithm [Ben03] can be used.

The pre-parsing techniques are simple but they can significantly increase the performance of a natural language understanding system.

Chapter 3

Semantic Parsing

This chapter presents a number of computational approaches to the problem of semantic analysis, the process whereby meaning representations are composed and assigned to input utterances.

In the following, semantic will be restricted to the context-independent literal representation of sentences or phrases, ignoring phenomena such as irony, metaphor or conversational implication.

This chapter is organized as follows. At first the requirements that the semantic representation should fulfill are described. Then the main approaches of the semantic analysis based on expert knowledge are briefly mentioned. Many of the concepts of analysis based on expert knowledge are used in approaches that are based on stochastic models. The stochastic models create the main core of this thesis. First, they are explained theoretically in detail. Then the methodology of models training and evaluation follows. The theoretical explanation is extended by the description of existing systems that use these theoretical approaches.

3.1 Semantic Representation Requirements

The result of semantic analysis has to be represented in a correct representation that meets several requirements. The requirements are discussed in [JM00]. Fulfilling of these requirements could ensure that the following interpretation process will work with proper form of semantic representation. The incorrect semantic analysis output can disable successful interpretation of an utterance meaning.

The description of the requirements is shortened in this thesis. For a full description, see [JM00] – section 14.1. The requirements are:

- Verifiability

The natural language understanding system stores its information about the world in the knowledge base. The verifiability concerns a system ability to compare the state of affairs described by a representation to the state of affairs in some world as modeled in a knowledge base.

- Unambiguous Representations
  The final representation of an input meaning should be free from any ambiguity. Regardless of any ambiguity in the raw input, it is critical that a meaning representation language support representations that have a single unambiguous interpretation.

- Canonical Form
  Any linguistic input that has the same meaning in a domain should have the same meaning representation. Eg. the utterances "The tram is in Bory station at 11 pm" and "The tramcar arrives in Bory at 23:00" should have the same representation.

- Inference and Variables
  The term inference refers to a system ability to draw valid conclusions based on the meaning representation of inputs and its store of background knowledge. Variables allow replacing an unknown fact in representation with a variable.

- Expressiveness
  A meaning representation scheme must be expressive enough to handle all the wide range of subject matter concerning a domain.

For our purposes of statistical parsing (section 3.3) the Canonical Form, Inference and Expressiveness are crucial to fulfill. The FOPC[1] used in the Syntax-Driven Semantic Analysis (section 3.2.1) meets all the requirements listed in this section.

3.2 Semantic Analysis Based on Expert Knowledge

This section briefly describes the semantic analysis systems that are based on knowledge put into the system by an expert. That means that a human must make the rules for analysis up and then put them into the system. The expert uses a so called introspection to make these rules up. Sympathizers of

[1] First Order Predicate Calculus

semantic analysis based on expert knowledge claim that no computer learning algorithm is capable of introspection and so the rules learned by a computer learning algorithm can not compete with ones written by an expert.

But, the fact of the matter is that today the majority of semantic analysis systems are stochastically based (see section 3.3). The reason is the expensive development of expert systems. The other problem of development these systems is maintaining the consistency of expert rules. While the amount of rules is growing, the rules are starting to interact with each other.

The syntax-driven rule-by-rule approach to semantic analysis – discussed in section 3.2.1 – is attractive on theoretical grounds. From the theoretical point of view this system fulfils all the requirements that were proposed by linguists. It produces a rich logically-based representation of the meaning of a sentence and it allows to capture almost all the phenomena of the natural language. The insights from linguistics can be readily incorporated into this system. The problem with the syntax-driven approach is that building such a system is complex and many theoretical issues arise and slow the development of the system. Moreover, the purely syntax-driven systems have problems dealing with real langauge which is often ungrammatical. Thus a wide range of other systems was proposed.

The semantic grammars approach (section 3.2.2) is a method of writing grammars in terms of semantic rather than syntactic concepts. Hence, it is not so prone to have problems with ungrammatical utterances. Section 3.2.3 describes methods that use simple templates for computationally effective semantic analysis in specific domains.

3.2.1 Syntax-Driven Semantic Analysis

The Syntax-Driven Semantic Analysis ([All95] – chapter 9, [JM00] – chapter 15.1) was the leading approach to the semantic analysis in the 20^{th} century. The Syntax-Driven Semantic Analysis approach was strongly influenced by the work of [Mon74] that proposed the semantics for natural languages to be defined compositionally in the same way as the semantic for formal languages.

In this approach an input is first passed through a syntactic parser to derive its *syntactic analysis*. This analysis is then passed as input to a *semantic analyzer* to produce a *meaning representation* of an utterance. The meaning representation has to be powerful enough to capture all aspects of the natural language. The key idea underlying the semantic analysis in this approach is the *principle of compositionality*. The principle of compositionality is strongly related to the *lambda calculus* that enables to build the meaning of a phrase from sub-phrases. The emphasized terms from this paragraph will be described in this section.

1	S → NP VP	4	V → serves
2	NP → ProperNoun | Mass-Noun	5	ProperNoun → AyCaramba
3	VP → V NP	6	Mass-Noun → meat

Table 3.1. A simple syntactic grammar

Syntactic Analysis

The syntactic analysis uses a hand made database of context-free rules. The rules are carefully designed to handle all possible natural language phenomena like agreement[2], coordination[3], subcategorization[4], gaps[5] and so on. To handle all these phenomena easier, the features and unification[6] mechanism is added to a context free grammar.

Table 3.1 shows an example of a simple grammar (without the features extension) for a fraction of the natural language. Rules express the relation between the sub-phrases and their syntactic function in a sentence. For example the rule number 1 says that sentence (S) consists of a noun phrase (NP) and a verb phrase (VP), rule number 2 means that the noun phrase (NP) can be a proper noun or a mass noun, rule number 4 says that the verb (V) rewrites to the word "serves" and so on.

The grammar for natural languages (mainly for English) is defined in [All95] – chapter 5 or [JM00] – chapter 9 in detail. When the grammar is defined, a sentence is parsed according to this hand-made grammar by a standard parser (e.g. CYK, Early etc – for more details see section 3.3.2). The result of the syntactic parser (see fig. 3.1) is used by the semantic analyzer (presented further in this section).

Meaning Representation

The First Order Predicate Calculus (FOPC) is used for meaning representation in this approach. The advantage is that the FOPC provides a computational basis to satisfy all the requirements raised in section 3.1. In particular, the inference and variables are the outstanding properties of FOPC.

[2]Concerns inflectional morphology – the form of one word requires a corresponding form of another word; for more detail see [JM00] – section 9.6

[3]Stands for conjoining sentence phrases with conjunctions like *and*, *or*, etc.; [JM00] – chapter 9.5

[4]Concerns distinguishing predicates according to the arguments they take; [JM00] – chapter 9.7

[5]Stands for the movement of a phrase in questions or relative clauses; [All95] – section 5.6

[6]For explanation of terms feature and unification, see [All95] – chapter 4

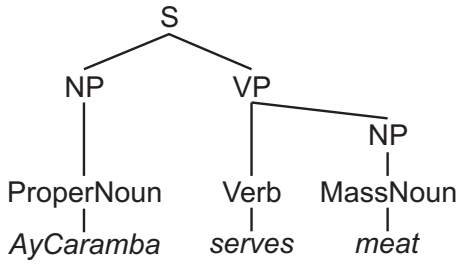

Figure 3.1. The example of a syntactic parse tree.

The FOPC is organized around the notion of the predicate. Predicates are symbols that refer to the relations that hold among objects in a domain. A reasonable FOPC representation for the sentence "I have a car" might look like the following formula:

$$Have(Speaker, Car)$$

This FOPC sentence asserts that the binary predicate *Have* holds between the objects *Speaker* and *Car*.

But, for the reasons that are presented in [JM00], the meaning representation should look like this:

$$\exists x, y Having(x) \wedge Haver(Speaker, x) \wedge HadThing(y, x) \wedge Car(y)$$

The reasons why the meaning representation has the presented form mainly concern fulfilling the requirements stated in the section 3.1. For the explanation of the form of the representation, see [JM00] – section 14.3.

The principle of compositionality

One of the principal assumptions often made about semantic interpretation is that it is a compositional process ([All95] – chapter 9.1). This means that the meaning of a phrase is derived incrementally from the meanings of its sub-phrases. For example, the meaning of the phrase "I want to go to Slovany" can be composed from sub-phrases "I want" and "to go to Slovany". And the sub-phrase "to go to Slovany" can be composed from "to go" and "to Slovany" and so on.

Compositional theories have some attractive properties. For instance, compositional models tend to make grammars easier to extend and maintain. Also, the semantic rules can be simpler and can deal with problems at separate levels.

But this theory also introduces many problems. For example a classic problem arises from quantified sentences or from the presence of idioms.

Lambda calculus

Lambda calculus is a mechanism for applying the compositional approach to the FOPC. It defines a way how to split a complicated statement into simpler ones and also a way how to combine two statements into one complex statement. The process of splitting is called *lambda abstraction* and the process of combination is called *lambda reduction*. Lambda calculus is explained in the following paragraphs first by formal definition and then by an example.

Lambda calculus ([All95] – section 9.1) is a powerful language based on simple a set of primitives. Formulas in the lambda calculus consist of equality assertions of the form:

`<expresion>` = `<expresion>`

The most crucial axiom in this system for our purposes is:

$$((\lambda x P x)a) = P\{x/a\} \tag{3.1}$$

where Px is an arbitrary formula involving the variable x and $P\{x/a\}$ is the formula where every instance of x is replaced by a. From this axiom, two principal operations can be defined: lambda reduction (moving from left to right across the axiom) and lambda abstraction (moving from right to left across the axiom).

The two principal operations are demonstrated by the following example. Consider the sentence "AyCaramba serves meat" where "AyCaramba" is a restaurant (borrowed from [JM00]). This sentence has the following logical form:

$$\exists e Isa(e, Serving) \wedge Server(e, AC) \wedge Served(e, Meat) \tag{3.2}$$

Now, consider the case when we need to compose the meaning from a noun phrase "AyCaramba" and a verb phrase "serves a meat". The noun phrase "AyCaramba" is a constant denoted by AC. But, how to capture the meaning of the verb phrase "serves meat"? It should be a predicate that is true of any object that serves meat. Here, we can use the lambda calculus that provides a formalism for such expressions: This expression is a predicate that takes one parameter x. This predicate is then true of any object O, such that substituting O for x in the expression results is a true proposition. Like any other predicate, you can construct a proposition from a lambda expression and an argument. In the logical form langauge, the following is a proposition:

$$(\lambda x \exists e Isa(e, Serving) \wedge Server(e, x) \wedge Served(e, Meat))(AC)) \tag{3.3}$$

Now, we can use the lambda reduction to get the expression 3.2 from the expression 3.3 by substituting x for AC. The lambda abstraction can produce the expression 3.3 from the expression 3.2.

Semantic Analysis

When the representation formalism and the lambda calculus are defined, the semantic analysis can be easily explained. The lambda calculus gives us a way how to build the meaning of a phrase compositionally from sub-phrases.

The semantic in the Syntax-Driven Semantic Analysis approach is defined by augmenting the syntax rules by the semantic information. Every syntax rule has a special feature[7] called *SEM* that contains the semantic information relating to a particular syntax rule. The semantic analysis process consists of extracting the semantic information from rules and applying the lambda reduction. The figure 3.2 contains an example of the semantic analysis of the sentence "AyCaramba serves meat".

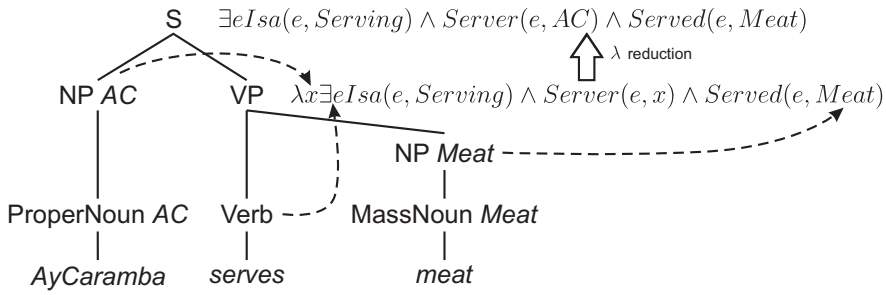

Figure 3.2. Example of semantic analysis based on syntactic parse tree. The analysis uses the λ reduction.

Summary of Syntax-Driven Semantic Analysis Approach

This approach provides very effective mechanism for retrieving knowledge from an utterance and for maintaining this knowledge. But, there is a reason why this approach is being replaced by stochastic methods (which of course produce much simpler representation) in some NLP[8] areas. The reason is the incredibly costly development of a Syntax-Drive Semantic Analysis system. The hand made rules are hard to maintain, hard to extend and hard to port[9] to another domain. The ability of this system to work with spoken input (full of speech errors) is very questionable. Although there are robust parsing techniques for this approach, this is still a problem.

The Syntax-Driven Semantic Analysis is the most "clever" system of all the semantic analysis systems presented in this work. It is able to produce

[7]The feature system extends a grammar to contain addition information. For definition of the feature system, see [All95] – chapter 4

[8]Natural Language Processing

[9]The term *porting* stands for the process of adapting a system to different domain than it was originally developed for.

#	Grammar rule	Example
1	TIME-QUERY → when does FLIGHT-NP leave	...
2	FLIGHT-NP → DET FLIGHT-C	*the flight*
3	FLIGHT-C → FLIGHT-N	*flight*
4	FLIGHT-C → FLIGHT-C FLIGHT-DEST	*flight to Chicago*
5	FLIGHT-C → FLIGHT-C FLIGHT-SOURCE	*flight from Chicago*
6	FLIGHT-C → FLIGHT-N FLIGHT-PART	*flight out*
7	FLIGHT-C → FLIGHT-PRE-MOD FLIGHT-C	*8 o'clock flight*
8	FLIGHT-DEST → to CITY-NAME	*to Chicago*
9	FLIGHT-SOURCE → from CITY-NAME	*from Chicago*
10	FLIGHT-N → flight	*flight*
11	DET → the	*the*
12	CITY-NAME → chicago	*Chicago*
	...	

Table 3.2. A simple semantic grammar

a rich output based on logical language. However, the problematic development of this system prevents it from a wider usage in practice.

3.2.2 Semantic grammars

The semantic grammars ([JM00] – section 15.5, [All95] – section 11.2) were originally developed for text-based dialogue systems in specific domains. The principle of semantic grammars is similar to the grammar for syntax presented in previous section (3.2.1). The difference is that relations are semantically rather than syntactically oriented. But, there is no precise line between a syntax and semantics in this approach. The syntax and semantics are freely mixed within a single rule.

Rules in a grammar consist of terminals and nonterminals. The terminals (denoted by lowercase letters) represent words of an utterance. The nonterminals (denoted by uppercase letters) represent semantic categories (that is the reason why these grammars are called semantic grammars). The rules and constituents of a semantic grammar are designed to correspond directly to entities and relations from the domain being discussed. The example (table 3.2, adapted from [All95]) presents a simple grammar for a fraction of a domain of a flight reservation system.

After the manual definition of the grammar, an input utterance can be parsed according to the grammar. Since the semantic grammar does not differ from a regular context-free grammar, a standard algorithm for parsing

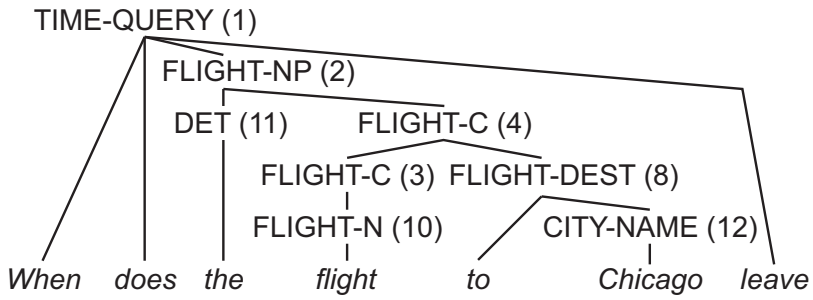

Figure 3.3. The parse tree for the example utterance. The numbers in brackets denote rule numbers of the example grammar in table 3.2

(e.g. CYK, Early etc – for explanation see section 3.3.2) can be used. The result of semantic parsing is represented by a parse tree. The parse tree for the utterance "When does the flight to Chicago leave?" is presented in fig. 3.3. The numbers of the rules that were used during the parsing are in the brackets.

One of the key motivations for the use of semantic grammars in specific domains was the need to deal with various kinds of anaphora and ellipses. The domain specific semantic rules are ideal to capture spoken language phenomena. Semantic grammars also combine aspects of syntax and semantics in a simple uniform framework. As a result semantic grammars are useful for rapid development of parsers in limited application domains.

Not surprisingly, there is a number of drawbacks to basing a system on a semantic grammar. The primary drawback arises from an almost complete lack of reuse in the approach. Combining the syntax and semantics of a domain into a single representation makes the resulting grammar specific to that domain. Semantic grammars are also susceptible to a kind of semantic overgeneration. Since the semantic grammars do not model the context of rules they could generate interpretations that are incorrect in a certain context.

3.2.3 Pattern matching

The pattern matching ([All95] – section 11.2)[10] semantic analysis can be used for very specific domains (e.g. summaries of newspaper stock transactions). The information in such domains usually falls into a fixed format that can be represented by a set of *patterns*. The principle of this approach consists in determining whether any part of the input *matches* a pattern from the set of patterns. If any match is found the information is extracted from specific

[10]In the [All95], a slightly different terminology is used

positions within the pattern. The extracted information is usually stored in a frame-based[11] representation.

Of course, the input utterance may match more than one pattern. Thus, it is necessary to try to match all the patterns from the set. Once all the possible patterns have been matched, the final stage of the analysis would merge the partial patterns.

For illustration, an example borrowed from [All95] is shown. In a domain where the system must generate summaries of articles reporting terrorist attacks in South America, the following pattern can be used:

take ⟨HUMAN⟩ hostage →
(TERRORIST-INCIDENT HUMAN-TARGET 1)

This pattern describes sentences containing the sequence consisting of the verb *take* followed by a phrase that describes a human, followed by the word *hostage*. When an input sequence matches this pattern the HUMAN-TARGET slot in a TERRORIST-INCIDENT frame is filled with the information contained in the ⟨HUMAN⟩ phrase.

To make this approach viable, the input must be parsed at least to the extent of identifying noun phrases (e.g. ⟨HUMAN⟩ in the previous example) and producing canonical forms for words (e.g. lemmatization[12]). A partial parsing technique can be used here to good effect. The partial parser is a syntactic parser that parses only certain syntactic categories (e.g. noun phrases etc). The partial parser takes advantage of the fact that certain parts of sentences can be parsed fairly reliably.

There are several reasons why not use a full rule-based parser in these domains: Firstly, it is not currently possible to construct a complete grammar for realistic domains, so many sentences will be unparsable. In addition, even if the sentence is parsable, it will likely be ambiguous between many different interpretations. Secondly, a full parser would be computationally expensive, specially for a system that processes millions of words a day.

Of course, a practical system would have to do additional processing rather than simply filling in the slots with the words. For instance, it would have to reason about dates, preprocess numbers etc.

This approach, while limited to a specific task, is capable of producing a more successful system than one based on general-purpose techniques. It takes advantages of the fact that the domain is very limited. For example, the preposition *to* usually indicates the recipient of the stock in the stock

[11]The frame-based representation is used for representing stereotypical information. The description of the frame-based representation can be found in [All95] – section 13.3

[12]The term lemmatization is explained in section 2.2

transaction domain. However, this needs not to be true in a different domain. For instance, in a public transport domain, the preposition *to* usually indicates the destination station.

Systems based on these techniques tend to be robust in that they produce some interpretation for nearly any input. Some systems attempt to gain the best of both worlds: They first use a full parser and semantic interpreter, and only if that fails they use pattern-matching techniques to extract what information they can from the sentence. This has the advantage of handling some sentences in detail but remaining robust when the full parse fails.

3.3 Stochastic Parsing

It may be foreseen from the description of the rule-based parsers that manual development of an understanding component by establishing and maintaining a system of rules is costly. In a stochastic method, semantic knowledge is usually represented by some kind of a tree (see fig. 3.4). Relations between semantic labels and the corresponding words are learned automatically from a large annotated training corpus and stored in the form of model parameters.

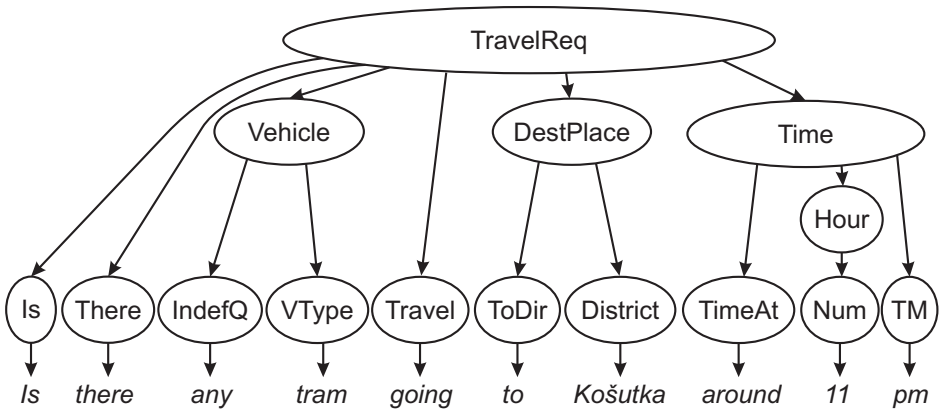

Figure 3.4. A tree representation of semantic knowledge

The central characteristic of tree structured representation is that individual concepts appear as nodes in a tree, with component nodes attached below them. For example, *Time* concept has component nodes *TimeAt*, *Hour*, *TM* in figure 3.4. The leaves of the tree are words of an utterance. It is often required that the order of component nodes must match the order of words they correspond to.

3.3.1 Flat Concept Parsing

There is a class of parsing methods which model the semantic parsing as a linear(flat) process rather than a hierarchic one. These methods are called *Flat concept parsing methods* (borrowed from [You02b]) in this work. Since the output of the semantic parser has to be structured there is a trick that makes the flat concept parsing to produce structured output (this trick is explained at the end of this section).

The Flat concept parsing is exactly the same process as the morphological tagging (described in 2.1). That means that you can take advantage of existing algorithms for morphological tagging. The main difference between morphological tagging and flat concept parsing is that instead of assigning morphological tags to the corresponding words, semantic labels are assigned to the corresponding words.

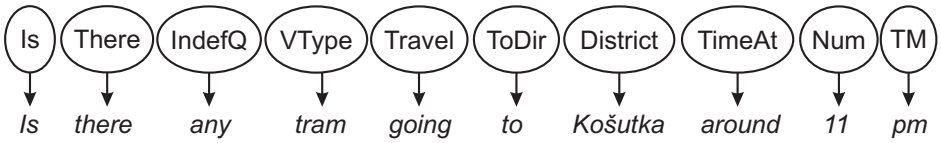

Figure 3.5. Example of flat concept parsing result.

The figure 3.5 shows a semantically labeled sentence. Some of the general terms are tagged with identical concepts (e.g. "is" is tagged with concept Is). A variety of methods (e.g. Hidden Markov Model tagger, Brill's tagger [Bri95] etc.) can be used to perform such a semantic labeling. The Hidden Markov Model (HMM, hereafter) approach is described in the following subsection.

Hidden Markov Models for Flat Concept Parsing

In the noisy channel model (fig. 3.6), the utterance production is modeled as follows: The user thinks in concepts (C) and produces actual words (W) in the presence of a noise (speech errors, actual lexeme selection from a synonymy class etc).

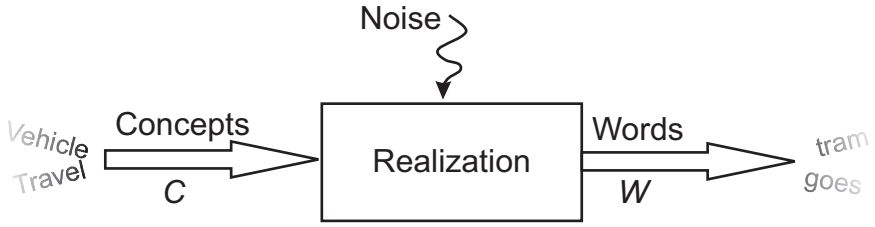

Figure 3.6. The noisy channel model.

The statistical decoder (3.4) determines the input concept sequence $\hat{C}$ from which W is most likely arisen ($\hat{C}$ is not necessary the same as the original C). Using Bayes' rule and with respect to the fact that the word sequence (W) is fixed we get equation 3.5:

$$\hat{C} = \arg\max_C P(C|W) = \tag{3.4}$$

$$= \arg\max_C \frac{P(W|C)P(C)}{P(W)} = \arg\max_C P(W|C)P(C) \tag{3.5}$$

where the $P(W|C)$ is often called the *lexical model* and the $P(C)$ is referred to as the *semantic model*.

In this work we assume left-to-right decoding. The actual word/concept depends only on previous words/concepts. The equation 3.5 can be reformulated by using the chain rule (3.6). Naturally, such model would require substantial amount of trainable parameters. Therefore, the context(history) should be limited (3.7).

$$\hat{C} \approx \arg\max_C \left\{ \prod_{t=1}^{T} P(w_t|W_1^{t-1}, C_1^t) P(c_t|C_1^{t-1}) \right\} \approx \tag{3.6}$$

$$\approx \arg\max_C \left\{ \prod_{t=1}^{T} P(w_t|W_{t-m}^{t-1}, C_{t-n+1}^t) P(c_t|C_{t-p}^{t-1}) \right\} \tag{3.7}$$

where $T = |W| = |C|$ is the length of the utterance, $W_{t_1}^{t_2}$ denotes word sequence $(w_{t_1}, ..., w_{t_2})$, $C_{t_1}^{t_2}$ denotes concept sequence $(c_{t_1}, ..., c_{t_2})$, t is "time" and m, n, p denote lengths of contexts.

If $m = 0$, $n = 1$, $p = 1$ we get conventional 1st order Markov model (as defined in appendix A.1). But, the 2nd order Markov model ($p = 2$, see in eq. 3.8) can be still reliably trained and is more often used for its higher predictive capability.

$$\hat{C} \approx \arg\max_C \left\{ \prod_{t=1}^{T} P_S(c_t|c_{t-1}, c_{t-2}) P_Y(w_t|c_t) \right\} \tag{3.8}$$

where P_S is the transition probability matrix and P_Y is the output probability matrix (see formal definition of Markov model in appendix A.1). Please note that all the assumptions presented in equations 3.6 – 3.8 involve a certain level of simplification needed to make the model computationally feasible.

The goal of decoder is to determine the most likely sequence of states given the input data (W) and model parameters ($\lambda = (P_S, P_Y)$):

$$\hat{C} = \arg\max_C P(C|W, \lambda) \tag{3.9}$$

The decoding is usually solved by a variant of Viterbi algorithm (3.10–3.13) which is an application of dynamic programming[13] technique. The most likely path from the beginning to state i is computed as the maximum of all previous states multiplied by the transition probability P_S and the output probability P_Y (follows from 3.6 – 3.8):

1. Initialization:

$$\delta_i(0) = \begin{cases} 1 & \text{if } i = 0,\ c_0 \text{ is initial state} \\ 0 & \text{if } i \neq 0 \end{cases} \tag{3.10}$$

2. Induction:

$$\delta_i(t) = \max_j \{\delta_j(t-1) P_S(c_i|c_j, ...)\}\, P_Y(w_t|c_i) \tag{3.11}$$

$$\psi_i(t) = \arg\max_j \{\delta_j(t-1) P_S(c_i|c_j, ...)\} \tag{3.12}$$

3. Most likely state sequence backtracking:

$$\begin{aligned} \hat{c}(T) &= \arg\max_i \delta_i(T) && \text{initialization} \\ \hat{c}(t-1) &= \psi_{\hat{c}(t)}(t) && \text{induction} \end{aligned} \tag{3.13}$$

where $\delta_i(t)$ is the probability of getting to state c_i at time t through the most likely path, $\psi_i(t)$ a backpointer which points to the state in time $t-1$ from which the optimal state in time t was reached and $\hat{C}_1^T$ is the resulting state sequence which most likely generated word sequence W_1^T.

When we come to higher order Markov model the question of pruning becomes vital. The *beam search* pruning (3.14) seems to be a good choice of pruning because it delivers high speedup (approximately by factor of thousands) at the cost of low precision loss.

$$\delta_i(t) \begin{cases} \geq \theta \max_j(\delta_j(t)) & \text{state is preserved} \\ < \theta \max_j(\delta_j(t)) & \text{state is discarded} \Rightarrow \delta_i(t) = 0 \end{cases} \tag{3.14}$$

where θ is a threshold.

The Markov model could be trained by supervised or unsupervised training. The supervised training needs a semantically annotated (like in fig 3.5) corpus. Then, the parameter estimation is only a matter of frequency

[13]Dynamic programming is a class of effective solution methods which compute the following state based on optimal computation of a previous state.

counting and an appropriate smoothing. The 3.15 shows MLE[14] of model parameters.

$$P_S(c_t|c_{t-1}, c_{t-2}) = \frac{\text{count}_{C3}(c_t, c_{t-1}, c_{t-2})}{\text{count}_{C2}(c_{t-1}, c_{t-2})}$$
$$P_Y(w_t|c_t) = \frac{\text{count}_{WC}(w_t, c_t)}{\text{count}_C(c_t)} \tag{3.15}$$

Certainly, these MLE estimates need to be smoothed. Despite the fact that papers are not usually precise about the details of the implementation (like smoothing) the good choice of smoothing seems to be the same as in TnT tagger (presented in section 2.1).

Unsupervised training is a harder task. Although the unsupervised training allows to significantly reduce the human effort needed to prepare the data, fully unsupervised training proved to be impossible, because the structures resulting from unsupervised training bear a little resemblance to any human-made linguistic structures. Thus, it appears that some priori knowledge must be present during the unsupervised training. A good balance between priori knowledge and model performance is shown in [HY05]. This approach is called *abstract semantic annotation* and will be described in section (3.3.7 – Hidden Vector State model) in detail. The actual parameter estimation is done by using Forward-Backward (Baum-Welch) algorithm (more details in [Rab89]), which is an application of the EM[15] algorithm.

To illustrate the tagging process an example follows. In the HMM semantic tagging approach to flat concept parsing, the utterance "Is there any tram going to Koutka around 11 pm?" could be tagged according to the figure 3.7. The model stays in the same state *Dummy* for first two words, then it stays in the state *VehicleType* for next two words and so on.

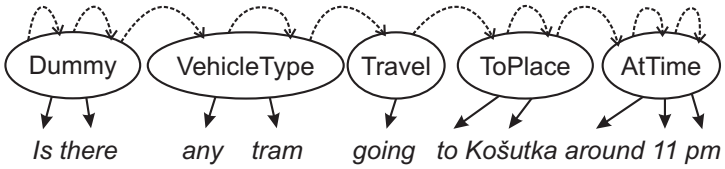

Figure 3.7. HMM semantic tagging.

Although this method is also used as a stand-alone semantic decoder ([Pie92], [Min99]) it suffers from a serious problem. It does not model the semantics as a hierarchic structure and therefore it is not capable of capturing

[14]MLE = Maximum Likelihood Estimate
[15]EM = Expectation Maximization [Dem77]

long dependencies. For instance, consider the example sentence borrowed from [CJ00]. In this example (illustrated in Fig 3.8) we try to predict the word **after** from its history. A 3-gram approach (A) would predict the word after from [7, cents], whereas it is intuitively clear that the strongest predictor (B) would be [ended], which is outside the reach of even 7-grams.

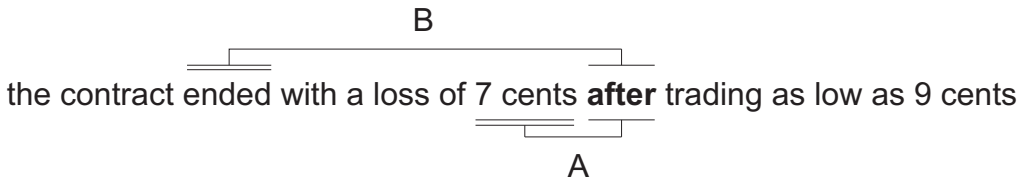

Figure 3.8. Long dependencies illustration.

The use of flat concept parsing

If it is desired to use this approach (incapable of modeling hierarchic structures) as a stand-alone semantic decoder able of producing structured output, the semantic label itself can be structured. For illustration, we can borrow an example from [Min99]. Figure 3.9 shows the conversion of semantically labeled sentence "probably sometime between nine and five would be good" (top) to corresponding semantic tree (bottom). The semantic labels are structured in this example, the components of a label are enclosed by parenthesis <>.

Although this approach is surpassed due to its problems by more sophisticated approaches (presented later), it is useful to present it here for two reasons. First, it provides the theoretical basis for Vector-state Markov Models (3.3.4). Second, it is used to determine the so called *preterminal* symbols in probabilistic grammars approach (3.3.2).

3.3.2 Probabilistic grammars for Semantics

A probabilistic context-free grammar (PCFG) for semantics logically follows from semantic grammars (presented in 3.2.2). Instead of creating the grammar manually, the grammar is inferred automatically from the data. The form of semantic analysis supported by the PCFG model is presented in figure 3.10.

In statistical parsing the PCFG has to determine the most likely parse tree given an input utterance. Because of its recursive nature, the probability of a parse tree has to be computed recursively. For better understanding, we first define the probability of a string given a grammar (expressed by

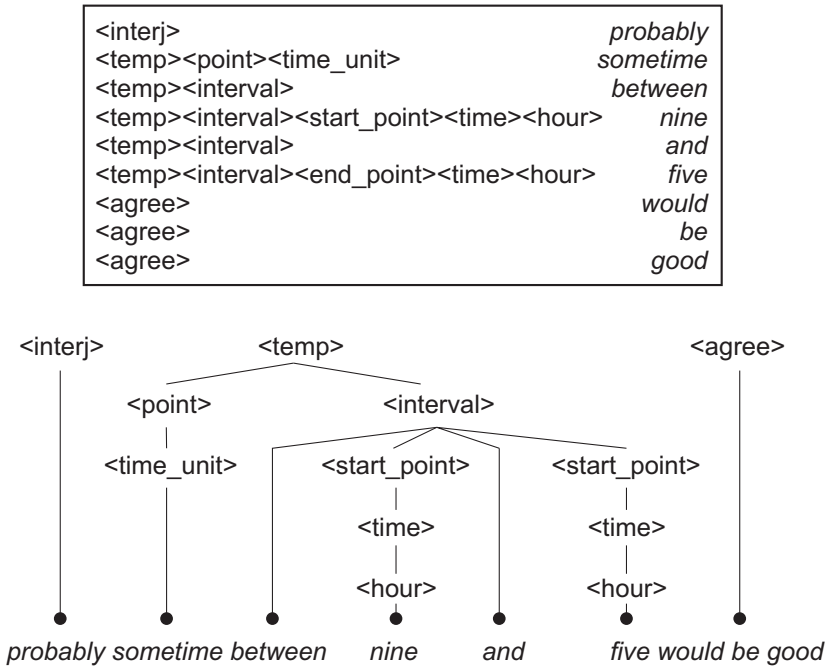

Figure 3.9. Conversion of structured semantic labels into a semantic tree.

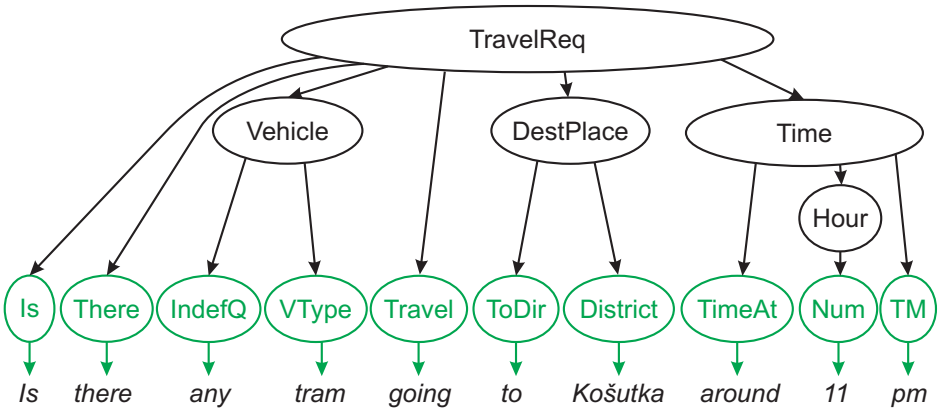

Figure 3.10. Example of PCFG semantic analysis. Preterminal symbols are marked with green color.

the β function). Then we use the β function to derive an algorithm for the computation of the best parse tree.

When we adopt bottom-up approach, we can compute the probability of a substring given a grammar using inside probabilities ($\beta_j(p,q)$). Inside probability formula for binary branching trees is defined in equation 3.16 and stands for probability of nonterminal N_j built up from a sequence of words $W_p^q = w_p, ..., w_q$. The β function sums the probability of all ways

that a certain constituent (N_j) can be built out of two smaller constituents by varying what the labels of the two smaller constituents are (N_r,N_s) and which words each spans ($w_p, ..., w_d$ and $w_{d+1}, ..., w_q$).

$$\beta_j(p,q) = \sum_{r,s}^{q-1} \sum_{d=p} P(N_j \rightarrow N_r N_S) \beta_r(p,d) \beta_s(d+1,q) \tag{3.16}$$

The β function can be easily generalized for trees with general branching (eq. 3.17). But the notation of generalized formula is complicated.

$$\beta_j(p,q) = \sum_{Q \leq d-p+1} \sum_{\{N_1 ... N_Q\}} \sum_{I_0^Q} P(N_j \rightarrow N_1 ... N_Q) \prod_{q=1}^{Q} \beta_q(I(q-1)+1, I(q)) \tag{3.17}$$

where $N_1 ... N_Q$ varies over all possible right sides of rule N_j and function I_0^Q partitions sequence of words $w_p ... w_k$ into $Q+1$ parts including the beginning $p-1$ in all possible ways (e.g. let $Q = 3$ and the partitioned sequence be $\{w_3, w_4, w_5, w_6\}$ then $I_0^3 = \{2,3,4,6\}$ and $I_0^3 = \{2,4,5,6\}$ and $I_0^3 = \{2,3,5,6\}$).

The derivation of the β function for both cases is demonstrated in fig. 3.11

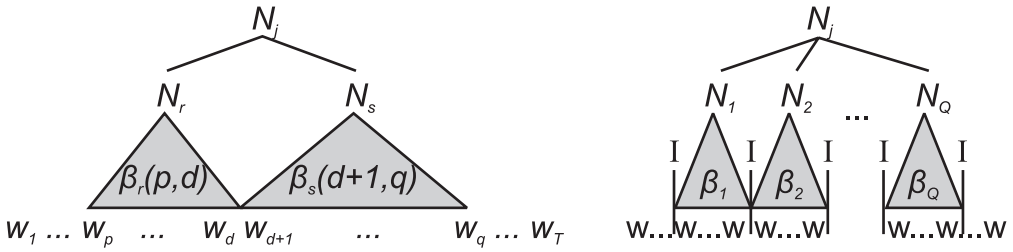

Figure 3.11. Derivation of inside probabilities for a binary branching tree (left) and for a general tree (right).

After initialization (3.18) we can compute the probability of a string according to the formula 3.19.

$$\beta_p(k,k) = P(N_p \rightarrow w_k) \, \forall k \in (1,T) \tag{3.18}$$

$$\beta_{root}(1,T) \tag{3.19}$$

where $T = |W|$ is the length of the string.

Now, we focus on the initialization for a moment. Nonterminal symbols N_p in equation 3.18 are called *preterminal* symbols. Preterminal symbols

(marked with green color in fig. 3.10) are the nonterminal symbols just above the words. After the definition of a lot of independence assumptions for PCFG (see in appendix A.3), the PCFG is too weak to reliably predict preterminal symbols because PCFG computes the probability of a rule $P(N_p \rightarrow w_k)$ as $P(w_k|N_j)$, so no context is involved. Thus a flat concept parsing technique (section 3.3.1) can be used to determine the preterminal symbols. Using the flat concept parsing technique for initialization we need to redefine the formula 3.18. There are two options how to redefine the formula depending on the style of the result of the flat concept parser. If the flat concept parsing technique assigns the most probable concept to each corresponding word we use the formula 3.20. If it assigns more concepts to each word with corresponding probability distribution, we use the formula 3.21.

$$\beta_p(k,k) = \begin{cases} 1 & \text{if } N_p = c_k \\ 0 & \text{otherwise} \end{cases} \tag{3.20}$$

$$\beta_p(k,k) = P(c_{k,i}|w_k) \text{ for } N_p = c_{k,i} \tag{3.21}$$

where $c_k/c_{k,i}$ is/are the concept/concepts assigned to word w_k by a flat concept parser and N_p is preterminal symbol which refers to the concept c_k.

Now, we can use the definition of the β function to find the most likely parse for an utterance given a grammar. Instead of summing the probabilities of all ways in which a certain constituent (N_j) can be built, we select the way with the maximum probability. Thus we substitute Σ for maximum in 3.16 and we get 3.24. The whole algorithm is very similar to the Viterbi algorithm and is described in 3.22–3.26.

1. Initialization:

$$\delta_p(k,k) = P(N_p \rightarrow w_k) \qquad \forall N_p \in N, 1 \leq k \leq T \tag{3.22}$$

$$\psi_p(k,k) = w_k \qquad \forall N_p \in N, 1 \leq k \leq T \tag{3.23}$$

2. Induction:

$$\delta_j(p,q) = \max_{r,s} \max_{p \leq d < q} (P(N_j \rightarrow N_r N_S)\delta_r(p,d)\delta_s(d+1,q)) \tag{3.24}$$

$$\psi_j(p,q) = \arg\max_{(r,s,d)} (P(N_j \rightarrow N_r N_S)\delta_r(p,d)\delta_s(d+1,q)) \tag{3.25}$$

3. Most likely parse tree backtracking:

$$\begin{aligned} &\mathrm{B}(\psi_1(1,T)) && \text{initialization} \\ &\mathrm{B}(\psi_j(p,q)) = \mathrm{NODE}(N_j, \mathrm{B}(\text{left}), \mathrm{B}(\text{right})) && \text{induction} \\ &\mathrm{B}(w_k) = w_k && \text{termination} \end{aligned} \tag{3.26}$$

where $\delta_j(p,q)$ is the highest inside probability parse of a subtree rooted in N_j and spans from w_p to w_j. $\psi_j(p,q)$ is a list of three integers recording the form of the rule application which had the highest probability (similar to back-pointers in Viterbi). We try to reconstruct the parse tree using the function B which recursively builds the most likely parse tree from start symbol N_1 that spans over entire utterance $(w_1, ...w_T)$. Function NODE$(N_j, left, right)$ defines a node with name N_j and with left and right subtree. Functions left and right take left and right pointer from $\psi_j(p,q)$. The building of the tree stops when the leaf (a word of the utterance) is reached.

The initialization step in 3.22 can be computed by 3.20 or 3.21. The induction step (in 3.24) assumes only binary branching. The generalization can be done similarly to the generalization of the β function (3.17). Presented approach assumes bottom-up parsing. The most likely parse tree can as also be computed by top-down approach using outside probability ([MS01], page 395).

Efficient parsers do not use the basic approach presented in 3.22–3.26 directly, but the approach is improved to be more efficient. The parsing is usually accomplished by a probabilistic variant of the Chart parsing, CYK[16] or Early algorithm in practise. These algorithms share the same core logic (the so called chart structure) although the mightiest of them, the Early, is far more complicated.

A basic chart parser [All95] uses a chart structure to record every constituent built during the parsing. The chart structure contains partial-parses ($\delta_j(p,q)$ values in equation 3.24). The CYK algorithm ([JM00], chapter 12) is optimized for CFG languages written in Chomsky normal form(CNF)[17]. Early algorithm [Ear70] uses so called dot rules to indicate the progress made in recognizing a rule. Early algorithm is based on the application of Predictor, Scanner and Complementer operators.

All these parsers go left-to-right during parsing. A basic chart parser and CYK build the parse tree bottom-up. Early builds it top-down. All these algorithms have the asymptotic time complexity $O(T^3)$, where T is the number of words in an utterance.

Again, there are two ways to learn the rules and their probabilities. The simpler way is to use a corpus of already parsed sentences (supervised training). Such a corpus is called a treebank. We create a new rule each time we find that a nonterminal N is expanded into a string α in the training set. The probability of each expansion of a nonterminal N can be computed by

[16] Cocke-Younger-Kasami

[17] A formal grammar is in Chomksy normal form iff all rules are of the form: $A \to BC$ or $A \to a$ or $S \to \varepsilon$, where A,B,C are nonterminals, a is terminal, S is the start symbol and ε is the empty string.

counting the number of times when expansion occurs and then normalizing.

$$P(N \rightarrow \alpha|N) = \frac{\text{Count}(N \rightarrow \alpha)}{\sum_{\gamma} \text{Count}(N \rightarrow \gamma)} = \frac{\text{Count}(N \rightarrow \alpha)}{\text{Count}(N)} \tag{3.27}$$

When a treebank is unavailable, unsupervised training has to be used. The standard algorithm for unsupervised PCFG training is called the *Inside-Outside* algorithm, which is again an EM training algorithm. The basic assumption is that a good grammar is one that makes the sentences in the training corpus likely to occur, and hence we seek the grammar that maximizes the likelihood of the training data. But there are problems. While the algorithm is guaranteed to increase the probability of the training corpus, there is no guarantee that the nonterminals that the algorithm learns will have any resemblance to the kind normally designed by a human. More over the algorithm usually gets stuck in a local maximum during learning. That means that the algorithm is very sensitive to the initialization of the parameters. The Inside-Outside algorithm and its problems are discussed further in [MS01] in chapter 11. The section 12.2 in [MS01] contains a solution to Inside-Outside algorithm problems. The solution is based on a partially unsupervised learning.

There are several key advantages of a hierarchical model when compared to a flat model discussed earlier. Firstly, the concepts are linked to a superior concept (e.g. Station to the SourcePlace, in a public transport domain). In more complex queries this type of explicit binding can be important. For instance in the utterance "I want to travel by the tram that goes from Slovany" it is crucial to attach the source station (Slovany) to the vehicle (tram) and not to the root of the sentence. Attaching to the root would mean that the user wants to travel from Slovany and it needs not be true. A flat concept parse can be adapted to work with shallow dependencies, but such a deep dependence would be out of its reach. Negative sentences represent the same problem. Here it is crucial to determine what is negated and what is not.

Secondly, the ability to create preterminal symbols with distinct model from making nested hierarchic semantic concepts makes it possible to avoid fragmenting the training data through arbitrary partitioning of attribute values. For example, in figure 3.7 the places are divided between arrival (ToPlace) and departure (FromPlace)[18]. That means that the model has in average only half of the examples to learn from. On the contrary, a hierarchic model links a Place to the Departure or Arrival concept and thus allowing the learning algorithm to learn the Place concept from all examples.

[18]The concept (FromPlace) is not shown in the figure.

At last but not at least, this model is able to capture long dependencies. The problem of long dependencies was discussed in 3.3.1 – HMM Flat Concept Parsing.

But, PCFGs also have their problems. In particular, they are significantly more complex compared to HMMs. Thus they are computationally more expensive, especially when implementing Inside-Outside algorithm (This algorithm computes the α[19] and β functions for all words in all training sentences for each iteration). PCFGs also suffer from normalization problems because the probabilities of utterances with complex parse trees consisting of many nodes are underestimated. It is because the probability of a smaller tree is higher than the probability of a larger tree (for more details please consult [MS01] – section 11.1).

3.3.3 Probabilistic Recursive Transition Networks

Probabilistic Recursive Transition Network (PRTN) model is another way of dealing with recursion that is strongly related to PCFG. In fact without any extension, these two formalisms are equivalent and have the same algorithms for training and decoding.

The recursive transition network (RTN) principle is based on FSA[20]. The difference between a RTN and a FSA lies in how the nonterminals are handled. In a RTN, every time the machine comes to an arc labeled with a nonterminal, it invokes a subroutine associated with that nonterminal. The subroutine invocation process places current state onto a stack, jumps to the nonterminal network and then jumps back when that nonterminal has been parsed.

The figure 3.12 shows parsing of the nonterminal A in the network S. During transition from state S to state S1 the current state (S) is placed onto a stack, the network A is activated and traversed from A to A3. Then the state S is taken from the stack and transition from S to S1 is completed. Of course, traversing from A to A3 involves jumping to network B and since network B contains self reference it can invoke itself repeatedly.

Probabilistic RTN (PRTN) adds a probability p to each transition from state S_a to state S_b.

$$P(S_b|S_a) = p,\ 0 \leq p \leq 1 \tag{3.28}$$

$$\sum_{S_i} P(S_i|S_a) = 1 \tag{3.29}$$

[19]The α function stands for the so called outside probability. It is defined similarly to the β function (see [MS01] – 11.3.2).

[20]Finite State Automaton, for definition see [All95] – section 3.5

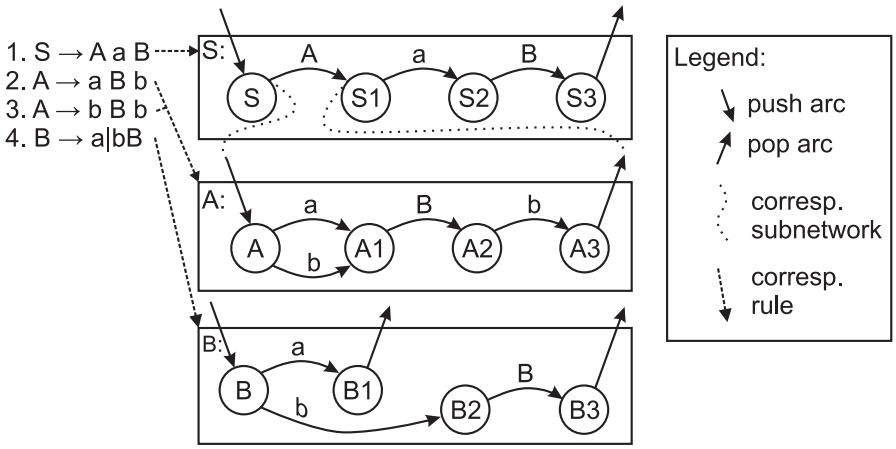

Figure 3.12. Recursive transition network model and its relation to a context-free grammar.

If we want to strengthen the predictive power of PRTN we can extend the context. For example we can take two preceding states into account. Then the probability of a transition is computed by formula $P(S_c|S_a, S_b)$. In fig. 3.12 the arc from B to B1 has probability $P(B1|B, A1)$ or $P(B1|B, S2)$ depending on the network that invoked B. Using this extension of PRTN, the algorithms for training and decoding have to be significantly modified.

As long as the subroutine invocation stack is unlimited the PRTNs have the same descriptive power as the PCFGs. Every PCFG can also be easily converted to a PRTN and vice versa. The conversion process gets slightly complicated when a kind of PRTN minimization is involved. The figure 3.12 shows a grammar and corresponding minimized PRTN. The minimization here consists in sharing states A1 – A3 by rules 2 and 3.

3.3.4 Vector-state Markov Model

The key feature of the 1st order Markov model (introduced in section 3.3.1) is that the current state at time t holds all of the information needed to account for the observation at time t and the transition to a new state at time $t + 1$. It is this property which gives the model its mathematical simplicity. The Vector-state Markov Model [You02b] improves the basic model by extension of the state space to record broader context. Every Markov state is extended to contain a vector of semantic concepts. The vector of concepts is utilized to capture the hierarchic structure of an utterance.

The information in a parse tree relating to any single word can be stored as a vector of node names starting from the preterminal and ending in the root node. For instance the state associated with the word “tram” (in fig.

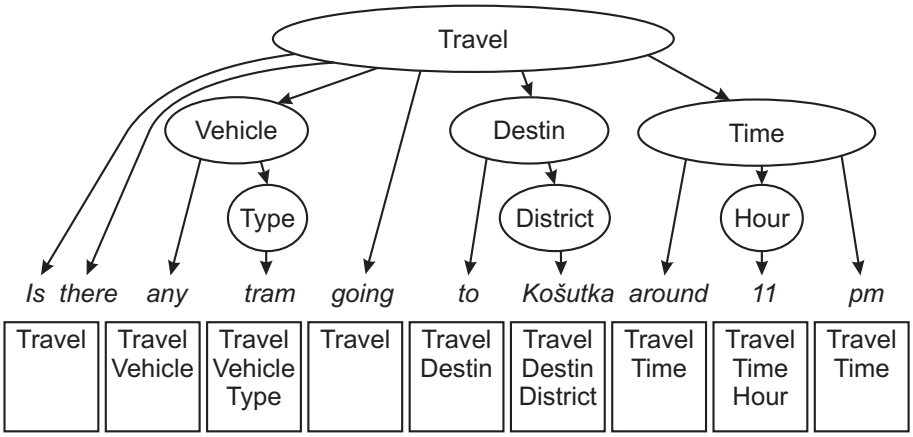

Figure 3.13. Example of a right branching tree and corresponding vector state sequence.

3.13) contains a semantic vector [Type,Vehicle,Travel]. Complete parse tree can be represented by a sequence of semantic vectors (see fig. 3.13). If the nodes in a parse tree are all distinct from each other, there is a one-to-one mapping between semantic vector sequences and a parse tree. The semantic vector has indeed the structure of a stack because new nodes are put at the end of the vector and the last nodes are taken first from the vector while parsing a sentence.

The parsing of a sentence (see fig. 3.13) works within Vector-state Markov model in the following way: the parser goes left to right, each time a word is processed, a transition based on current state is triggered. Every transition is constrained to take the form of stack shift (a number of nodes are removed) followed by a push of one or more nodes. Please note, when the transition has exactly the form of removing one node followed by pushing of one node, we get an ordinary 1^{st} order Markov model.

Given an unlimited stack, any PCFG formalism (see sec. 3.3.2) can be converted into a Vector-state Markov Model. The problem, however, is that in doing so the state space rapidly becomes huge. Thus, the form of transitions can be limited to alleviate the model complexity. The [You02b] proposes following transition limitation: a stack shift followed by pushing at most one node. The effect of this on parse tree of semantic concepts is to make it right branching. The example in fig. 3.13 was made in this manner.

The generative process (eq. 3.30) associated with this model consists of three steps for each word position t: (a) choose a value n_t which represents a number of removed nodes; (b) select a preterminal node c_{w_t} for word w_t; (c) select a word w_t. As with the PCFG model, the probability distribution $P(W, \mathbb{C}, N)$ can be decomposed either top-down or bottom-up. The top-down

decomposition is presented in 3.30:

$$P(W,\mathbb{C},N)=\prod_{t=1}^{T}\overbrace{P(n_t|W_1^{t-1},C_1^{t-1})}^{\text{(a)}}\overbrace{P(c_t[d_t]|W_1^{t-1},C_1^{t-1},n_t)}^{\text{(b)}}\underbrace{P(w_t|W_1^{t-1},C_1^{t})}_{\text{(c)}} \tag{3.30}$$

where $\mathbb{C}$ is matrix whose columns are vector stacks $\vec{c}$; d_t denotes the length of the stack $\vec{c}_t$ ($d_t = |\vec{c}_{t-1}| - n_t + 1$); $c_t[d_t]$ is a new preterminal symbol on the top of the stack $\vec{c}_t$.

In the version of the Vector-state Markov model discussed in [You02b], the components of eq. 3.30 are approximated by:

$$P(n_t|W_1^{t-1},C_1^{t-1}) \approx P(n_t|\vec{c}_{t-1}) \tag{3.31}$$

$$P(c_t[d_t]|W_1^{t-1},C_1^{t-1},n_t) \approx P(c_t[d_t]|c_t[1..d_t-1]) \tag{3.32}$$

$$P(w_t|W_1^{t-1},C_1^{t}) \approx P(w_t|\vec{c}_t) \tag{3.33}$$

where $c_t[1..d_t-1]$ denotes stack elements 1,2,..,d_t-1.

Then, by substitution 3.31 – 3.33 in 3.30 we get the 3.34. This formula is being used in practical applications.

$$P(W,\mathbb{C},N) \approx \prod_{t=1}^{T} P(n_t|\vec{c}_{t-1})P(c_t[d_t]|c_t[1..d_t-1])P(w_t|\vec{c}_t) \tag{3.34}$$

The training of Vector-state Markov model is analogous to the training of the HMM or the PCFG. The supervised training consists in events counting, normalization and smoothing. The fully unsupervised training is again impossible and partially unsupervised training has to be used. An excellent example of partially unsupervised training was proposed and tested in [HY05] and is discussed in section 3.3.7.

Unlike the PCFG model, this probabilistic model is well-suited to left-right decoding. Since each partial path covering word sequence W_1^t contains exactly the same number of probabilities, paths can be compared directly without normalization.

The Vector-state Markov model extends the flat-concept HMM model by expanding each state to encode the stack of a push-down automaton. This allows the model to encode hierarchical context. This model stands in between the HMM and PCFG model. With the limited stack it covers regular languages only. However, by extending the capacity of the stack it approaches the context-free languages. This model was proposed to be simple

and thus reliable and to cover natural language at the same time. The formerly described transition limitation covers right-branching languages only. The [HY05] claims that majority of English sentences is right-branching. Still, it is question whether Czech utterances are also mainly right branching.

3.3.5 Model training

We have two basic options for stochastic model training. It is

- supervised
- unsupervised

training. It is clear from previous sections that fully unsupervised training is impossible for a nontrivial domain. Although the supervised training is simple and makes the model to be robustly trained, it is too costly for a direct usage. It seems to be wise to use the golden mean. That means that instead of fully unsupervised training partially unsupervised training is used. The cost of supervised training can be alleviated by a technique called bootstrapping.

The unsupervised training is usually solved by a variant of EM algorithm. The particular algorithm is called Baum-Welch or Forward-Backward[21] algorithm for the HMM model training and Inside-Outside algorithm for a PCFG model. There are two key moments in a partially unsupervised training. Firstly, the model should be initially trained on a small amount of manually annotated data. Secondly, a minimal annotation has to be still present during the unsupervised training process. Such a model is then capable of producing the same form of minimal annotation that it was trained from. We can not expect full parse tree to be produced.

A major problem with supervised approaches is the need for a large annotated training set. The bootstrapping approach decreases the effort needed for annotation by an incremental building of a training set. It starts with an initial small training set created by an annotator from scratch. The initial training set is then used to train an initial parser using any of the supervised learning methods mentioned in the previous sections. This initial parser is then used to extract a larger training set from a part of the remaining unannotated training set. The extracted training set needs to be corrected by hand. The new larger training set is then used to train a better parser and so on. This process continues until the desired amount of data is annotated. It is supposed that the human effort needed for correction is lower than the

[21] Baum-Welch and Forward-Backward are synonyms for the same algorithm

effort needed for building a training set from scratch. This becomes true while the training set is growing larger and the parser is improving. There is a few alternatives that reduce the human effort. A human annotator need not always correct the automatic parses. He or she may for instance decide whether the parse is correct or not and only correct parses proceed to next training iteration. Or, the parser itself can select which parse it has the biggest confidence in. Again, only these parses proceed to next iteration. Then the training is fully automatic.

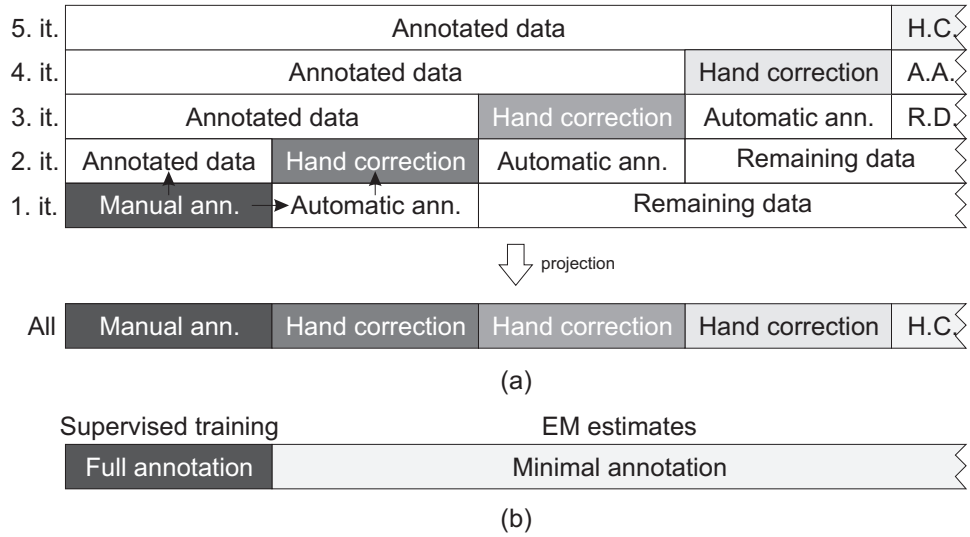

Figure 3.14. Comparison of (a) the bootstrapping aided supervised training and (b) partially unsupervised training. The gray intensity shows the annotator effort needed.

The partially unsupervised training estimates parameters of a model with the lowest possible human effort cost. The bootstrapping technique significantly lowers the costs of the supervised training. But, the costs of the supervised training with the bootstrapping are higher than the costs of the partially unsupervised training (see the projection in fig. 3.14). However, the model trained with the supervised training produces full parse trees. Therefore, if the parse tree is not needed, unsupervised training should be preferred and vice versa.

3.3.6 Evaluation

An appropriate evaluation metric is crucial for every parsing model because it measures the effect of an alternation in a parser. A right evaluation metric can help us to accept useful alternations and to deny ineffective ones.

We are not interested in semantic analysis for its own sake. Thus, in principle the best way to evaluate semantic parsers is to embed them in a complete system and to investigate the differences that the various parsers make in a task-based evaluation. However, a desire for simplicity and modularization often means that it would be better to have metrics on which a parser can by easily evaluated without the necessity of evaluation of a complete system. So we try to create a parser evaluation which would hopefully reflect the performance of the complete system. The evaluation is mostly based on the comparison between the result of our system and the result of the hand annotation, which we regard as a gold standard.

One of the simplest metrics is the exact match criterion. The exact match criterion is a binary function

$$\delta(A_t = H_t) = \begin{cases} 1 & \text{if } A_t \text{ is exactly equal to } H_t \\ 0 & \text{otherwise} \end{cases} \tag{3.35}$$

that awards the parser 1 point iff the parser output A_t is exactly equal the hand annotation H_t for the t-th testing utterance.

The exact math criterion is, however, sometimes too strict and we need a more fine-grained criterion. Such a criterion should award the parser a number in the $\langle 0, 1\rangle$ interval according to how the computer results match the human results.

We use the *error rate* and *accuracy* metric if we compare single events (e.g. semantic labels in a Flat concept parsing technique) and the parser produces only one answer. The *recall* and *precision* metric can be used whenever the parser produces more answers (or no answer) for a word in an utterance. And the *PARSEVAL* metric is used for comparing parse trees.

For the error rate - accuracy and the recall - precision metrics we define following functions:

- $Out(w_t)$ - set of output answers for the word w_t
- $True(w_t)$ - single correct answer
- $Errors(S) = \sum_{t=1}^{T} \delta(Out(w_t) \neq True(w_t))$ - number of mistakes made by a single-output parser on the test set S
- $Correct(S) = \sum_{t=1}^{T} \delta(True(w_t) \in Out(w_t))$ - number of events, where the multiple-output parser produced at least one correct answer
- $Generated(S) = \sum_{t=1}^{T} |Out(w_t)|$ - number of generated answers

where S is the test set and $T = |S|$ is the length of the test set.

The error rate (Err) expresses the percentage of wrong answers per word and the accuracy (Acc) the percentage of correct answers for a single output parser. The measures are formally defined in 3.36 and 3.37:

$$Err(S) = \frac{Errors(S)}{|S|} \tag{3.36}$$

$$Acc(S) = \frac{|S| - Errors(S)}{|S|} = 1 - Err(S) \tag{3.37}$$

The recall (equation 3.38) shows how many correct answers the multiple-output system was able to reveal per word. It is easy to develop a system having the precision of one. Such a system can easily assign all possible labels or tags to each word. But, such a system would have very low precision (equation 3.39) measure. Because, the precision measures how many of the answers that the system returned is actually correct (a lot of incorrect answers means a low precision). It means that recall and precision are opposing to one another since a system that tries to maximize precision will lower its recall score and vice versa. A proper parser should balance both recall and precision. For a direct system comparison, the precision and recall can be merged into one value called the *F-measure* (eq. 3.40). The F-measure is different to the mean in terms of disqualifying extreme systems (system with very high precision and very low recall and vice versa). Therefore, it also expresses the system ability to balance both recall and precision.

$$R(S) = \frac{Correct(S)}{|S|} \tag{3.38}$$

$$P(S) = \frac{Correct(S)}{Generated(S)} \tag{3.39}$$

$$F(S) = \left(\frac{\alpha}{P(S)} + \frac{1-\alpha}{R(S)} \right)^{-1} \tag{3.40}$$

where the parameter $\alpha \in < 0, 1 >$ is used to distribute the preference between the recall and the precision. When α is greater than 0.5 the precision is favored, when α is less than 0.5 the recall is favored and when $\alpha = 0.5$ the precision and recall are given equal weight.

The PARSEVAL evaluation (described in [MS01]) is used for comparing parse trees with a hand-made gold standard annotation. Every tree can be represented in a bracketing notation where the brackets denote the span of a subtree. For example, the bracketing notation of the tree in the figure

3.13 is: TRAVEL-(Is there VEHICLE-(any TYPE-(tram)) going DESTIN-(to DISTRICT-(Košutka)) TIME-(around HOUR-(11) pm)).

Then in PARSEVAL evaluation, three basic measures are proposed: *precision* is the number of brackets in the parse tree matching those in the correct tree, *recall* measures the number of the brackets in the correct tree that are in the parse tree, and *crossing brackets* gives the average number of constituents in the tree that cross over constituent boundaries in the other tree.

The following example (3.15) helps to understand the PARSEVAL metric. Let 3.15-a be the gold standard parse tree and the 3.15-b be the candidate parse produced by a parser in the bracketing notation. Then we start by extracting the ranges which are spanned by subtrees. The ranges for gold standard and candidate parse tree are represented by GS (3.15-c) and CP (3.15-d), respectively. The ranges are compared and the resulting intersection of sets GS and CP is shown in $INT1$ (3.15-e) and in $INT2$ (3.15-f). The intersection $INT1$ ignores the node labels. The intersection $INT2$ to the contrary considers the node labels. The PARSEVAL measure computation is finally shown in 3.15-g. The NC denotes the set of normalized constituents (the PARSEVAL measure performs various ad hoc tree normalization, e.g. ignoring unary branching nodes, etc).

The PARSEVAL measure is widely used, however, it suffers from many problems: it is not very discriminative (especially for flat trees), it treats the same parser mistake differently in different conditions, etc. The more comprehensive discussion about PARSEVAL problems is presented in [MS01]. The parser evaluation problematic is discussed on a specialized workshop[22] hosted by LREC[23] conference. Although a number of evaluation metrics for parsers was proposed, no improved evaluation metric is yet taken as a new standard.

3.3.7 Existing Systems

A variety of semantic analysis systems based on stochastic approach was proposed and created during last two decades. Most of them use one of the theoretical approaches presented in sections 3.3.1 – 3.3.4. The following sections introduce some of the stochastic-based systems which illustrate the theoretical models in practise.

[22]The last one was called Beyond PARSEVAL – Towards Improved Evaluation Measures for Parsing Systems.

[23]International Conference on Language Resources and Evaluation.

(a) Gold standard tree in the bracketing notation:
TRAVEL-(Is there VEHICLE-(any TYPE-(tram)) going DESTIN-(to DISTRICT-(Košutka)) TIME-(around HOUR-(11) pm))

(b) Candidate parse tree in the bracketing notation:
TRAVEL-(Is there VEHICLE-(any TYPE-(tram)) going DESTIN-(to DISTRICT-(Košutka)) TIME-(around MINUTE-(11))) _-(pm)

(c) The span ranges for gold standard tree:
GS = {TRAVEL-(1:11), VEHICLE-(3:5), TYPE-(4:5), DESTIN-(6:8), DISTRICT-(7:8), TIME-(8:11), HOUR-(9:10) }

(d) The span ranges for candidate parse tree:
CP = {TRAVEL-(1:10), VEHICLE-(3:5), TYPE-(4:5), DESTIN-(6:8), DISTRICT-(7:8), TIME-(8:10), MINUTE-(9:10), _-(10:11) }

(e) The intersection without regard to node labels:
$INT1$= { VEHICLE-(3:5), TYPE-(4:5), DESTIN-(6:8), DISTRICT-(7:8), HOUR-(9:10) }

(f) The intersection with respect to node labels:
$INT2$ = {VEHICLE-(3:5), TYPE-(4:5), DESTIN-(6:8), DISTRICT-(7:8) }

(g) PARSEVAL measure computation:

Precision:	$P = \frac{\lvert INT1\rvert}{\lvert GS\rvert} = \frac{5}{7} \doteq$	71.4%
Recall:	$R = \frac{\lvert INT1\rvert}{\lvert CP\rvert} = \frac{5}{8} =$	62.5%
Labeled Precision:	$LP = \frac{\lvert INT2\rvert}{\lvert GS\rvert} = \frac{4}{7} \doteq$	57.1%
Labeled Recall:	$LR = \frac{\lvert INT2\rvert}{\lvert CP\rvert} = \frac{4}{8} =$	50.0%
Crossing brackets:	$CB =$	1
Crossing accuracy:	$CA = 1 - \frac{CB}{\lvert NC\rvert} = \frac{3}{4} =$	75.0%

Figure 3.15. The PARSEVAL metric example

Chronus

An early statistical approach to semantic tagging is used in the CHRONUS[24] system [Pie92]. In the CHRONUS knowledge representation, each unit of meaning consists of a pair $m_j = (c_j, v_j)$, where c_j is a conceptual relation (e.g. *origin*, *destination*, *meal* in the ATIS [Pri90] flight domain), and v_j is

[24]Conceptual Hidden Representation Of Natural Unconstrained Speech

the value with which c_j is instantiated in the actual sentence (e.g. *Boston*, *San Francisco*, *breakfast*).

The semantic processing in the CHRONUS system is composed of following main components:

- Lexical parser
- Conceptual decoder
- Template generator

The *lexical parser* generates a lattice with all possible interpretations of a string (e.g. the substring "b seven four seven" could be interpreted as "B 747" or "B7 47" or "B74 7", etc). The preprocessing is task dependent and reduces the model size.

The *conceptual decoder* provides a conceptual segmentation given the lattice generated by the lexical parser. This conceptual segmentation maps a sequence of words $(w_{I_j}, ..., w_{I_j+N_j})$ from the input utterance into a concept c_j:

$$S : (w_{I_j}, ..., w_{I_j+N_j}) \rightarrow c_j, \forall j \tag{3.41}$$

The conceptual decoder is modeled by a Markov process. The Viterbi based decoder is generalized to work on a word lattice. The concept conditional model is smoothed via back-off smoothing. The conceptual decoder is similar to the parser described in the section 3.3.1 – HMM Flat Concept Parsing. It is capable of being robustly trained, but it is not capable of capturing hierarchical structures.

The *template generator* consists of a simple pattern matching procedure that, given the conceptual segmentation, produces for each concept relation c_j the corresponding concept value v_j.

The training of this model is supervised. The training is improved by a process similar to bootstrapping (see sec. 3.3.5). The training starts with a hand-made initial training set. A basic system is trained from the initial training set. The basic system is a complete system that is able to generate answers to questions. The remaining unannotated corpus is then processed with the basic system and for every question an answer is generated. The training corpus is annotated with right answers and the system can compare its output with the correct answers. Only the utterances, which the system answered correctly, are added to the training set with their automatic annotation. Then, a new model is trained on the new larger training set. This process is iterated until no new utterance can be added to the training set.

The semi-automatic training process is finished by hand-labeling of remaining utterances (which the system was not able to annotate automatically).

Although the CHRONUS system is old, it is an elegant system that uses several practical points: a preprocessing to reduce model size, the variant of the bootstrapping training to reduce annotation costs and a simple model (HMM) for robust training.

Hidden Understanding Model

The [Mil94] introduces the Hidden Understanding Model (HUM). The word "hidden" refers to the fact that only words can be observed. The internal states of the model are hidden and have to be derived from a word sequence.

In the HUM, the utterance is modeled by two statistical models:

- Semantic language model
- Lexical realization model

The semantic language model chooses the meaning to be expressed, effectively deciding "what to say".

The lexical realization model generates words sequences once a meaning is given, effectively deciding "how to say it".

Both these models are treated as a PRTN (see 3.3.3). The problem of understanding is then to find the highest probability path among all possible paths in a PRTN:

$$P(\hat{C}) = \arg\max_{C} \prod_{t=1}^{T} \begin{cases} P(state_n|state_{n-1}, H_n) \text{ for the semantic lang. model} \\ P(word_n|state_{n-1}, H_n) \text{ for the lexical realiz. model} \end{cases} \tag{3.42}$$

where H_n is the context (History) of a $state_n$ or a $word_n$. The context can be for example $state_{n-2}$, etc.

Effective search among all possible paths is performed by a modification of Viterbi algorithm [Mil94]. Since the underlying model is a recursive transition network, the states for Viterbi search must be allocated dynamically as the search proceeds. In addition, it is necessary to prune low probability paths in order to keep the algorithm computationally feasible.

Both semantic and lexical model are trained in supervised manner. Transitions within the PRTN representing the data (the [Mil94] shows how to transform data to a PRTN) are counted and normalized. Robustness of the statistical model is improved through back-off smoothing and lexical classes

definition (see following section Hidden Vector State Model for the explanation of lexical classes). A bootstrapping learning method is introduced in [Sch96] to lower the expenses of training.

The Hidden Understanding Model is further improved in [Mil96] and in [Sch96] through incorporating the syntactic information within the model. Other models besides the semantic parsing model are defined in [Mil96] and [Sch96] to make the whole utterance processing statistical. The utterance is then processed in sequence by the semantic parsing model, the semantic interpretation model and the discourse processing model.

Hidden Vector State Model

The Hidden Vector-state Markov (HVS) model is described in [You02b] and in [HY05]. The performance of the model is tested with ATIS [Pri90] corpus and with DARPA Communication Task (in [HY05]). The HVS model is in fact the Vector-state Markov model (see section 3.3.4) that is trained by a partially unsupervised learning method. The authors claim that the provision of fully annotated data is not realistic in practice. Training from partially annotated corpora is therefore crucial for practical applications. Partially unsupervised training requires some priory knowledge about a domain.

The priori knowledge consists of the two following parts:

- A set of domain specific lexical classes.
- Abstract semantic annotation for each utterance.

Lexical classes typically group proper names into hypernym[25] class. E.g., in a flight information system, typical classes might be CITY = {Boston, Denver, New York, ...}, AIRPORTS = {Dulles, ...}, etc. These domain specific classes can be usually extracted from the domain database schema. The generic classes (covering times, dates, etc.) may be also included among domain specific classes. Given these classes, all words in the training set are replaced with corresponding lexical class when possible. This reduces the size of the model.

The *abstract semantic annotation* lists a set of applicable semantic concepts with the dominance relationship between them for each training utterance. However, it does not take any account of word order or attempt to annotate every part of the utterance. Every particular utterance that has the

[25]hypernym relates more general term with a lexeme, e.g. vehicle is hypernym of car, for more detail see [JM00] – chapter 16

same database SQL query should have the same abstract semantic annotation (for example, see figure 3.16). This is one of the requirements presented in the section 3.1 which is called Canonical Form.

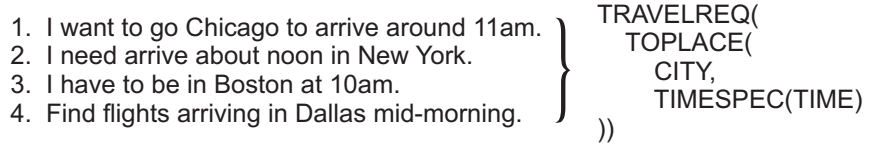

Figure 3.16. An abstract annotation of utterances carrying the same meaning.

The training of this model is defined in [HY05] and involves *preprocessing*, *parameter initialization* and *parameter re-estimation*.

The preprocessing prepares the data for training. The [HY05] uses so-called *flat-start* whereby all model parameters are initially made identical.

The parameter re-estimation is defined in [HY05] as follows: Let the complete set of model parameters be denoted by λ, EM-based parameter estimation aims to maximize the expectation of $L(\lambda) = \log P(N, \mathbb{C}, W|\lambda)$ given the observed data and current estimates. To do this, the 3.43 expression has to be maximized with respect to $\hat{\lambda}$:

$$\sum_{N,\mathbb{C}} P(N, \mathbb{C}|W, \lambda) \log P(N, \mathbb{C}, W|\hat{\lambda}) \tag{3.43}$$

Substituting eq. 3.34 into eq. 3.43 and differentiation lead to the following re-estimation formulae:

$$P(n_t|\vec{c}) = \frac{\sum_t P(n_t = n, \vec{c}_{t-1} = \vec{c}|W, \lambda)}{\sum_t P(\vec{c}_{t-1} = \vec{c}|W, \lambda)} \tag{3.44}$$

$$P(c_t[d_t]|c_t[1..d_t - 1]) = \frac{\sum_t P(\vec{c}_t = \vec{c}|W, \lambda)}{\sum_t P(c_t[1..d_t - 1] = c[1..d_t - 1]|W, \lambda)} \tag{3.45}$$

$$P(w_t|\vec{c}_t) = \frac{\sum_t P(\vec{c}_t = \vec{c}|W, \lambda)\delta(w_t = w)}{\sum_t P(\vec{c}_t = \vec{c}|W, \lambda)} \tag{3.46}$$

where $\delta(w_t = w)$ is one iff the word at time t is w, otherwise it is zero.

The key components of the above re-estimation formulae are the likelihoods $P(n_t = n, \vec{c}_{t-1} = \vec{c}|W, \lambda)$, $P(\vec{c}_t = \vec{c}|W, \lambda)$ and $P(c_t[1..d_t - 1] = c[1..d_t - 1]|W, \lambda)$. These likelihoods can be efficiently calculated using the forward-backward algorithm. The formulae for the forward and backward probabilities (α, β) are defined in [You02b] in detail.

The [You02b] also defines the formulae for model adaptation. Regardless of how much training data is available, a semantic decoder will perform

Task	ATIS	DARPA
Recall	89.82%	87.31%
Precision	88.75%	88.84%
F-measure	89.28%	88.07%

Table 3.3. The performance of HVS model on ATIS and DARPA data

badly if the training data does not well-represent the test data. The model adaptation can be used to increase the performance of semantic decoder.

The experiments conducted on ATIS corpus and DARPA Communicator Travel Data in [HY05] are presented in table 3.3. The direct comparison of the Flat Concept HMM model and the HVS model is furthermore contained in [HY05]. The results show HVS model perform significantly better then a HMM model in this area.

The HVS model solves the problem of the need to collect large quantity of fully annotated tree-bank data. It proves that the Vector-state Markov model is constrained enough to be trained by partially unsupervised training whilst at the same time it is able to capture hierarchical dependencies.

On the other hand, the unsupervised training prohibits the model to produce a full parse tree which could be a problem if we need the parse tree for contextual interpretation.

Other systems

The above mentioned systems do not represent all significant stochastic systems recently created. They were chosen to demonstrate described models in practise. There are other parsing systems that include for example the system described in [Min99] which uses Hidden Markov models with structured semantic tags to perform semantic analysis. The [Pra04] uses support vector machines, a powerful classifier, to perform shallow semantic analysis.

The authors of [CJ00] try to solve the problem of long dependencies by introducing a new type of language model capable of capturing hierarchical structures in binary branching trees. This model is then used in [CM01] to create an information extraction system. The structured language model is in [CM01] extended to include both syntactic and semantic information in non-terminal nodes.

The Prague Dependency Treebank 2.0 (PDT 2.0) [Haj04], which is being developed by the Institute of Formal and Applied Linguistics at the Charles University would deserve its own chapter. But the extent of this work is limited and the semantic within the PDT 2.0 is not fully developed yet. Moreover the PDT 2.0 deals mainly with written language. Therefore, this

approach is described only shortly in this thesis. The language in PDT 2.0 is described by three main layers: Morphological, Analytical (syntax) and Tectogrammatical (syntax–semantic). Morphological layer describes morphology of words by structured tags that have 15 positions (every position relates to a morphological category). The syntax description within this approach covers a variety of syntactic phenomena (coordination, subcategorization, gaps and so on). The tectogramatic layer stands in between the syntactic and semantic description of a sentence. It describes semantic roles of constituents together with the syntactic roles. A semantic layer will describe the semantic roles only but this layer has not been developed yet. The research into this approach is targeted at inflectional languages, mainly at Czech language. This approach has the ambition to cover the general language. But the broad coverage is the main advantage and drawback of this approach at the same time. The development is very slow and many theoretical problems have not been solved yet. Anyway, the research in this field yields many important pieces of knowledge in the natural language processing.

Since the semantic analysis is an important part of every NLU system, there is a plentiful amount of other semantic analysis systems that were not discussed in this thesis. The authors of [MW99] give a comprehensive list of semantic analysis projects and various natural language corpora.

Chapter 4

Conclusions and Future Work

Stochastic systems have several advantages when compared to systems based on expert knowledge. For example, in a system based on expert knowledge, human must provide the sematic system with the exact instructions for utterance parsing. Whereas, in the stochastic approach, human gives the computer just the examples of the parsing for training utterances. Stochastic systems infer parsing instructions by their own algorithms. The provision of examples is much simpler and therefore less expensive than the provision of exact parsing instructions. A less expensive development is one of the key advantages of stochastic systems.

Significant portion of [MW99] deals with the portability of semantic analysis system. It arises from the experiments conducted in [MW99] that porting[1] a stochastic system is easier than porting a rule based system. The portability is another key advantage of stochastic systems.

This thesis is a theoretical preparation for the development of semantic analysis system that will serve in the project of City Information Dialogue (CID) system [Mou04]. The analysis of a domain and the definition of concept hierarchy have been done in [Mou04].

However, the main task is not to develop a dedicated parsing system for CID domain. The primary goal of the future work is to develop a semantic analysis system that will be capable of analyzing any domain. The process of porting the system should consist merely in the provision of other training data. Of course, a little tuning (preprocessing alternation, training constants tuning) is inevitable during the process of porting.

Such systems currently exist (see section 3.3.7). The contribution of this work should be to develop a system capable of working with a language that

[1] The term *porting* stands for the process of adapting a system to different domain than it was originally developed for.

is inflectionally rich and has a relatively free word order (e.g. Czech language and other languages from the Slavic language family).

The ultimate solution of the semantic analysis problem is to develop a universal system that would not be restricted to a domain. Such a system would be able to analyze any utterance. There are, however, two principal problems of such a system: representation and ambiguity. There is currently no domain independent and universal semantic representation formalism. Moreover, ambiguity in the unrestricted domain would be unmanageable.

4.1 Aims of Doctoral Thesis

1. Continue with the work in [Mou04] and develop a suitable formalism for the CID domain description.

2. Assemble a team of annotators and lead them during annotation of CID domain utterances. Perform various statistical measures on the annotated corpus.

3. Choose a suitable algorithm for stochastic semantic parsing and propose alternations that enable the algorithm to work with language that is inflectionally rich and has a relatively free word order (e.g. Czech language and other Slavic languages).

4. Evaluate the performance of all the alternatives proposed and implemented during the development of the semantic parsing system.

Appendix A

Definitions of Formal Models

This chapter gives the formal definition of all the models used in this thesis. Besides the model definition, the sections refer to the solution of three basic questions. The questions are:

1. Given the sequence of generated symbols $Y = (y_1, ..., y_T)$ and a model[1] λ, how do we efficiently compute $P(Y|\lambda)$, the probability of the sequence Y, given the model?

2. Given the generated symbols Y, and the model λ, how do we find the sequence of states from which Y is most likely generated?

3. How do we adjust the model parameters λ to maximize the probability of training data?

A.1 Hidden Markov Models

A Markov model is a probabilistic system that can be described at any time as being in one of a set of N distinct states, $s_0, s_2, ..., s_N$. Every discrete time instance, the current state emits one output symbol from the output alphabet. A Hidden Markov Model (HMM) is a Markov Model, where the sequence of states that the model passes through can not be directly observed and is hidden.

Markov models have two properties. Suppose $X = (X_1, ...X_T)$ is a sequence of random variables taking values in some finite set $S = \{s_1, ...s_N\}$ (the state space). Then, the *Markov* properties are:

[1]The model consists of model parameters.

- Limited horizon

$$P(X_t = s_t|X_{t-1} = s_{t-1},..., X_1 = s_1) = \\ P(X_t = s_t|X_{t-1} = s_{t-1}) \quad \text{(A.1)}$$

- Time invariant (stationary)

$$P(X_t = s_k|X_{t-1} = s_r) = P(X_u = s_k|X_{u-1} = s_r) \quad \text{(A.2)}$$

The *limited horizon* property means that the probability distribution of being in state s_k depends only on limited history. In our case the probability distribution depends only on one previous state. The model is then called 1^{st} order Markov model. When the model depends on k previous states (A.3) the model is called k^{th} order Markov model.

$$P(X_j = s_t|X_{j-1} = s_{t-1},..., X_1 = s_1) = \\ P(X_j = s_t|X_{j-1} = s_{t-1}, ..., X_{t-k} = s_{n-k}) \quad \text{(A.3)}$$

The *time invariant* property says that the distribution of transitions between states does not change over time.

HMM is formally a five-tuple (S, s_0, Y, P_S, P_Y), where

- $S = \{s_0, s_1, ..., s_N\}$ is the set of states,
- s_0 is the initial state,
- $Y = \{y_1, y_2, ..., y_M\}$ is the output alphabet,
- P_S is the set of probability distributions of transitions from a state to a state,
- P_Y is the set of output (emission) probability distributions.

The solution to the basic questions (see the introduction of this chapter) is explained e.g. in [Rab89]. Question 1 is solved by the *forward* or *backward* procedure. The algorithm that solves question 2 is usually called *Viterbi*. The algorithm that solves question 3 is called *Baum-Welch* or *Forward-Backward reestimation* algorithm.

A.2 Context Free Grammar

Context Free Grammar (CFG) is a 4-tuple $G = (N, \Sigma, R, S)$, where

- N is a set of non-terminal symbols (or "variables"). The non-terminals are denoted by uppercase letters,
- Σ is a set of terminal symbols (disjoint from N). The terminals are denoted by lowercase letters,
- R is a set of rules, each of the from $A \to \alpha$, where $A \in N$ is a non-terminal and α is a string of symbols from the infinite set of strings $(\Sigma \cup N)^*$,
- S is the start symbol.

A *string*, denoted by a small Greek letter, is a sequence of terminals and non-terminals $(\Sigma \cup N)^*$. One string *derives* another one if it can be rewritten as the second one via some series of rules application. More formally, if $A \to \beta$ is a rule and α and γ are any strings, then we say that $\alpha A \gamma$ *directly derives* $\alpha\beta\gamma$, or $\alpha A \gamma \Rightarrow \alpha\beta\gamma$. Derivation $\alpha_1 \overset{*}{\Rightarrow} \alpha_m$ then consists of a sequence of direct derivations $\alpha_1 \Rightarrow \alpha_2, \alpha_2 \Rightarrow \alpha_3, ..., \alpha_{m-1} \Rightarrow \alpha_m$. A *formal language* $\mathcal{L}_G$ generated by a grammar G is a set of strings W composed of terminals which can be derived from the start symbol S:

$$\mathcal{L}_G = \{W : w_i \in \Sigma^* \text{ and } S \overset{*}{\Rightarrow} w_i\} \tag{A.4}$$

Parsing of a string $\omega \in \Sigma^*$ is a process whereby an algorithm tries to find the sequence of rules that are needed to derive string ω from the start symbol S. A *parse tree* is a tree that shows the rules application hierarchy.

A.3 Probabilistic Context Free Grammar

Probabilistic Context Free Grammar (PCFG) is a 5-tuple $G = (N, \Sigma, R, S, D)$, where

- N is a set of non-terminal symbols (or "variables"),
- Σ is a set of terminal symbols (disjoint from N),
- R is a set of rules, each of the from $A \to \alpha$, where $A \in N$ is a non-terminal and α is a string of symbols from the infinite set of strings $(\Sigma \cup N)^*$,

- S is the start symbol,
- D is a function assigning probabilities to each rule in R.

To define the probability, several independence assumptions are made. The assumptions are illustrated by examples that use the figure A.1. Examples contained in explanations examine the independency of the probability $P(A \rightarrow \alpha)$ of the rule $A \rightarrow \alpha$ where $\alpha \overset{*}{\Rightarrow} w_a...w_b$. The independence assumptions are:

- *Place invariance*: The probability of a subtree does not depend on where in the string the words it dominates are (this is like time invariance in HMMs). In figure A.1 the probability $P(A \rightarrow \alpha)$ does not depend on the positions a, b in the string.
- *Independence of context*: The probability of a subtree does not depend on neighboring sub-trees. Thus, $P(A \rightarrow \alpha|C, D) = P(A \rightarrow \alpha)$.
- *Independence of ancestors*: The probability of a subtree does not depend on the parent (upper) node. Thus, $P(A \rightarrow \alpha|B) = P(A \rightarrow \alpha)$.

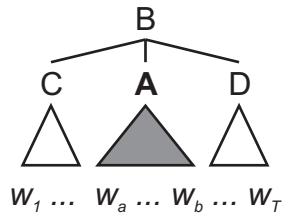

Figure A.1. Illustration of independence assumtion.

The probability that a given non-terminal A is expanded to the string α depends only on the non-terminal A:

$$P(A \rightarrow \alpha|A) \tag{A.5}$$

The probability is defined in such a way that the equation A.6 holds true for all non-terminals.

$$\sum_i P(A \rightarrow \alpha_i|A) = 1 \text{ and } 0 \leq P(A \rightarrow \alpha_i|A) \leq 1 : \forall A \in N \tag{A.6}$$

where α_i are all the right hand sides of the rule with the left hand side A.

The solution to the basic questions (see the introduction of this chapter) is explained e.g. in [MS01] – chapter 11. Question 1 is solved by the definition of inside (so called β function) and outside probability (so called α function). The algorithm that solves question 2 is a *probabilistic parser* (see sec. 3.3.2). The algorithm that solves question 3 is called *Inside-Outside* algorithm.

A.4 Recursive Transition Networks

A Probabilistic Recursive Transition Network (PRTN) is a 4-tuple $(\mathcal{A}, \mathcal{B}, \Gamma, \Xi)$, where

- $\mathcal{A}$ is a transition matrix containing transition probabilities,
- $\mathcal{B}$ is an output matrix containing probability distribution of the output symbols at each terminal transition. In the matrix, row and column correspond to terminal transitions and list of words, respectively;
- Γ specifies types of transitions (see further),
- Ξ denotes an invocation stack.

According to the stack operation, transitions are classified into three types:

1. *Push* – current state is pushed onto the stack Ξ.
2. *Pop* – the top most state is taken from the stack.
3. Transition not committed to a stack operation.

The reference to the solution of three basic questions follows. The question 1 is solved by the definition of inside and outside probability, similarly to the PCFG approach [HC94]. The question 2 can be solved by a generalized Viterbi algorithm [Mil94]. The algorithm that solves question 3 uses inside and outside probabilities [HC94].

Bibliography

[All95] Allen, J.: *Natural Language Understanding.* Benjamin/Cummings Publ. Comp. Inc., Redwood City, California, 1995.

[Ben03] Beneš, V.: Sémantická analýza doménově roztříděných dialogů (in Czech), Thesis, Plzeň, Czech Republic, 2003.

[Bri95] Brill, E.: Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging, *Computational Linguistic, Volume 21*(4), 1995, 543-565.

[CJ00] Chelba, C., Jelinek, F.: Structured Language Modeling, *Computer Speech and Language, Volume 14* (4), 2000, 283-332.

[CM01] Chelba, C., Mahajan, M.: Information Extraction Using the Structured Language Model, *In Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2001.

[Col03] Collins, M.: Head-Driven Statistical Models for Natural Language Parsing, *Computational Linguistics, Volume 29*, Part 4, 2003, 589-638.

[Dem77] Dempster, A. P., Laird, N. M., Rubin, D. B. : Maximum Likelihood from imcomplete data via the EM algorithm, *Journal of the Royal Statistical Society 39*(1), 1997, 1-38.

[Ear70] Earley, J.: An efficient context-free parsing algorithm. *Communications of the ACM. Volume 13*, Issue 2 1970, 94-102.

[Fel98] Fellbaum, C.: *WordNet, an electronic lexical database.* MIT Press, Cambridge, 1998.

[Fin98] Fine, S., Singer, Y., Tishby, N.: The Hierarchical Hidden Markov Model: Analysis and Applications, *Machine Learning, Volume 32*, Issue 1, 1998, 41-62.

[Haj04] Hajič, J.: *Complex Corpus Annotation: The Prague Dependency Treebank.* Jazykovedný ústav Ľ. Štúra, SAV, Bratislava, Slovakia, 2004.

[HC94] Han, Y. S., Choi, K-S.: A Reestimation Algorithm for Probabilistic Recursire Transition Network. *In Proc. of the 15th conference on Computational linguistics, Volume 2*, 1994, 859 - 864.

[HH98] Hajič, J., Hladká, B.: Tagging Inflective Languages: Prediction of Morphological Categories for a Rich, Structured Tagset, *In Proc. of the 17^{th} international conference on Computational linguistics*, 1998.

[HY05] He, Y., Young, S.: Semantic processing using the Hidden Vector State model. *Computer Speech and Language, Volume 19*, Issue 1, 2005, 85-106.

[JM00] Jurafsky, D., Martin, J.: *Speech and Language Processing.* Prentice Hall, 2000.

[Mar93] Marcus, M. P., Santorini, B., Marcinkiewicz, M. A.: Building a Large Annotated Corpus of English: The Penn Treebank, *Computational Linguistics, Volume 19*(2), 1993, 313-330.

[Mil94] Miller, S., Schwartz, R., Bobrow, R., Ingria, R.: Statistical Language Processing Using Hidden Understanding Models. *Proc. ARPA Speech and Natural Language Workshop*, 1994, 278–282.

[Mil96] Miller, S., Stallard, D., Bobrow, R., Schwartz., R.: A Fully Statistical Approach To Natural Language Interfaces. *In Proc. of the 34th Annual Meeting of the Association for Computational Linguistics*, 1996.

[MW99] Minker, W., Waibel, A., *Mariani, J.: Stochastically-based Semantic analysis.* Kluwer Academic Publishers, 1999.

[Min99] Minker, W., Gavalda, M., Waibel, A.: Hidden Understanding Models for Machine Translation. *In Proc. ECSA*, 1999.

[Mon74] Montague, R.: *Formal Philosophy*, Yale U.P., New Haven, 1974

[Mou04] Mouček, R.: Semantics in Dialogue Systems, *Doctoral Thesis*, Pilsen, 2004.

[MS01] Manning, Ch.D., Schütze, H.: *Foundations of statistical natural language processing*, The MIT Press, 2001.

[NJ98] Nancy, I., Jean, V.: Word sense disambiguation: The state of the art, *Computational Linguistics, Volume 24*(1), 1998, 1-40.

[Pie92] Pieraccini, R., Tzoukermann, E., Gorelov, Z., Levin, E., Lee, C-H., Gauvain, J-L.: Progress Report on the Chronus System: ATIS Benchmark Results. *In Proc. of the workshop on Speech and Natural Language*, 1992.

[Pra04] Pradhan, S., Ward, W., Hacioglu, K., Martin, J., Jurafsky, D.: Shallow Semantic Parsing using Support Vector Machines. *In Proc. of the HLT/NAACL*, 2004.

[Pri90] Price, P.: Evaluation of Spoken Language Systems: the ATIS Domain. *In Proceedings of the third DARPA Speech and Natural Language Workshop*, 1990.

[PS04] Pala, K., Smrž, P.: Building Czech Wordnet, *Romanian Journal of Information Science and Technology 2004*(7), 2004, 79-88.

[Rab89] Rabiner, L.R.: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, *Proceedings of the IEEE vol. 77*, no. 2, 1989

[Rtn99] Ratnaparkhi, A.: A Maximum Entropy Model for Part-of-Speech Tagging, *In Proc. the Conference on Empirical Methods in Natural Language Processing*, 1999

[Sch96] Schwartz, R., Miller, S., Stallard, D., Makhoul, J.: Language Understanding Using Hidden Understanding Models. *In. Proc. ICSLP*, 1996.

[Tho00] Brants, T.: TnT – A Statistical Part-of-Speech Tagger, *In Proceedings of the Sixth Applied Natural Language Processing*, Seattle, Ca, 2000.

[You02a] Young, S.: Talking To Machines (Statistically Speaking). *In Proceedings of the International Conference on Spoken Language Processing*, 2002.

[You02b] Young, S.: The Statistical Approach to the Design of Spoken Dialogue Systems. *Tech. rep. CUED/F-INFENG/TR.433*, Cambridge, 2002.