Computing aggregations from linguistic web resources

Jan Dědek
Department of software engineering
Charles University in Prague
Tel: +420-221-914-264
Email: jan.dedek@mff.cuni.cz

Peter Vojtáš
Institute of Computer Science
Academy of Sciences of the Czech Republic
Tel: +420-221-914-239
Email: vojtas@cs.cas.cz

Abstract

Semantic computing aims to connect the intention of humans with computational content. We present a study of a problem of this type: extract information from large number of similar linguistic web resources to compute various aggregations (sum, average,...). In our motivating example we calculate the sum of injured people in traffic accidents in a certain period in a certain region. We restrict ourselves to pages written in Czech language. Our solution exploits existing linguistic tools created originally for a syntactically annotated corpus, Prague Dependency Treebank (PDT 2.0). We propose a solutions which learns tree queries to extract data from PDT2.0 annotations and transforms the data in an ontology. We present a proof of concept of our method. This enables to compute various aggregations over linguistic web resources.

Keywords:
Semantic computing, natural language processing, linguistic extraction, semantic annotation, semantic web services

1 Introduction and motivation

Our motivation is to contribute to semantic computing vision studying a instance of a relevant problem, namely computing aggregations from linguistic web resources. We understand semantic computing as an extension of semantic web services. First step is to have machine processable data (vision of semantic web), then semantically annotated web services and finally integrate this with linguistic methods.

For the Web to scale, tomorrow's programs must be able to share and process data even when these programs have been designed totally independently. It is understood in general that web resources RDFa annotated with an Ontology will be machine processable. Web services provide a standard means of inter operating between different software applications, running on a variety of platforms and/or frameworks. Web services are characterized by their great interoperability and extensibility, as well as their machine-processable descriptions thanks to the use of XML [10]. Semantic Annotations for WSDL and XML Schema[1] allow description of additional semantics of WSDL components. The specification defines how semantic annotation is accomplished using references to semantic models, e.g. ontologies.

To have all of this, somebody has to annotate data and services. Our paper will contribute also to this showing how can be linguistic web resources automatically annotated.

Motivation.

Main motivation of this research is a big number of text pages on the web where there is a big degree of repetition. Good example are pages of public institutions in the Czech repiblic, which have to publish lot of data by law (Act 106/1999 Coll. on free access to information, English version[2]). These reports are hard for machine processing.

We have experimented with two types of such information. First are reports on bankruptcy[3]. One could be interested in an overview of all cases in which a layer acts as an administrator in bankruptcy. A sum of financial reward of such a person in a year could be interesting.

Another example: The Ministry of Interior of the Czech Republic presents on its Web pages[4] also reports from fire departments of several regions of the Czech Republic. These departments are responsible for rescue and recovery after traffic accidents. These reports are rich in information, e.g. where and when an traffic accident occurred, which units helped, how much time it took them to show up on the place of accident, how many people were injured, killed etc. An example of such report can be seen in the Figure 2.

[1] http://www.w3.org/TR/2007/REC-sawsdl-20070828/
[2] http://aitel.hist.no/~walterk/wkeim/files/foia-czech.htm
[3] http://www.justice.cz/cgi-bin/sqw1250.cgi/upkuk/s_i8.sqw
[4] http://www.mvcr.cz/rss/regionhzs.html

In this paper we describe initial experiments with information extraction from traffic accident reports of fire departments in several regions of the Czech Republic. We would like to demonstrate the prospects of using linguistic tools from the Prague school of computational linguistic (described in 3). These experiments are promising, they e.g. enable the summarization of the number of injured people.

Main contributions.

Main contributions of this paper are:

1. Integration of an experimental chain of tools which captures text of web-pages, annotates it linguistically by PDT tools, extracts data and stores the data in an ontology (annotates) and provides semantic computing of aggregations;

2. In the data extraction phase we formulate an inductive logic programming task for learning queries over linguistically annotated trees. This enables to get those sentences and their parts which are relevant to our query

3. Transformation of extracted data to an ontology instance

The paper is organized as follows. In the next chapter we describe the chain of tools for semantic computing. In the chapter 3 we briefly describe third party tools for linguistic annotation. Next we concentrate on learning tree queries over such linguistically annotated data and finally we describe transformation to an ontology instance, which enables the semantic computing.

2 Chain of tools for semantic computing

We present a chain of tools for the linguistic extraction and semantic annotation of linguistic data from text-based web-resources (grammatical sentences in a natural language). This process consists of four steps. The Figure 1 describes it.

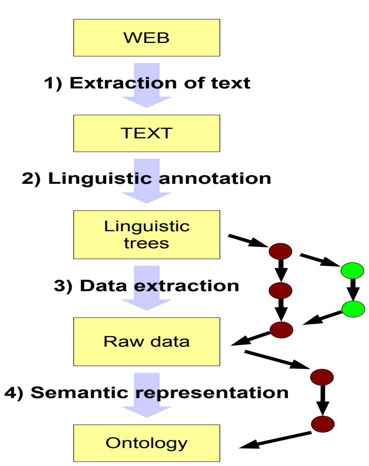


Figure 1. Schema of the extraction process

Figure 2. Example of web-page with a report of a fire department

Notice, more detailed structure of the third and fourth process we focus in this paper.

1. **Extraction of text**
   The linguistic annotating tools process plain text only. In this phase we have to extract the text from the structure of a given web-resource. In this first phase we have used RSS feed of the fire department web-page. From this we have obtained URLs of particular articles and we have downloaded them. Finally we have extracted the desired text (see highlighted area in the Figure 2) by means of a regular expression. This text is an input for the second phase.

   Let us notice, that it is not always that easy to have an RSS feed (e.g. above mentioned data on bankruptcy). In this case we use more sophisticated methods of web information extraction [2].

2. **Linguistic annotation**
   In this phase the linguistic annotation tools process the extracted text and produce corresponding set of dependency trees representing the deep syntactic structure of individual sentences. We have used third party linguistic tools described in the section 3 for this task.

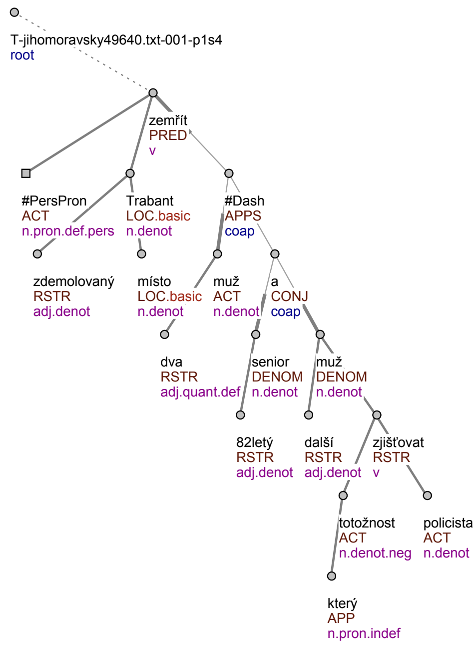


Figure 3. Example of a tectogrammatical tree

3. **Data extraction**
   We use the structure of tectogrammatical (i.e. deep syntactic) dependency trees to extract relevant data. An ILP procedure for this is one of main contributions of this paper, see section 4 for more details.

4. **Semantic representation**
   This phase consists of transformation or conversion of the extracted data to the desired ontology format, see section 5 for more details.

3 PDT linguistic tools for automatic linguistic annotation of texts

In this section we will describe the linguistic tools that we have used to produce linguistic annotation of texts. These tools are being developed in the Institute of Formal and Applied Linguistics[5] in Prague, Czech Republic. They are publicly available – they have been published on a CD-ROM under the title PDT 2.0 [4] (first five tools) and in [5] (Tectogrammatical analysis). These tools are used as a processing chain and at the end of the chain they produce tectogrammatical [6] dependency trees. The Table 1 shows some details about these tools.

1. **Segmentation and tokenization** consists of tokenization (dividing the input text into words and punctuation) and segmentation (dividing a sequences of tokens into sentences).

2. **Morphological analysis** assigns all possible lemmas and morphological tags to particular word forms (word occurrences) in the text.

3. **Morphological tagging** consists in selecting a single pair lemma-tag from all possible alternatives assigned by the morphological analyzer.

4. **Collins' parser – Czech adaptation** [1]

   Unlike the usual approaches to the description of English syntax, the Czech syntactic descriptions are dependency-based, which means, that every edge of a syntactic tree captures the relation of dependency between a governor and its dependent node. Collins' parser gives the most probable parse of a given input sentence.

5. **Analytical function assignment** assigns a description (*analytical function* – in linguistic sense) to every edge in the syntactic (dependency) tree.

6. **Tectogrammatical analysis** produces linguistic annotation at the tectogrammatical level, sometimes called "layer of deep syntax". Such a tree can be seen on the Figure 3. Annotation of a sentence at this layer is closer to meaning of the sentence than its syntactic annotation and thus information captured at the tectogrammatical layer is crucial for machine understanding of a natural language [5].

[5]http://ufal.mff.cuni.cz

Table 1. Linguistic tools for machine annotation

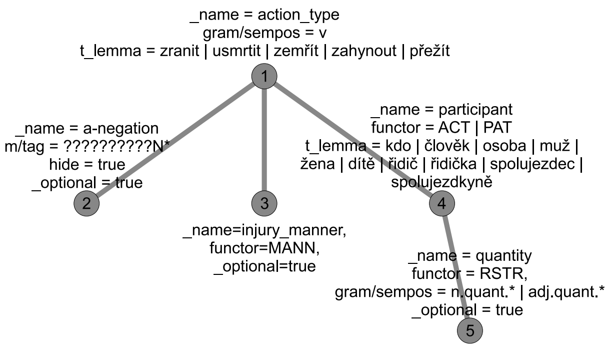
Name of the tool	Results (proclaimed by authors)
Segmentation and tokenization	precision(p): 98,0%, recall(r): 91,4%
Morphological analysis	2,5% unrecognized words
Morphological tagging	93,0% of tags assigned correctly
Collins' parser (Czech adapt.)	precision: 81,6%
Analytical function assignment	precision: 92%
Tectogrammatical analysis [5]	dependencies p: 90,2%, r: 87,9% f-tags p: 86,5%, r: 84,3%

4 Data extraction - querying linguistic trees

Having finished two initial preparatory steps - web crawling with text extraction and linguistic annotation (using third party tools) - we are facing main problem of semantic computing over such data.

Formally, starting with web resource with some URL, we recognized a traffic accident with URI U. First phase gave us sentences

$$S_1, \ldots, S_j$$



Transcript:

zranit	usmrtit	zemřít	zahynout	přežít
to injure	to kill	to die	to wane	to survive

kdo	člověk	osoba	muž	žena	dítě
somebody	(hu)man	person	man	woman	child

řidič	řidička	spolujezdec	spolujezdkyně
driver	woman driver	passenger	woman passenger

Figure 4. Netgraph query – extract rule.

this U consists of. Second phase gave us corresponding tectogramatic trees (see e.g. Figure 3) of these sentences

$$t_1, \ldots, t_j.$$

In a sense, these trees contain already semantic information (annotated with respect to specific linguistic ontology). Moreover, this semantic information is a complete one, it annotates all the information in those sentences. They contain information about the place where the accident occurred, which types of car were involved, which rescue unit arrived and when, number and degree of injuries,... For our specific task (query Q) "find the number of injured", we need only a part of these trees. Moreover, our (human) formulation of the query Q usually does not correspond directly to linguistic annotation. Our task here is to find a tree t_k (possibly several trees) and subtree(s)

$$t_Q \lhd t_k$$

such that t_Q contains number of injured people (or more exactly, from t_Q we can more easily learn the number of injured people).

Our data extraction method is based on extraction rules. These rules correspond to query requests of Netgraph application. The Netgraph application [7] is a linguistic tool used for searching through a syntactically annotated corpus of a natural language. It was originally developed for searching the analytical and tectogrammatical levels of the Prague Dependency Treebank, a richly syntactically annotated corpus of Czech [4]. Netgraph queries are written in a special query language. An example of such Netgraph query can be found in the Figure 4. The Netgraph is a general tool for searching trees, it is not limited only to the trees in the PDT format.

In our application we use it for searching the tectogrammatical trees provided by a set of language processing tools described in the previous chapter. The tectogrammatical trees have a very convenient property of containing just the type of information we need for our purpose, namely the information about inner participants of verbs - actor, patient, addressee etc.

Let us note, that tectogramatic trees are a specific form of XML data, following specific XML Schema description. In general we can use a native XML repository and query language. Nevertheless, for our initial experiments and small number of data, it is more convenient use original PDT repository and Netgraph querying.

Using Netgraph we can query both analytical and tectogramatical level of linguistic annotations. It turns out, that the tectogrammatical (deep syntactic) level of representation is more suitable for our purpose than the analytical (surface syntactic) level of representation of the structure of each sentence. The inner participants (actor, patient, addressee etc.) provide much more reliable information about the actual meaning of the sentence than the syntactic roles (subject, object etc.). The analytical level roles of a subject or object are misleading especially in the case of passive sentences, where usually the subject of the sentence corresponds to the patient or addressee while the actor is expressed by an object of the passive sentence. In these cases it would be necessary to develop (for the analytical level representation) some kind of algorithm analyzing these cases and providing the assignment of proper roles to individual words, while at the tectogrammatical level we get the desired information directly. We cannot exclude, that both analytical and tectogramatical can be useful, here we decided to use only the second.

4.1 Netgraph query learning procedure

As motivated in the beginning of this chapter, having a query Q we would like to get Netgraph query like in the Figure 4. Of course, the more automatic procedure of Netgraph query formulation, the better.

So far the process of building up the extraction rules is heavily dependent on skills and experience of a human designer. Fulfillment of this process is quite creative task. Such a human assisted query learning has served in previous research as a proof of concept. In this paper we would like to make a step toward automating this phase and design automated procedures.

The procedure of learning the Netgraph query is demonstrated in the Figure 5. One obvious preposition of this learning procedure is that we have a collection of learning texts.

Assume we have a query Q and a collection $\mathcal{C}$ of traffic accident reports with URI $U_1, \ldots, U_i, \ldots, U_n$. For $i =$

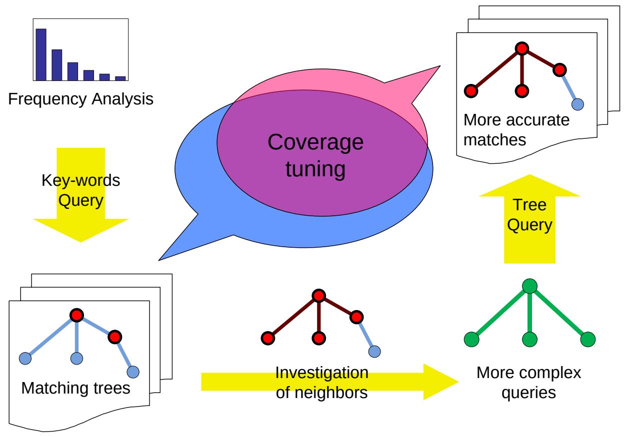
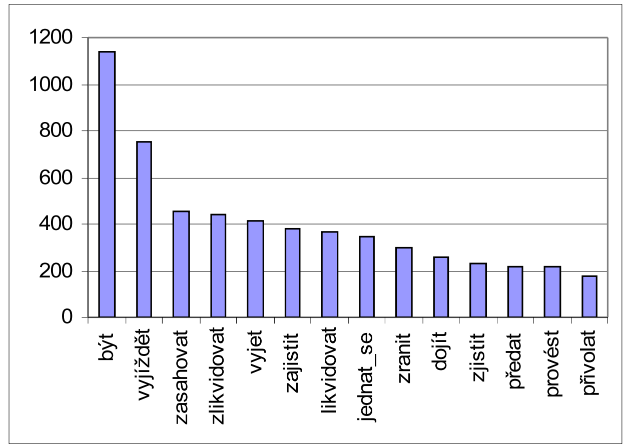


Figure 5. Schema of the query learning procedure



Transcript:
být – to be, *vyjíždět* – to drive out, *zasahovat* – to intervene,
(z)likvidovat – to liquidate, *vyjet* – to rush off, *zajistit* – to ensure,
jdnat se – to deal, *zranit* – to injure, *dojít* – to reach, *zjistit* – to find out,
předat – to hand over, *provést* – to make, *přivolat* – to call in

Figure 6. Frequency analysis of verbs (most frequented only).

$1, \ldots, n$, U_i consists of sentences

$$S_1^i, \ldots, S_{n_i}^i$$

and we have corresponding tectogramatic trees

$$t_1^i, \ldots, t_{n_i}^i.$$

Of course a part of collection $\mathcal{C}_l$ serves as learning set and a part $\mathcal{C}_t$ as a testing set. Moreover we have learning examples (trees) manually classified into two classes, $\mathbb{C}_0$ are trees of sentences with no relevant information on number of injured people and $\mathbb{C}_1$ are relevant trees.

The procedure starts with frequency analysis of words (their lemmas) occurring in these texts. Especially frequency analysis of verbs is very useful — meaning of a clause is usually strongly dependent on the meaning of corresponding verb. An example of frequency analysis of our data can be seen on the Figure 6.

Frequency analysis helps the designer to choose some representative words (**key-words**) that will be further used for searching the learning text collection. Ideal choice of key-words would cover a majority of sentences that express the information we are looking for and it should cover minimal number of the not-intended sentences (maximization of relevance). An initial choice need not be always sufficient and the process could iterate.

Our experiments show that an upper τ_u and lower τ_l threshold can be found, such that relevant verb v frequency $f_{\mathcal{C}_l}(v)$ with respect to collection $\mathcal{C}_l$ fulfills $\tau_u \geq f_{\mathcal{C}_l}(v) \geq \tau_l$.

This part can be automated as follows. Take NL formulation of the query Q and select verbs. You get Q_{verbs} = `usmrtit(kill), zranit(injure)`. Here we would need an extension of (Czech) Word Net [9] to get neighboring verbs, like `utrpet(suffer), zemrit(die)` and get expanded set $Q_{verbsExp}$.

Put $\tau_u = \max\{f(v) : v \in Q_{verbsExp}\}$ and $\tau_l = \min\{f(v) : v \in Q_{verbsExp}\}$. Take the set of all verb candidates

$$V = \{v : \tau_u \geq f_{\mathcal{C}_l}(v) \geq \tau_l\}.$$

A Netgraph query gives me all tectogramatic trees T, which contain $v \in V$. From $T \cap \mathbb{C}_1$ we would like to learn more.

Investigating trees From the human point of view (as we have experimented with this), next step of the procedure consists in investigating trees of sentences covered by key-words. System responds with a set of **matching trees**. The designer examines corresponding syntactic trees — looks for the position of key-words and their matching **neighbors** in the trees.

After that the designer can formulate an initial (Netgraph) **tree query** and he or she can compare result of the Netgraph query with the coverage of key-words. Based on this he or she can reformulate the query and gradually **tune** the query and the **query coverage**.

There are two goals of the query tuning. The first goal is maximization of the relevance of the query. The second goal is to involve all important tree-nodes to the query. This second goal is important because the **complexity of the query** (number of involved nodes) makes it possible to extract more complex information. For example see the query on the Figure 4 — each node of it keeps different kind of information.

In an automated version (not running so far, here we present a design only) we formulate a task for Inductive

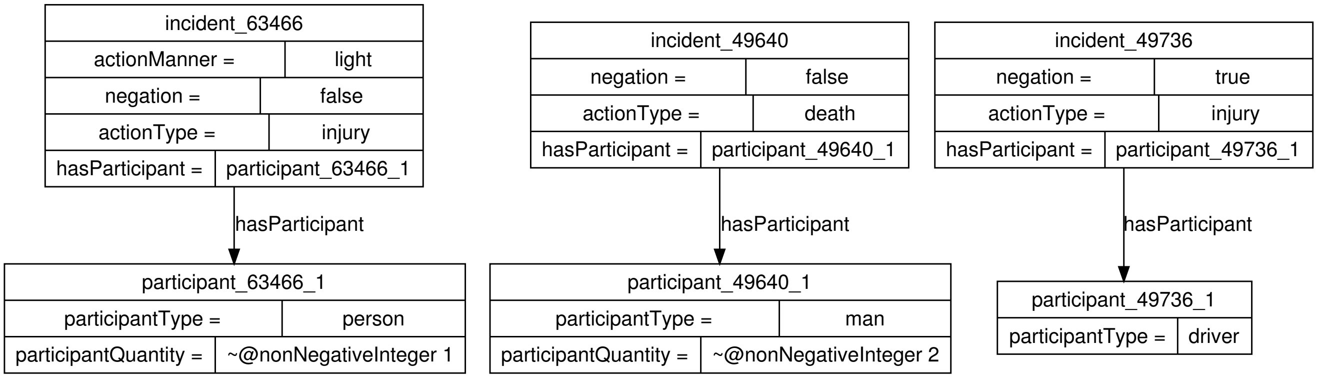


Figure 7. Example of three instances of incident ontology.

logic programming (ILP)[6] [8]. The main reason for choosing ILP for learning is the fact that it is quite efficient in learning from structured data like graphs and trees e.g. in chemistry [3].

Our ILP task, given query Q, is formulated by a set of positive examples

$$E^+ = \{t^i_k : t^i_k \in T \cap \mathbb{C}_1\}$$

and a set of negative examples

$$E^- = \{t^i_k : t^i_k \in T \cap \mathbb{C}_0\}$$

(of course after some reformulation of tree data into a logical language) and a set of logic program rules B, called background knowledge. So far B consists only of structural properties describing subtrees in our logical representation of trees.

Result of this ILP task is a set of trees

$$t^i_{k_i,Q} \lhd t^i_{k_i}.$$

Our experience show, that resulting Netgraph query

$$Merge(t^i_{k_i,Q} : i = 1, \ldots, n)$$

gives best results. Here the merge operation means

- make a node union of trees (nodes that appear in some results are optional)

- put different terms on the same tectogramatic position into disjunction (e.g. `zranit|usmrtit|...`).

One example of such a merged tree, which serves as a Netgraph query is depicted in Figure 4.

4.2 Results of the extraction

We have evaluated the extraction rule shown in the Figure 4 by using the set of 800 texts of news of several Czech fire departments. There were about 470 sentences matching the rule and we found about 200 numeral values contained in the node number 5. This extraction rule (from the Figure 4) is a result of a learning procedure described in the section 4.1.

Small part of results of the extraction is shown in the Figure 8. This result contains three matches of the extraction rule in three different articles. Each match is closed in the `<Match>` element and each contains values of some linguistic attributes closed inside the `<Value>` elements. Each value comes from some of the nodes of the extraction rule. Name of corresponding query node is saved in the `variable_name` attribute of the `<Value>` element. Value identified as `action_type` specifies the type of the action. So in the first and in the third case there was somebody injured (*zranit* means to injure in Czech) and in the

```
<QueryMatches>
  <Match root_id="T-vysocina63466.txt-001-p1s4" match_string="2:0,7:3,8:4,11:2">
    <Sentence>
      Při požáru byla jedna osoba lehce zraněna - jednalo se
      o majitele domu, který si vykloubil rameno.
    </Sentence>
    <Data>
      <Value variable_name="action_type" attribute_name="t_lemma">zranit</Value>
      <Value variable_name="injury_manner" attribute_name="t_lemma">lehký</Value>
      <Value variable_name="participant" attribute_name="t_lemma">osoba</Value>
      <Value variable_name="quantity" attribute_name="t_lemma">jeden</Value>
    </Data>
  </Match>
  <Match root_id="T-jihomoravsky49640.txt-001-p1s4" match_string="1:0,13:3,14:4">
    <Sentence>
      Ve zdemolovaném trabantu na místě zemřeli dva muži - 82letý senior
      a další muž, jehož totožnost zjišťují policisté.
    </Sentence>
    <Data>
      <Value variable_name="action_type" attribute_name="t_lemma">zemřít</Value>
      <Value variable_name="participant" attribute_name="t_lemma">muž</Value>
      <Value variable_name="quantity" attribute_name="t_lemma">dva</Value>
    </Data>
  </Match>
  <Match root_id="T-jihomoravsky49736.txt-001-p4s3" match_string="1:0,3:3,7:1">
    <Sentence>Čtyřiatřicetiletý řidič nebyl zraněn.</Sentence>
    <Data>
      <Value variable_name="action_type" attribute_name="t_lemma">zranit</Value>
      <Value variable_name="a-negation" attribute_name="m/tag">VpYS---XR-NA---
      </Value>
      <Value variable_name="participant" attribute_name="t_lemma">řidič</Value>
    </Data>
  </Match>
</QueryMatches>
```

Figure 8. Example of the result of the extraction procedure.

[6] `http://www-ai.ijs.si/~ilpnet2/apps/index.html`

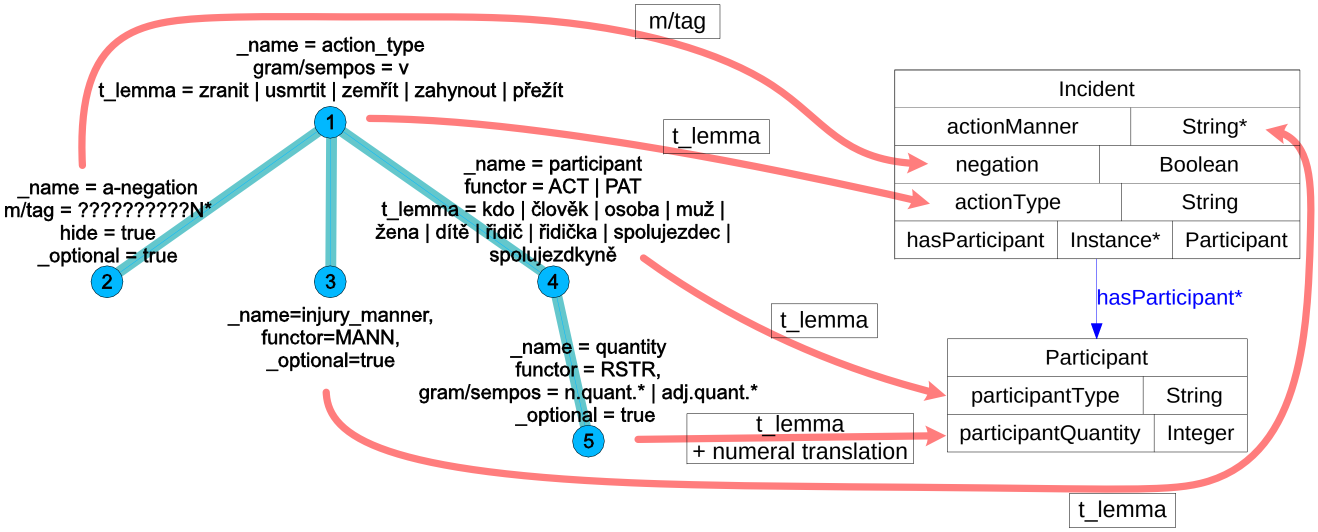


Figure 9. Semantic interpretation of Netgraph query – mapping of nodes of the query tree to the ontology.

second case somebody died (*zemřít* means to die in Czech). Value identified as `a-negation` contains the information about negation of the clause. So we can see that the participant of the third action was **not** injured. Values identified as `participant` and `quantity` contain information about a participants of the action. Value identified as `participant` specifies the type of the participants and value identified as `quantity` holds a number of these participants. So in the first action only a single (*jeden*) person (*osoba*) was injured, in the second action two (*dva*) men (*muž*) died and in the third action a driver (*řidič*) was not injured.

5 Semantic representation

In this section we will discuss semantic interpretation of extracted data. After the designer have successfully formulated the Netgraph query, he or she have to supply semantic interpretation of the query. This interpretation expresses how to transform matching nodes of the query (and the available linguistic information connected with the nodes) to the format of output ontology. Complexity of the transformation varies form simple (e.g. setting value of a data-type property to the value of some linguistic attribute) to complex. For example a translation of a numeral to a number can be seen in the Figure 7 and also in the Figure 9 (data-type property *participantQuantity*).

Figure 10 shows schema of a small ontology we have designed for interpretation of the extracted data (an example on the extracted data has been shown on the Figure 8). This ontology consists of two classes (or concepts): *Incident* and *Participant*. These classes are connected with a relation *hasParticipant*. There are also some data-type properties (*actionType*, *actionManner*, *negation*, *participantType*, *participantQuantity*) to cover the extracted data. Three individuals (or instances) of this ontology are shown on the Figure 7. These individuals represent exactly the same data as was shown in the Fingure 8.

Results of the above Netgraph query are trees, which are instances of the query tree. Mapping of nodes of the query tree to the ontology (as shown on the Figure 9) will provide mapping of the results.

This mapping can be done by another ILP task, similar to previous. Learning instances of query result trees will be divided into several classes, corresponding to ontology classes (or concepts).

After we saved all the extracted data in our ontology, we can formate SPARQL query that gives us a table of fatalities occurred in the fire-department's articles. This SPARQL

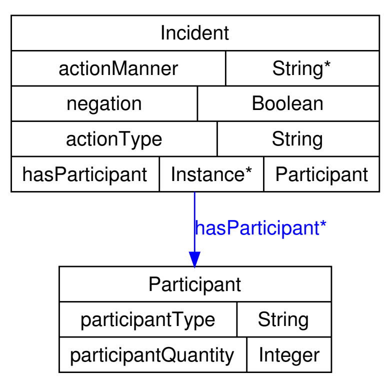


Figure 10. Schema of incident ontology.

```
SELECT ?participant ?participant_type ?quantity
WHERE {
?action   rdf:type          :Incident;
          :actionType       "death";
          :negation         false;
          :hasParticipant   ?participant.

?participant :participantType ?participant_type.

OPTIONAL {
?participant :participantQuantity ?quantity.
}}
```

Figure 11. SPARQL query returning a table of fatalities.

query is shown in the Figure 11.

SPARQL does not enable aggregations, we have to use SQL aggregation query over SPARQL result.

6 Conclusion

We have presented a proposal of and experiments with a system for semantic computing of information from Czech text on Web pages. Our system relies on linguistic annotation tools from PDT [4] and the tree querying tool Netgraph [7]. Our contributions are an experimental chain of tools which captures text of web-pages, annotates it linguistically by PDT tools, extracts data and stores the data in an ontology and hence enables semantic computing. In the third phase – data extraction – we formulate an inductive logic programming task over linguistically annotated data. Finally we describe transformation of these data to an ontology instance. Our initial experiments verified these methods and tools.

Acknowledgment
This work was partially supported by Czech projects 1ET100300517 and 1ET100300419.

References

[1] M. Collins, J. Hajič, E. Brill, L. Ramshaw, and C. Tillmann. A Statistical Parser of Czech. In *Proceedings of 37th ACL Conference*, pages 505–512, University of Maryland, College Park, USA, 1999.

[2] A. Eckhardt, T. Horvath, D. Maruscak, R. Novotny, and P. Vojtas. Uncertainty issues in automating process connecting web and user. In *In URSW-2007, Eds. Paulo C. G. da Costa et al. CEUR Workshop Proceedings,*, number 327, 2008.

[3] N. Fonseca, R. Rocha, R. Camacho, and F. Silva. Efficient data structures for inductive logic programming. In *Proceedings of ILP 2003*, number 2835 in Lecture Notes In Computer Science, pages 130–145. Springer-Verlag Berlin Heidelberg, 2003.

[4] J. Hajič, E. Hajičová, J. Hlaváčová, V. Klimeš, J. Mírovský, P. Pajas, J. Štěpánek, B. Vidová-Hladká, and Z. Žabokrtský. Prague dependency treebank 2.0 cd-rom. Linguistic Data Consortium LDC2006T01, Philadelphia 2006, 2006.

[5] V. Klimeš. Transformation-based tectogrammatical analysis of czech. In *Proceedings of the 9th International Conference, TSD 2006*, number 4188 in Lecture Notes In Computer Science, pages 135–142. Springer-Verlag Berlin Heidelberg, 2006.

[6] M. Mikulová, A. Bémová, J. Hajič, E. Hajičová, J. Havelka, V. Kolářová, L. Kučová, M. Lopatková, P. Pajas, J. Panevová, M. Razímová, P. Sgall, J. Štěpánek, Z. Urešová, K. Veselá, and Z. Žabokrtský. Annotation on the tectogrammatical level in the prague dependency treebank. annotation manual. Technical Report 30, ÚFAL MFF UK, Prague, Czech Rep., 2006.

[7] J. Mírovský. Netgraph: A tool for searching in prague dependency treebank 2.0. In J. Hajič and J. Nivre, editors, *Proceedings of the Fifth Workshop on Treebanks and Linguistic Theories (TLT)*, number 5, pages 211–222, Prague, Czech rep., 2006.

[8] S. Muggleton. Inductive logic programming. *New Generation Computing*, 8:295–317, 1991.

[9] K. Pala and P. Smrž. Building czech wordnet. *Romanian Journal of Information Science and Technology*, 2004(7):79–88, 2004.

[10] W3C. Web services activity statement, 2008.