FP7-ICT-2011.4.2
Contract no.: 288342
www.xlike.org

Deliverable D3.4.1

Reporting on existing USP-techniques

Editor:	Achim Rettinger, KIT
Author(s):	Achim Rettinger, KIT; Lei Zhang, KIT;
Deliverable Nature:	Report (R)
Dissemination Level: (Confidentiality)	Public (PU)
Contractual Delivery Date:	M18
Actual Delivery Date:	M18
Suggested Readers:	All partners using XLike Toolkit
Version:	0.1
Keywords:	Unsupervised Semantic Parsing, Dependency Tree

Disclaimer

Full Project Title:	XLike – Cross-lingual Knowledge Extraction
Short Project Title:	XLike
Number and Title of Work package:	WP3 – Cross-lingual Semantic Annotation
Document Title:	D3.4.1 Reporting on existing USP-techniques
Editor (Name, Affiliation)	Achim Rettinger, KIT
Work package Leader (Name, affiliation)	Achim Rettinger, KIT
Estimation of PM spent on the deliverable:	6 PM

Executive Summary

Unsupervised Semantic Parsing (USP) is an existing knowledge extraction method which tries to build clusters of syntactic variations of the same meaning. Given are dependency trees of a text collection as they are output by task T2.2. Thus, the input is already syntactically analysed and the output is semantic content, which then can be used as a knowledge representation and mapped to an existing knowledge representation. The goal of task T3.4 is to extend existing approach to multiple languages and test if clustering of syntactic variations across languages but of the same meaning can be achieved. In this deliverable D3.4.1, we will report the existing USP-techniques to the multilingual parse trees from T2.2. In the subsequent deliverable D3.4.2, we will provide the development, implementation and testing of extensions to USP-techniques from D3.4.1 for cross-lingual USP.

Table of Contents

List of Figures and/or List of Tables

Abbreviations

USP	Unsupervised semantic analysis
MLN	Markov logic networks
MAP	M*aximum a posteriori probability*

Definitions

First-order logic	A language to analyse the formal structure of statements in a domain of discourse.
Markov Logic Networks	A combination of first-order logic with Markov Networks.

1 Theoretical Foundations

This section aims to introduce the major definitions and concepts which provide the theoretical background of Unsupervised Semantic Parsing (USP). First, we will review some basic terms from first-order logic, followed by definitions and tools from the field of Natural Language Processing. Next, we will explain some concepts from the probability theory used in machine learning.

1.1 First-Order Logic

First, we would like to provide an introduction to the basics of first-order logic, which is used in USP to represent meaning. Further, USP makes use of the lambda notation and hence it is useful to introduce the respective definitions.

First-order logic provides a language to analyse the formal structure of statements in a domain of discourse. Beyond the declarative propositions of propositional logic it deals with predicates and quantifiers. A predicate defines a placeholder for a type of objects, while a quantifier provides a statement about the binding of the predicate in the domain of discourse. Important definitions of first-order logic:

Term is a formal expression providing the name of an object or form. A term is any symbol denoting an individual constant or a functional symbol of the form f ($t1$, . . . , tn) , where f is a functional symbol with arity n and terms $t1$, . . . , tn. Any **variable** is a term.

Constant is a symbol with the functional valence of zero, e.g. a name or an object. Constants are often represented by small letters a, b, c, ...

Predicate expresses the value of an attribute regarding an object, while this object can posses this attribute or not, e.g. if the predicate *Phil* represents a philosopher, then *Phil* (a) is true iff a is a philosopher.

Atom (or **atomic formula**) is the smallest type of a formula. It can not be divided into sub-formulas. An atom has the form p ($t1$, . . . , tn), with p being a predicate symbol of arity n and $t1$, . . . , tn terms. The simplest form of an atom is a formula representing a relation. For example, the statement “Mircrosoft bought Skype” is represented by the following atomic formula *BOUGHT*(*Microsoft*, *Skype*). Additionally, it is a **ground atom** since all arguments are constants.

Formula is type of a symbolic representation. It is a construction of atoms and make use of quantifiers and logical connectives.

Formulas are recursively constructed using logical connectives and quantifiers, which allow the representation of complex dependencies between atoms. Provided the formulas $F1$ and $F2$ and a variable x, the following are also formulas: $\neg F1$ (negation); $F1 \wedge F2$ (conjunction); $F1 \vee F2$ (disjunction); $F1 \Rightarrow F2$ (implication); $F1 \Leftrightarrow F2$ (equivalence); $\exists x\, F1$ (existential quantification), $\forall x F1$ (universal quantification). In the following an example of a phrase and its representation in first order logic formulas.

Phrase: A restaurant that serves Mexican food near Golden Gate.

First-Order Logic:

$$\exists x\,(Restaurant\,(x)\ \wedge\ Serves\,(x,MexicanFood)$$

$$\wedge\, Near((LocationOf\,(x), LocationOf\,(GoldenGate)))$$

As demonstrated in the example above, first-order logic is suitable as a language for meaning representation.

1.1.1 Lambda Notation

A useful tool for semantic analysis is the lambda notation, which helpswith the abstraction of fully specified first-order logic formulas. The first-order logic syntax is extended in order to use the following expression type (compare to [6]):

$$\lambda x.P(x),$$

where lambda is followed by one or several variable and a formula which employs those variables.

1.1.1.1 Lambda Reduction

λ-reduction is a process of applying *l*-expressions to logical terms in order to yield new first-order logic expressions with formal parameter variables bound to a specific term:

$$\lambda x.P(x)(A) \underset{\lambda-reduction}{\Rightarrow} P(A).$$

An example with actual first-order-logic formulas and constants:

$$\lambda x.\lambda y.Near(x, y)\ (SanFrancisco)(MountainView)$$

$$\underset{\lambda-reduction}{\Rightarrow} \lambda y.Near\ (SanFrancisco)(MountainView)$$

$$\underset{\lambda-reduction}{\Rightarrow} Near\ (SanFrancisco,MountainView).$$

1.2 Natural Language Processing

In this section, we will discuss some fundaments of natural language processing, an important part of USP. Natural Language Processing (NLP) is a field of computer science and linguistics occupied with studies of natural (human) language. In this section, we introduce important definitions from morphology, which is a domain of linguistics. Further, we will discuss syntactic and semantic analysis, methods used to process natural language and other inputs. We will then explain quasi logical forms, a semantic formalism used in USP, and finish this section with the introduction of the Stanford- Parser.

1.2.1 Morphological Analysis

Morphology is a domain of linguistics, which deals with the meaning of words in natural language. Below are some important definitions:

Lemma is the dictionary form of a set of words. E.g. for the words *run*, *ran*, *runs*, *running* the lemma would be *run*.

Surface form is the form of a word found in a text, e.g. the words *run*, *ran*, *runs*, *running* are all surface forms.

Lexical form of a surface form is the lemma followed by the grammatical information. The lexical form of *running* would be *run, verb, continuous.* A lexical form represents one meaning and can have different inflections.

Lexical item is limited to a single meaning and may have different inflections. In contrast to the lexical form it can consists of several words.

Part of Speech (POS) is the linguistic category of a lexical item. The most frequent POS' are Noun, Verb, Adverb, Adjective. One of the most populars lists of POS tags is defined in the Brown Corpus (see [5]).

Lexical analysis is the process of converting a sequence of characters into a sequence of tokens.

Formal language is a set of words over an alphabet. Often defined through a formal grammar, which defines the purely syntactic rules.

Formal grammar generates a formal language.

Backus Normal Form (BNF) is a notation technique for context-free grammars to describe the syntax of languages.

Context-free grammar generates a formal language.

Probabilistic context-free grammar (PCFG) is a context-free grammar in which each production is augmented with a probability.

Treebank or parsed corpus: A text corpus in which each sentence has been annotated with syntactic structure (parsed). Used for training of parsers.

1.2.2 Syntactic Analysis

Syntactic Analysis or Parsing is the process of mapping the output of lexical analysis to formal grammar, which usually results in a syntactical tree. The latter is suitable for a variety of use cases. A syntactic tree, or parse, represents the grammatical structure of a sentence.

Typically, syntactic analysis starts with a string input, which is converted into a set of tokens with the help of lexical analysis and a grammar of regular expressions. In the next step, the tokens are parsed to a parse tree, which checks that the tokens form an valid expression according to a context-free grammar.

1.2.3 Semantic Analysis

Syntactic analysis creates a parse tree, while the semantic analysis or a semantic parse interprets this input. In particular, it relates the syntactic structure to a languageindependent meaning. The semantics of a sentence can be expressed in predicate logic, an other possibility of semantic representation is the quasi logical form described in the next section.

1.2.4 Quasi-Logical Form

Quasi Logical Form (QLF) is a semantic formalism developed with the motivation of having a practical method to describe semantics of a sentence. This method is used in USP and hence is of importance for this work. QLF expressions can appear as terms or formulas, which are defined below (see [1] for further information).

QLF-term is one the following:

· a term index: *x*, *y*, ...

· a constant term: 3, *table*1, ...

· a term variable: +*i*,+*j*, ...

· a term expression: *term*(*Index*, *Equation*,*Quantifier*, *Referent*)

QLF-formula is one of the following:

· the application of a predicate to an arguments: *Predicate*(*Argument*1, ...*Argumentn*)

· an expression of the form: *form*(*Idx*, *Category*, *Restriction*, *Resolution*)

· a formula with scoping constraints: *Scope* : *Formula*

· a formula with a re-interpretation: *Formula* : {*Term*1/*Term*2, *Term*3/*Term*4, ...}

Figure 1 illustrates a set of QLFs expressions and the corresponding predicate logic formulas. Figure 1a shows a term expression with an index, a referent in the form of a lambda-abstraction and a quantifier. Figure 1b contains a variable formula, with a variable and a scope formula. The scope form embeds a scope and a formula of the predicate type. Figure 1c contains the QLF expression and the predicate logic formula for the sentence “Everybody speaks two languages”. This example illustrates very well the usage of QLF and is described in further detail in [17].

1.2.5 Stanford Parser

The Stanford Parser is a natural language parser, which discovers the grammatical structure of a sentence. It is a statistical parser, which means it takes use of hand-parsed sentences in order to deliver the most probable analysis of new input. The Stanford Parser loads a Probabilistic Context-Free Grammar for English before processing the input files. Currently, the Stanford Parser is mainly trained on a corpus with very formal sentences like the Wall Street Journal or Biomedical English.

The question is how well the Stanford Parser can understand the grammatical structure of a sentence (i.e. build the dependencies, which are the input for USP). It is a probabilistic parser and is trained on hand-parsed sentences. Therefore, it will output the most likely analysis of the input, based on the training data. It is possible to train the parser with a different corpus, however syntactically annotated data is required. For this thesis, we are interested in two outputs of the Stanford Parser, namely the part of speech tagging and the dependency trees. While the first was described earlier, we will briefly discuss the latter. A **dependency tree** provides a simple description of grammatical relations in a sentence. In particular it looks at different parts of the sentence and outputs the grammatical relation between those. The most efficient explanation can be done in the form of an example. The example sentence "My dog also likes eating sausage." has the following typed dependency representation:

poss(dog-2, My-1)
nsubj(likes-4, dog-2)
advmod(likes-4, also-3)
xcomp(likes-4, eating-5)
dobj(eating-5, sausage-6)

Where *nsubj* stands for the relation *nominal subject* and *dobj* stands for the relation *direct object*. Further information and explanation about typed dependencies can be found in the StanfordParser manual (see [10]). More information about the Stanford Parser can be found at the projects website.

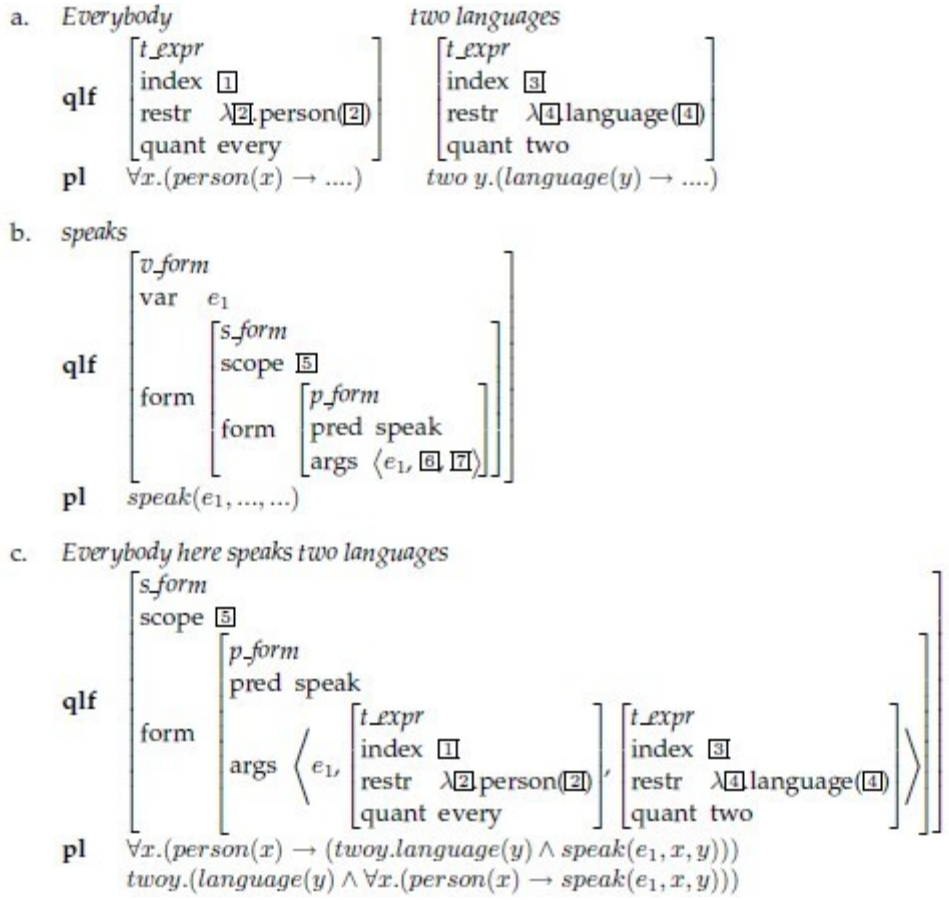

Figure 1: The relation between an expression in QLF and a formula of predicate logic from [17]

1.3 Probability Theory and Machine Learning

This section contains important definitions from probability theory, which are used in USP. We will start by introducing Markov Networks, Markov Logic Networks and MAP. Last, we will discuss the difference between supervised and unsupervised methods in machine learning.

1.3.1 Markov Networks

In a Markov Network or Markov Random Field, a set of random variables $X = (X_1,..., X_n) \in \chi$ with the Markov property is described by an undirected graph *G* with links, which describe symmetrical probabilistic dependencies according to a set of potential functions ϕ_k (compare with[12]).

Variables are represented as nodes in the graph, while the probabilistic interactions between the variables (functions), by edges in the graph. In order to define a global model for a graph, the functions are combined and thus multiplied like in a Bayesian Network. Koler at al. (see [8]) provide the following normalization distribution:

$$\mathrm{P}(\mathrm{X} = \mathrm{x}) = \frac{1}{Z}\prod_k \phi_k(x_{\{k\}})$$

with the partition function as the normalization constant. Markov networks can be expressed through a log-linear model. The potential functions can often be easier to interpret, when converted to the log-space:

$$\phi(\chi) = \exp(f(\chi)).$$

With this in mind, the probability distribution of a Markov network can be represented as:

$$P(X_1,..., X_n) = \frac{1}{Z}\exp(\sum_{i=1}^{k} \omega_i f_i(x)),$$

with f_i being a feature and ω_i being a weight. A feature is any state function in $\Re$, of particular interest for our work are binary features with values *fi*(*x*) $\in$ {0, 1}. In relation to the potential function, each possible state $x_{\{k\}}$ of a clique is a feature with the weight $\omega_i = \log\phi_k(x_{\{k\}})$. As suggested by Koller at al. (see [8]), the advantage of the log-linear model is the possibility to represent many distributions with large-domain variables in a much more compact way.

1.3.2 Markov Logic Networks

Markov Logic Networks (MLN) combine first-order logic with Markov Networks. Firstorder logic makes an absolute statement about a world, where a violation of a formula states that such a world has the probability of zero. Kok and Yin suggest in [7] that in Markov Logic, each formula receives a weight in such a way, that a violation of the formula makes such a world less probable, but not impossible [7]. A Markov Logic Network is defined by Richardson and Domingo [16] as follows:

A Markov Logic Network *L* is a set of pairs (F_i, ω_i), where F_i is a formula in first-order logic and ω_i is a real number. Together with a finite set of constants $C = \{c_1, c_2,..., c_{|C|}\}$, it defines a Markov network $M_{L,C}$ as follows:

1. $M_{L,C}$ contains one binary node for each possible grounding of each atom appearing in *L*. The value of the node is 1 if the ground atom is true, and 0 otherwise.

2. $M_{L,C}$ contains one feature for each possible grounding of each formula F_i in *L*. The value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is the ω_i associated with F_i in *L*.

From this definition and the formulas described in the previous section, a ground Markov network $M_{L,C}$ has the following probability distribution over possible worlds *x* (see [13] for details):

$$P(X = x) = \frac{1}{Z}\exp(\sum_i \omega_i n_i(x)) = \frac{1}{Z}\phi_i(x_{\{i\}})^{n_i(x)}.$$

Where $n_i(x)$ is the number of true groundings of F_i in *x*; $x_{\{i\}}$ is the state of the atoms appearing in F_i and $\phi_i(x_{\{i\}}) = e^{\omega_i}$. The inference goal in MLN is to find a stationary distribution.

1.3.3 MAP

An important part of probability theory is to compute joint probability distributions over multiple random variables in order to answer queries of interest. Following Koller et al. (for details see [8]), the basic query is the *probability query* of the type:

$$P(Y|E = e),$$

where *Y* is a subset of random variables and *E* a subset of random evidence variables in the model, with an instantiation *e* to these variables. The probability query computes the posterior probability distribution over the values *y* of *Y*, conditioned on the fact that *E* = *e*.

Further, Bayesian Statistics defines the mode of the posterior distribution called MAP (*maximum a posteriori probability*). The problem of computing MAP is to find the most likely assignment to all of the (non-evidence) variables. For this purpose we define $\chi = X_1,..., X_n$ as the set of all evidence and non-evidence variables, *E* as the subset of evidence variables and $W = \chi - E$ as the subset of non-evidence variables. The MAP assignment is defined as follows (see [8] for details):

$$MAP(W \mid e) = \arg\max_{\omega} P(\omega, e),$$

with the $\arg\max_x f(x)$ function representing the value of *x* for which *f* (*x*) is maximal. There might be multiple values with the highest posterior probability.

1.3.4 Supervised vs. Unsupervised Methods

Algorithms in machine learning can be divided in several types. Two prominent types are supervised and unsupervised learning. USP is an unsupervised algorithm and therefore it is interesting to understand the idea behind unsupervised approaches. In supervised learning the algorithm aims to infer a function that maps some inputs to a desired output, the inferred function is called classifier. In order for the algorithm to do so, it requires an input of representative training examples, i.e. the algorithm needs a labeled input in such a way that for every input object an output is available. In contrast to this, unsupervised learning aims to find a function without labeled data, this means it has no training sample which would help to validate the constructed function.

2 Unsupervised Semantic Parsing

2.1 Introduction

This section discusses Unsupervised Semantic Parsing (USP). First, we will provide a theoretical framework for USP. This will include the discussion of the ideas behind this method, as well as the different parts, approaches and algorithms behind the concept. Next, we will go into detail on the implementation of USP. This will contain an overview over the classes used, the outline of the implemented algorithm and the output delivered by the system.

2.2 USP - Unsupervised Semantic Parsing

The idea behind unsupervised semantic parsing (USP) is to cluster synonyms and synonymic expressions. A cluster contains several relations, which are interchangeable in the same context. The USP system has been developed by Hoifung Poon and Pedro Domingos from the University of Washington. The code, their test data set and some documentation can be found at their website[14]. Three key concepts can be identified behind USP[15]:

· The same meaning can be expressed in different forms, i.e., a different words can have the same meaning in a particular context. E.g. the sentences "The author grew up in Germany" and "The author is from Germany" provide the same information, but uses different relation to do so, namely, "grew up in" and "is from". Both relations can be viewed as semantically interchangeable and thus be clustered together. Further it is assumed, that this interchangeability can be learned from data.

· A meaning is not encapsulated by just a single word, but can be extended to word combinations of arbitrary length. Therefore, the previously described clustering process is simultaneously looking for forms and for composition of forms. Clustered forms can be built up recursively of sub-forms.

· USP connects syntactic and semantic parsing. It does so by starting with a syntactic analysis, the results of which are transformed into a semantic content. By doing so, the complexities of syntactic and semantic analyses are separated, which leads to performance improvement. For the sake of completeness, it should be noted that USP is always considering relations on the scope of one sentence.

2.3 USP - Fundamentals and Algorithm

Unsupervised Semantic Parsing clusters only content words, i.e. words which belong to the following categories: Noun, Verb, Adjective, Adverb. Further, it requires information on relation between those words in a sentence. Therefore, the USP system receives two types of inputs for a text corpus: sentences with annotated part of speech tags and dependency trees of training sentences. Dependency trees already contain a lot of the information about relations in the sentence, the POS tags add further information.

2.3.1 From Dependency Trees to Quasi-Logical Form

In the first step of the USP algorithm, dependency trees are converted to quasilogical forms (QLFs). The nodes of a dependency tree are words, which are represented as unary atoms in QLF. An unary atom consists of a lemma and its POS tag. The edges of a dependency tree are the grammatical relations, which correspond to dependency labels in QLF. E.g. the sentence "*Microsoft bought Skype"* contains the nodes "*Microsoft"*, "*Skype"* and a subject and object relation. These correspond to *Microsoft*(n_1), *Skype*(n_2), *bought*(n_3), *nsubj*(n_1, n_3) and *nobj*(n_2,n_3). Thereby, n_i represent Skolem constants indexed by the nodes. The resulting QLF of the example sentence consists of the conjunction of the atoms and dependency labels: $bought(n_3) \wedge Microsoft(n_1) \wedge Skype(n_2) \wedge nsubj(n_1, n_3) \wedge dobj(n_2, n_3)$. As seen in the example, the meaning composition of two formulas is their conjunction.

Every QLF sub-formula F is converted to a corresponding lambda form, e.g. the QLF sub-formula $bought(n_1) \wedge nsubj(n_3, n_1) \wedge dobj(n_3, n_2)$ is represented by the lambda form $\lambda x_1 \lambda x_2 .bought(n_3) \wedge nsubj(n_3, x_1) \wedge dobj(n_3, x_2)$. A lambda form is further decomposed into the core form and the argument form. This allows

to handle a variable number of arguments. The core form does not contain any lambda variables (e.g. *bought*(n_3)), while an argument form contain one lambda variable (e.g. λx_1 .*nsubj*(n_3, x_1) or λx_2 .*dobj*(n_3, x_2)).

A lambda-form cluster is representing syntactic variations of the same meaning. For example, the sentences "*Microsoft bought Skype"* and "*Microsoft acquired Skype"* have the same meaning, therefore "*bought"* and "*acquired"* can be interchanged in this context and thus they would be in the same lambda form cluster. Each lambda form cluster may have one or more *argument types*. Argument types describe distinct grammatical relations the core forms of the cluster are involved in. The sentences *Microsoft bought Skype* and *The Microsoft campus is in Redmond* both have the lambda form Microsoft which would represent a lambda-form cluster with the argument types *nsubj* for nominal subject (between Microsoft and bought) and *nn* for noun compound modifier (between Microsoft and campus) respectively. An abstract lambda form is defined by Poon and Domingos[15] as follows:

Given an instance of cluster *T* with arguments of argument types $A_1,...,A_K$,
its *abstract lambda form* is given by $\lambda x_1 \cdots \lambda x_k.T(n) \wedge \Lambda_{i=1}^{k} A_i(n, x_i)$

USP is converting the input sentence into a QLF, then partitions the atoms in the QLF by dividing each part into core form and argument forms. Further each form is assigned to a lambda-form cluster or its argument type. The final logical form is the abstract lambda form of the parts derived by using the *l*–reduction rule.

2.3.2 Markov Logic Network in USP

The USP system performs a semantic parse of the sentences in quasi-logical form, by dividing the sentences in logical components. A semantic parse *L* partitions a QLF Q into **parts** $p_1, p_2, \cdots, p_n$**.** Each part is a content-type word (i.e. token) in an input sentence. Content-type means words of the grammatical type *noun*, *verb*, *adverb* and *adjective* and excludes particles such as prepositions or articles. In the following, we will use 'word' in lieu of content-type word. Each part is assigned to a **lambda-form cluster** *c*. A new part is created for each individual word in each sentence, while a lambdaform cluster represents a distinct word to which a part instance is assigned. Therefore, several parts can be assigned to one lambda-form cluster but not vice versa. A part is further partitioned in a **core form** *f* and **argument forms** $f_1, \cdots, f_k$. The core form represents the actual word, while an argument form describes a relation the core form is involved with. Each argument form is assigned to an **argument type** *a* in a lambda-form cluster *c*. As with parts and clusters, an argument type aggregates the information about the relations of a cluster over the text corpus, while an argument form is on sentence level. Figure 2 provides an example of the parse of two sentences and illustrates the difference between the aforementioned terms. The word "information" appears twice in the sentences and hence two parts are created for this core form. However there is just one cluster for this token. Further, p_3 contains the argument forms *conj_and* and *nn*, p_{11} contains the argument form *nn*, which are aggregated in the two argument types of the cluster c_3 with the respective references.

Further, Poon and Domingos *[15] define the USP MLN as a joint probability over [the QLF]* Q *and [the semantic parse]* L *by modeling the distribution over forms and argument given the cluster and argument type*. In order to model the distributions over lambda forms, some predicates need to be specified. Below, p is a part, i is the index of an argument and f is a QLF sub-formula; further ! denotes that each part or argument can have only one form.

(1) **Form**(**p**, **f**!) is true iff part p has core form f.

(2) **ArgForm**(**p**, **i**, **f**!) is true iff the i-th argument in p has form f.

In order to model the distributions over arguments, the following predicates are defined.

(3) **ArgType**(**p**, **i**, **a**!) signifies that the i-th argument of p is assigned to argument type a.

(4) **Arg**(**p**, **i**, **p**′) signifies that the i-th argument of p is p' .

(5) **Number**(**p**, **a**, **n**) signifies that there are n arguments of p that are assigned to type a.

For better understanding, some examples for the predicates (1) to (5); the examples refer to Figure 2.

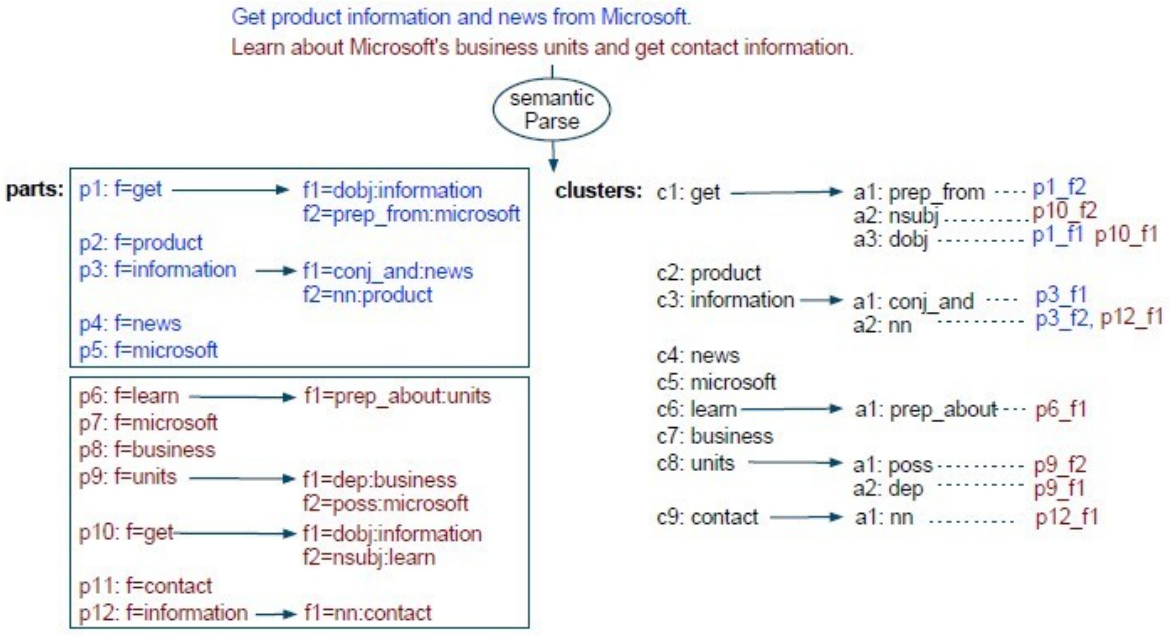

Figure 2: Example of a Semantic Parse, with the resulting parts, core forms, argument forms, clusters and argument types

(1) The part *p1* has the core form *get* and hence *Form(p1, get)* is true, the same holds for *Form(p9, units)*.

(2) The second form argument of part *p1* has the form microsoft and hence *ArgForm(p1, 2, microsoft)* is true, the same holds for *ArgForm(p9, 1, business)*.

(3) The second argument of *p1* is microsoft, it is assigned to argument type *prep_from* in *c1*, therefore *ArgType(p1, 2, c1_a1)* holds, as well as *ArgType(p9, 1, c8_a2)*.

(4) The second argument of *p1* is microsoft, which is assigned to part *p5*, the fore *Arg(p1, 2, p5)* holds, as well as *Arg(p8, 1, p7)*.

(5) The argument *p1_f2* is assigned to *c1_a1*, but no other arguments of *p1* are assigned to *c1_a1*, therefore *Number(p1, c1_a1, 1)* holds.

Poon and Domingos capture USP in four formulas *[15]* :

$$p \in +c \wedge Form(p, +f)$$
$$ArgType(p, i, +a) \wedge ArgForm(p, i, +f)$$
$$Arg(p, i, p') \wedge ArgType(p, i, +a) \wedge p' \in +c'$$
$$Number(p, +a, +n).$$

The free variables are implicitly universally quantified. The + denotes that the MLN has a weighted instance of the respective formula, for each value combination of the variable with a plus sign. The first formula notes that a part has a core form and a cluster. The second formula describes that an argument in p is assigned to a form f and a type a. The next formula states that a part p has an argument a, which is simultaneously part p', which in turn is element of cluster c'. The last formula is simply the of a predicate.

2.3.3 Inference

The conditional probability for a semantic parse *L*, provided a sentence and its QLF *Q*, is defined by Poon and Domingos [15] as:

$$\Pr(L \mid Q) \propto \exp(\sum_i \omega_i n_i(L, Q)),$$

with the MAP semantic parse:

$$\arg\max_L \sum_i \omega_i n_i(L,Q).$$

Since it would be too expensive to calculate the values for all *L*'s, USP uses a greedy algorithm to search for the MAP parse. The algorithm draws on some definitions [15]:

A partition is called a λ *-reducible form* p if it can be obtained from the current partition by recursively λ-reducing the part containing p with one of its arguments. A λ-reducible partition is called *feasible* if the core form of the new part is contained in some cluster. Lets consider the QLF of "Microsoft bought Skype" with the partition $\lambda x_1 x_2$.*bought*(n_3) $\wedge$ *nsubj*(n_3, x_1) $\wedge$ *dobj*(n_3, x_2),*Microsoft*(n_1), *Skype*(n_2). The partition λx_2 .*bought*(n_3) $\wedge$ *nsubj*(n_3, n_1) $\wedge$ *Microsoft*(n_1) $\wedge$ *dobj*(n_3, x_2), *Skype*(n_2) is λ-reducible. The new partition is feasible if the core form of the new part λx_2 .*bought*(n_3) $\wedge$ *nsubj*(n_3, n_1) $\wedge$ *Microsoft*(n_1) $\wedge$ *dobj*(n_3, x_2) (i.e. λx_2 .*bought*(n_3) $\wedge$ *nsubj*(n_3, n_1) $\wedge$ *Microsoft*(n_1)) is part of some lambda-form cluster.

Algorithm 1 USP-Parse(MLN,QLF)

Form parts for individual atoms in QLF and assign each part to its most probable cluster
repeat
  for all parts p in the current partition **do**
    for all partitions that are *l*-reducible from p and feasible **do**
      Find the most probable cluster and argument type assignments for the new part and its arguments
    end for
  end for
  Change to the new partition and assignments with the highest gain in probability
until none of these improve the probability
return current partition and assignments

Algorithm 1[15] provides the pseudo-code for this part of the USP system. The finding of λ-reducible partitions from a part p can be done in time O(ST), where *S* is the amount of the clusters defined by the number of core forms and *T* is the maximum number of atoms in a core form. Given a new part p with the respective cluster, there are *km* ways of assigning p's *m* arguments to the *k* argument types of the cluster. Since this is expensive for larger *k* and *m*, USP uses an approximation by assigning each argument to the best type, independent of other arguments. This algorithm is considered to be very efficient and is used repeatedly throughout the USP system.

2.3.4 Learning

The learning problem in USP is to find the MLN weights and semantic parses, which maximize the log-likelihood of observing dependency-tree generated QLFs. It is defined as follows[15]:

$$L_\theta(Q) = \log P_\theta(Q)$$

$$= \log \sum_L P_\theta(Q, L)$$

Where *L* are the semantic parses, *q* are the MLN parameters and P_θ (*Q*, *L*) are the completion likelihoods. Algorithm 2 provides pseudo-code for the USP learning algorithm. It takes an MLN without defined weights and QLFs from the dependency trees as input. The algorithm contains two major operators, which are used for updates of semantic parses. The MERGE(C1, C2) operator merges the argument types of two clusters [15]:

1. Create a new cluster C and add all core forms in C_1, C_2 to C;

2. Create new arguments types for C by merging those in C_1, C_2 so as to maximize the log-likelihood;

3. Remove C_1, C_2.

The merging process takes the argument forms in the argument types of the clusters and puts them into a new argument type. Alternatively, if the type already exists, it is merged into the type. If a new or an existing type is considered depends on the log-likelihood. Further, the types are considered in the order of maximal number of occurrences in both clusters; this provides more information for each decision. This operator clusters syntactically different expressions with the same meaning as defined by the model. The described method is an approximation, which reduces the amount of possibilities to create new argument types considerably.

The COMPOSE(C_R, C_A) operator is used to merge two existing clusters into one:

1. Create a new cluster C;

2. Find all parts r $\in$ C_R, a $\in$ C_A such that a is an argument of r, compose them to r(a) by λ-reduction and add the new part to C;

3. Create new argument types for C from the argument forms of r(a) so as to maximize the log-likelihood.
This operator creates clusters of large lambda-forms, provided that they are composed of the same sub-forms (e.g. “is acquired by”). These can later be merged with other clusters.

Algorithm 2 contains the pseudo-code of the algorithm executed in the USP system. In the first step, the USP algorithm performs semantic parses and creates initial clusters. Then the system applies the MERGE operation on clusters with the same core form. Further, USP looks at candidate operations (all potential clusters which could be merged). These operations are evaluated and if the score is above a threshold, the operations are added to the agenda. The agenda contains evaluated operations, which are pending execution. In the next step, the candidate operations are being executed in decreasing order of their scores. If the score is above a threshold after the execution, the clusters underlying the candidate operations are merged into one. This process is repeated until the agenda is empty, the outputs are MLN parameters and semantic parses.

Algorithm 2 USP-Learn(MLN,QLF)

```
Create initial clusters and semantic parses
Merge clusters with the same core form
Agenda ← φ
repeat
    for all candidate operations O do
        Score O by log-likelihood improvement
        if score is above a threshold then
            Add O to agenda
        end if
    end for
Execute the highest scoring operation O* in the agenda
Regenerate MAP parses for affected QLFs and update agenda and candidate operations
until agenda is empty
return the MLN with learned weights and the semantic parses
```

The USP system faces two major challenges The first challance is the identifiability problem common in unsupervised learning methods. This problem in MLN is due to the multitude of optimal parameters. E.g. lets consider the formula $p \in +c \wedge Form(p,+f)$, which is satisfied by exactly one value of f conditioned on $p \in c$, therefore a constant number can be added to the weights of the formula without changing its probability distribution. This problem is addressed by Poon and Domingos [15], by imposing local normalization constraints on specific groups of formulas that are mutually exclusive and exhaustive. For the first USP formula, a group is defined by all instances with a fixed cluster c. For the other USP formulas, a group is defined by all instances with a fixed argument type a. The normalization requires

that $\sum_{i=1}^{k} e^{\omega_i} = 1$ with ω_i being the weight of a formula in the group. With this constraint it is possible to compute the completion likelihood $P(Q, L)$ in closed form for any L. Further the optimal weights that maximize the completion likelihood are derived in closed form using empirical relative frequencies. The optimal weights for the first USP formula is $\log(n_{c,f}/n_c)$, with $n_{c,f}$ being the number of parts p that satisfy the predicate $Form(p, f)$ and $p \in c$, and n_c is the number of parts p that satisfy $p \in c$.

2.4 USP Implementation

In this section we will describe the implementation of the USP system as it is provided by Poon and Domingos [14]. The implementation is done in Java, however we will try to abstract it from the particular programming language and prove a high level view of the implementation. First, we will describe the input format of USP. Then we will talk about the preparation of the text corpus for the input into the USP algorithm. Next, a detailed description of the algorithm implementation will be described, which involves the following steps:

1. Reading the Input Sentences

2. Initialization and Creation of Initial Clusters

3. Merging of Arguments

4. Creation of the Agenda

5. Processing the Agenda

Last, we will address the output of the USP algorithm and how it can be used for question answering.

2.4.1 USP Input

The USP system expects three types of input files for a document: the .dep file which contains dependency relations between parts of a sentence and the .input file with POS tags, as well as the .morph file which is the .input file without the POS tags. In the following an explanation of the input files as described by an example.

2.4.1.1 Text

The original document is a plain .txt file.

Example: *The KIT campus is in Karlsruhe.*

2.4.1.2 Dependencies

The dependencies represent sentence relationships. A dependency is a triple of the relation between a pair of words in a sentence. The dependencies are created with the “typedDependencies” output format option of the Stanford Parser, here the sample output for the example sentence:

det(campus-3, The-1)
nn(campus-3, Microsoft-2)
nsubj(is-4, campus-3)
prep_in(is-4, Redmond-6)

2.4.1.3 Input

The .input is a file where the original document is split up into sentences with a word per line and a line between the sentences. Additionally, each word has its part of speech tag in the line.

In order to create this file, first the “wordsAndTags” output format option of the Stanford Parser was used, which creates a file of the following format:

The/DT KIT/NNP campus/NN is/VBZ in/IN Karlsruhe/NNP ./.

After some simple string manipulation the desired input format is created:

The_DT

KIT_NNP
campus_NN
is_VBZ
in_IN
Karlsruhe_NNP
._.

2.4.1.4 Morphologies

The morphology input is a simple string manipulation of the .input file:

the
kit
campus
is
in
karlsruhe
.

2.4.2 Creating the USP Input Files

The current data set from the NanOn project has a total of 5.414 papers from the nanoscience background. From each paper we have taken the abstract and prepared it for the processing with USP.

A Java program was written in order to create the input files for the USP system. This program reads the files from the input corpus and creates the input files for USP with the help of the Stanford Parser. The Stanford Parser is initialized with the following parameters: *"-encoding", "UTF-8", "-retainTmpSubcategories", "-outputFormat" "wordsAndTags,typedDependencies" , "-outputFormatOptions", "treeDependencies", "englishPCFG.ser.gz", null*. The parameter "typedDependencies" creates the .dep file, that the USP needs as an input. The parameter "wordsAndTags" creates a file, which can be easily edited in order to match the .input and .morph inputs. Furthermore, the encoding is set to UTF-8 to ensure the correct usage of special characters, such as greek letters which appear frequently in a scientific document. With "english- PCFG.ser.gz" the Stanford Parser receives the probabilistic model for the english language, which has been trained on the WSJ corpus among others. The input file is not considering sentences with less 4 or more than 100 words.

2.4.3 Reading the Input Sentences

USP reads the text corpus sentence by sentence. First, it takes all the tokens and the respective POS tags of a sentence from the .morph and .input files. Every **Sentence** object is broken down into three maps, which serve different purposes for the further processing of data. E.g.:

· **tokens**: [R:ROOT, PRP:we, R:recently, V:demonstrated, DT:a, J:new, N:process, IN:for, DT:the, N:formation, IN:of, R:partially, J:spherical, N:structures, IN:as, DT:an, J:omnidirectional, N:antireflection, N:coating, -LRB-:-lrb-, N:omni-ar, - RRB-:-rrb-, .:.]

· **token parents:** {1=<nsubj , 3>, 2=<advmod , 3>, 3=<ROOT , 0>, 5=<amod , 6>, 6=<dobj , 3>, 9=<prep_for , 3>, 11=<advmod , 12>, 12=<amod , 13>, 13=<prep_of , 9>, 17=<nn , 18>, 16=<amod , 18>, 18=<prep_as , 3>, 20=<appos , 18>}

· **token children:** {0=[<ROOT , 3>], 18=[<amod , 16>, <appos , 20>, <nn , 17>], 3=[<advmod , 2>, <dobj , 6>, <nsubj , 1>, <prep_as , 18>, <prep_for , 9>], 6=[<amod , 5>], 9=[<prep_of , 13>], 12=[<advmod , 11>], 13=[<amod , 12>]}

The token parents identify the parent relation of a token to other tokens in the sentence. E.g. the first token (we) is the nominal subject (nsubj) of the third token (demonstrated), hence "demonstrated" is the parent token of "we". Equally the token children save the inverse relations.

2.4.4 Initialization and Objects in USP

Unsupervised Semantic Parsing performs a clustering of synonyms or synonymic expressions. Initially, USP creates a cluster object for each distinct word in the corpus, this is called an initial cluster in the following. An initial cluster object represents a distinct word and contains all the information about the contexts of this word in the corpus. The cluster contains the positions of the word in the text, that is the index of all the sentences with the word. Additionally, for each of the individual appearances of the word, the relations to other words of the same sentence are attached.

During the initialization step, USP iterates through each token, sentence and article of the corpus. Each token in the corpus receives an individual ID, which is assigned to the object **TreeNode**. For each TreeNode a **Part** object is created, it represents a unique token with its ID and relations to other tokens of the same sentence, it corresponds to the *core form*. For each relation in a sentence, an **Argument** object is created, it is assigned to a Part object and contains the index of the respective token and the grammatical relation in which it is engaged. An Argument corresponds to an *argument form*. Furthermore, each unique token is represented through the **RelType** and **Clust** object. Each Clust object is associated with **ArgClust** objects, which represent all the relations a unique token is involved in throughout the corpus. A Clust object is the programmatic representation of a *l-form cluster* and an ArgClust object corresponds to an *argument type*.

The TreeNode class can be seen as an index of all the tokens in the text corpus. While the RelType class is comparable to a list of all unique tokens in the text corpus. Similarly, the Part class represents a concrete token in a sentence, while the Clust class represents a unique token from the corpus and its relations with other tokens of the same sentence all over the corpus. While Part considers the scope of a sentence, Clust represents relations on corpus level. The same is true for the Argument and ArgClust classes. While Argument captures relations inside a sentence, ArgClust represents relations on the corpus level. Further details about these objects:

TreeNode: A TreeNode assigns an id to each token, except for punctuation, prepositions and similar. E.g. the id “0012:5:010” stands for the 10th token of the 5^{th} sentence in the 12th input document of the text corpus. If the TreeNode has children, it shows the relation to these.

Part: A Part refers to a TreeNode. More broadly speaking, it is a particular token in a sentence in an article. It provides all information about a token in a particular sentence, i.e. its index and its relations to other part of the sentence. Each Part object is assigned to one Clust object.

Argument: Is assigned to a Part. It shows relations of this Part to an other part in the same sentence. A Part is linked to its parent Part and its children Parts through an Argument.

RelType: The class for a unique token, it exists just once for each token in the corpus. RelType can be seen as the vocabulary of a text corpus.

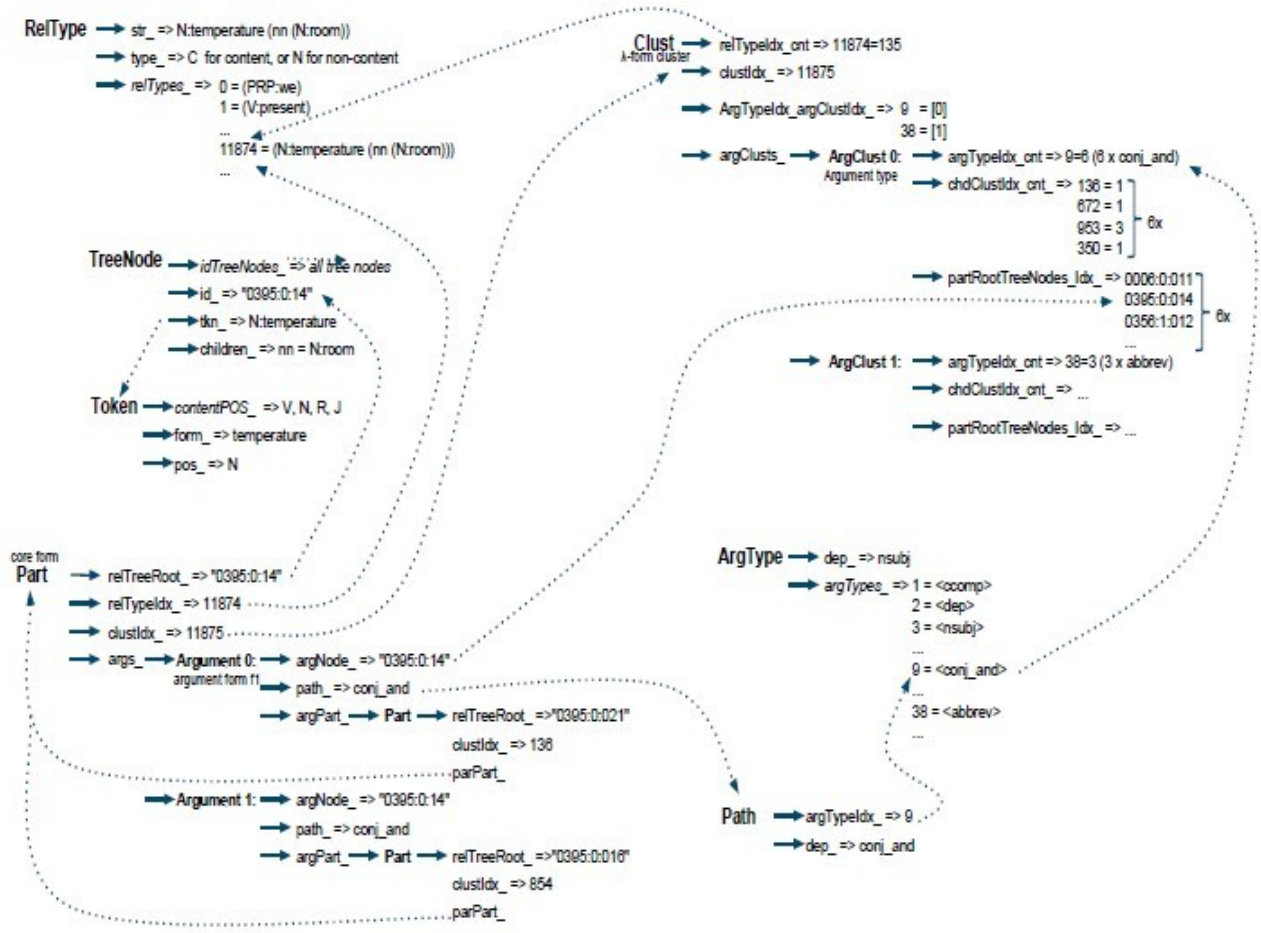

Figure 3: Overview over the objects of the USP system and the connections between those objects

Clust: Refers to a single RelType. It represents one unique token, provides information about the locations of this token in the corpus and the relations this word is involved in. It is the counterpart of Part on the corpus level.

ArgClust: For a given Clust it contains all relations of the assigned Parts. It provides information on the TreeNodes of the Parts, the relation types and the amount of occurrences. E.g. it could provide the information that the represented Clust 10 is 2 times in an "nn" relation with Clust 125 and 527, and once in a relation "nsubj" with Clust 345; the respective TreeNodes are 0010:3:023, 0098:1:012 and 0056:5:045.

Figure 3 provides an overview over the different classes in the USP system as well as the connections between those classes (see example in Figure 2). We call the objects created in this step *Initial Object Framework*.

2.4.5 Merge Arguments

This step corresponds to the MERGE operator, except that the argument merging is executed on one cluster. In this step all pairs of ArgClust of the initial cluster are scored. If the log-likelihood improvement is sufficient, the respective ArgClust pair is merged into one ArgClust.

2.4.6 Creation of the Agenda

This part of the algorithm creates candidate operations and adds them to the agenda. As input, this part receives the *Inital Object Framework*, built in the previous steps. The implementation provides the object SearchOp to represent a candidate operation and we will use this term in the following. For the understanding of the complexity of the algorithm, it is important to elaborate on the first two for-loops in Algorithm 3. The first for-loop iterates over all initial clusters, that is over all distinct content-type words in the text corpus. The second for-loop is iterating over all Parts which are associated with the current initial

clusters, i.e. all individual appearances of the word represented by the cluster in the text corpus. A SearchOp object is created for each pair of Parts attributed to Cluster, which results in a massive creation of SearchOps. In the next section, Figure 4a shows the polynomial growth of the SearchOp with increasing amount of input documents. Frequently created SearchOps are added to the agendaToScore, a list containing potential clusters.

Algorithm 3 Usp-CreateAgenda

Input: Initial Object Framework - objects created during Initialization and Merge Arguments

For each content-type Cluster c:

For each pair of Parts p1 and p2 with p2.id < p1.id and p1.clustIdx = p2.clustIdx = c.clustIdx

If p1.parentPart.clustIdx != p2.parPart.clustIdx **or if** p1.ai.clustIdx != p2.aj.clustIdx (where ai and aj are Arguments of the Parts):

Create a SearchOp so for p1.parentPart.clustIdx and p2.parentPart.clustIdx or p1.ai.clustIdx and p2.aj.clustIdx

Add so to the list mc_neighs.

If so appears more than a threshold of times (e.g. 10x) in mc_neighs, add it to the list agendaToScore.

Else:

Create a SearchOp so with p1.clustIdx and p1.parentClust.clustIdx

Add so to the list compose_cnt

If the so appears more than a threshold of times (e.g. 1225x) in compose_cnt, add it to agendaToScore.

Output: Agenda with SearchOp objects

2.4.7 Processing the Agenda

The created Agenda is being processed by following the structure laid out in Algorithm 4. The Agenda contains the list agendaToScore, which is the pool of candidate operations or SearchOps. These potential clusters are being scored and provided the fact that the log-likelihood improvement is sufficient, the SearchOps are being executed. During the execution two potential clusters are combined into one, i.e. the meaning composition is performed on the two lambda-form clusters represented by one SearchOp. At this point synonyms or synonymic expressions are clustered together.

2.4.8 Scoring

The USP system uses a sophisticated scoring algorithm to evaluate *MERGE* and *COMPOSE* operations. Explaining the scoring procedure in detail is beyond the scope of this paper, however we will provide a short overview over the operations. A scoring algorithm is used during the merge arguments step, where ArgClust are merged if a sufficient score is reached. The scoring of the merge arguments involves different factors, such as the total count of arguments of a cluster, count of root nodes associated with an ArgClust and the arguments associated with the ArgClust. Further, the scoring procedure is to score candidate operations. There are two type of candidate operations, the one in mc_neighs and the one in cnt_compose. These operations are evaluated considering such factors as their relation type, their parent and children clusters, their argument types, their argument counts and further statistics or connected objects. Overall the scoring tries to find two Clust objects with similar contexts.

Algorithm 4 USP-ProcessAgenda

Input: initialObjectFramework,

Agenda with SearchOp objects

Repeat

For each SearchOp so in list agendaToScore

Score so

If score is above threshold (e.g. -200):
If score is below a threshold (e.g. 10) put so in list clustIdxAgenda
Else put op in scoreActiveAgenda
Empty agendaToScore
Execute the MERGE/COMPOSE procedure for the SearchOp hso in scoreActiveAgenda with highest score
Delete hso
Update agendaToScore with operations in clustIdxAgenda
Until scoreActiveAgenda is empty
Output: MLN weights, Semantic Clusters, Semantic Parses

2.4.9 Output

The USP systems provides outputs with the MLN weights, the semantic parses for the input sentences and the created synonymic clusters. In the following we will describe each of the outputs in detail.

2.4.9.1 Synonymic Clusters

One of the outputs of the USP system contains the semantic clusters. These contain two or more lambda-form clusters with synonymic meanings. In the following a sample output:

[(J:different):563, (J:various):158]
[(V:studied):327, (V:examined):66, (V:investigated):390, (V:analyzed):121]
[(V:investigated (nsubj (PRP:we))):174, (V:studied (nsubj (PRP:we))):134]
[(N:gap (nn (N:band)) (prep_between (N:band))):2, (N:gap (nn (N:band))):333]
[(J:weak):42, (J:strong):173]

Each line represents a cluster from the output. A cluster is a pair, triplet, etc. of synonymic words or word combinations. The parentheses contain a word and its part of speech: V-Verb, R-Adverb, N-Noun, J-Adjective. The number after the parentheses gives the amount of appearances of the word in the text corpus, it is equivalent to the amount of Parts created for the distinct lambda-form cluster. The last line illustrates one problem about the USP MLN: it tends to cluster expressions with opposite semantic meanings. This is a general problem in semantic parsing and received some attention in [11].

2.4.9.2 MLN weights

The MLN output provides a detailed description of the lambda-form clusters created during USP execution. It lists all lambda-form clusters created, with the grammatical relations and the respective argument types (ArgClusts). The example below shows a sample output. The output contains three different lambda-form clusters: "*we*" which is the POS *personal pronoun* and appeared 138 times in the underlying text corpus, "*present*" being a *verb* appears 14 times and "*method*" a *noun* with 37 appearances. For the second lambda-form cluster the information about the argument types (ArgClusts) is available. E.g. the first argument type of "present" is of the relation "nsubj", the cluster is 13 times in this relation of which 12 are with the core form "we" and one of the core form "results". The following argument type is amerged one, with "advmod" and "prep_in".

1 [(PRP:we):138]
2 [(V:present):14]
1 1:13
5:<nsubj>:13
1:[(PRP:we):138]:12 175:[(N:results):74]:1
3 1:7
17:<advmod>:5 8:<prep_in>:2
762:[(R:then):13]:1 3112:[(R:here):8]:3 155:[(R:finally):9]:1 1595:[(N:addition):7]:1
172:[(N:paper):10]:1
...
3 [(N:method):37]

....

2.4.9.3 .parse

The semantic parse contains the information about the core forms and the respective argument forms (equivalent to Parts and Arguments in the implementation). Below a sample output of a parse. The word “cavity” is the second word of the third sentence in document 0000. Its cluster index is 6 and its Argument is “losses” which is the third word in the same sentence. The Parts “cavity” and “losses” are in the relation “nn”, which stands for noun compound modifier.

0000:3:002 cavity
    6 [(N:cavity):7]
    0000:3:003 37 [(N:losses):3]
    0 4 <nn>
0000:3:003 losses
    37 [(N:losses):3]
    0000:3:009 101 [(V:studied):42, (V:investigated):46]
    0 10 <nsubjpass>

2.4.10 Evaluation Questions

Poon and Domingos implemented a question engine *[14]*. It receives simple input questions of the type “What produces <Process A>?” or “What does <Element B> trigger?”. These simple questions can be provided to the USP system, which will be able to answer those, given the existence of a semantic parse of a text corpus with the respective information. The advantage of the algorithm is the fact, that answers can contain synonymic words, e.g. an answer might be “<Process A> delivers <Element C>.”.

3 Related Work

Semantic parsing relates the syntactic structure of a text to a language-independent meaning. Traditionally, semantic parsing has been executed manually, requiring a high amount of labour-intensive work. Recently, research in machine learning has addressed this tasks by developing supervised and unsupervised approaches. We will discuss some of these approaches in this section. First, we present two supervised approaches, followed by work on unsupervised approaches.

Wong and Mooney [18] have developed a statistical approach to semantic parsing with an algorithm called Word Alignment-based Semantic Parsing (WASP). The algorithm aims to construct a complete, formal, symbolic meaning representation of a sentence. WASP needs no natural language syntax input, however it requires a Context-Free Grammar of the target meaning-representation language, i.e. annotated sentences. In order to conduct a semantic parse, WASP needs a synchronous context-free grammar (SCFG), which defines the translation rules of a natural language string to its meaning representation. Since a rule might have multiple derivations, a probabilistic model serves as a mechanism to derive the correct derivations. The first step of the WASP algorithm is to induce a set of SCFG rules, i.e. a lexicon. The lexicon is learned by finding optimal word alignments between the natural language strings and their meaning representations. After a lexicon is built, the probabilist model is learned by means of a maximum-entropy model, which defines a conditional probability distribution over derivations given the observed natural language string. Wong and Mooney show recall rates of up to 70% for certain text corpora, which outperforms other models by up to 20%.

Zettlemoyer and Collins [19] propose an algorithm to map natural language to lambda calculus representation of their meanings. The input to this approach is a set of sentences in natural language form, as well as their logical forms, however the logical form derivation of the sentences is not annotated. Similarly to WASP, Zettlemoyer and Collins' approach induces a sentence-to-logical-form mapping grammar in the first step, followed by the learning of a probabilistic model that assigns a distribution over parses under the grammar. It differs in so far as it employs structure learning as a central part of the learning process. The experimental results suggest it is has precision and recall rates of 96% and 79% respectively.

Unsupervised approaches exist for information extraction, Banko et al. [2] developed the Open Information Extraction (OIE) which was implemented in the TextRunner system. OIE proposes an approach to extract relational tuples without any human input, thereby it requires just a single data-driven pass over the text corpus. The TextRunner system operates by first running a self-supervised learner, which outputs a classifier for labelling candidate extraction as trustworthy or not. Next, it executes a single pass over the corpus in order to extract tuples, which are evaluated by the classifier. Last, each trustworthy tuple receives a probability. The TextRunner system succeeds to outperform a similar system in precision, while scoring the same recall.

A unsupervised apsroach for shallow semantic processing has been developed by Lin and Pantel [9]. They propose a system for Discovery of Inference Rules from Text (DIRT). DIRT leverages information obtained through dependency trees. If the same set of words is linked by two paths, than the system assumes the meaning is similar and an inference rule for each pair of similar paths is generated.

4 Conclusion

In this deliverable, we firstly introduce the major definitions and concepts which provide the theoretical background of Unsupervised Semantic Parsing (USP). Then we describe the ideas behind USP, which try to build clusters of syntactic variations of the same meaning. We also go into the different parts, approaches and algorithms behind the concept as well as the detail on the implementation of USP. The goal of the task T3.4 is to extend existing approach to multiple languages and test if clustering of syntactic variations across languages but of the same meaning can be achieved. In the subsequent deliverable D3.4.2, we will provide the development, implementation and testing of extensions to USP-techniques from D3.4.1 for cross-lingual USP.

References

[1] Hiyan Alshawi and Richard Crouch. **Monotonic semantic interpretation**. In Proceedings of the 30th annual meeting on Association for Computational Linguistics, ACL '92, pages 32–39, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[2] Michele Banko. **Open Information Extraction for the Web.** PhD thesis, University of Washington, 2009.

[3] Neil Coffey. **Memory usage of java strings and string-related objects.**

[4] Jeffrey Dean and Sanjay Ghemawat. **Mapreduce: simplified data processing on large clusters.** Commun. ACM, 51:107–113, January 2008.

[5] W.N. Francis and H. Kucera. Brown Corpus Manual, 1979. www.hit.uib.no/icame/brown/bcm.html.

[6] D. Jurafsky and J. H. Martin. **Speech and language processing.** Prentice Hall, 2000.

[7] Stanley Kok and Wen tau Yih. **Extracting product information from email receipts using markov logic.** 2009.

[8] D. Koller and N. Friedman. **Probabilistic graphical models: principles and techniques.** Adaptive computation and machine learning. MIT Press, 2009.

[9] Dekang Lin and Patrick Pantel. **Dirt - discovery of inference rules from text.** In In Proceedings of the ACMSIGKDD Conference on Knowledge Discovery and DataMining, pages 323–328, 2001.

[10] Christopher D. Manning Marie-Catherine de Marneffe. **Stanford typed dependencies manual,** revised for stanfordÊparser v. 1.6.3 edition, June 2010.

[11] Saif Mohammad, Bonnie Dorr, and Graeme Hirst. Computing word-pair antonymy. **In Proceedings of the Conference on EmpiricalMethods in Natural Language Processing,** EMNLP '08, pages 982–991, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[12] J. Pearl. Probabilistic reasoning in intelligent systems: networks of plausible inference. **The Morgan Kaufmann series in representation and reasoning.** Morgan Kaufmann Publishers, 1988.

[13] Daniel Lowd Hoifung Poon Matthew Richardson Parag Singla Pedro Domingos, Stanley Kok. Markov logic. **In Raedt, L.D., Frasconi, P., Kersting, K., Muggleton, S., eds.: Probabilistic Inductive Logic Programming.**, Volume 4911 of Lecture Notes in Computer Science.:92Ð117, 2008.

[14] Hoifung Poon and Pedro Domingos. http://alchemy.cs.washington.edu/papers/poon09/.

[15] Hoifung Poon and Pedro Domingos. Unsupervised semantic parsing. **In Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing**: Volume 1 - Volume 1, EMNLP '09, pages 1–10, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[16] MatthewRichardson and Pedro Domingos. **Markov logic networks.** Machine Learning, 62(1):107–136, 2006-02-01.

[17] Gertjan Van Noord, Gosse Bouma, Rob Koeling, and Mark-Jan Nederhof. **Robust grammatical analysis for spoken dialogue systems.** Nat. Lang. Eng., 5:45–93,March 1999.

[18] Yuk Wah Wong and Raymond J. Mooney. **Learning for semantic parsing with statistical machine translation.** In Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, HLT-NAACL '06, pages 439–446, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[19] Luke S Zettlemoyer and Michael Collins. **Learning to map sentences to logical form : Structured classification with probabilistic categorial grammars.** Proceedings of 21st Conference on Uncertainty in Artificial Intelligence, 5(x):658–666, 2005

