Charles University in Prague

Faculty of Mathematics and Physics

DOCTORAL THESIS

Jan Dědek

Semantic Annotations

Department of Software Engineering

Supervisor of the doctoral thesis: Prof. RNDr. Peter Vojtáš, DrSc.

Study programme: Software Systems

Prague 2012

Acknowledgments

I wish to thank my supervisor ...

Název práce: Sémantické Anotace

Autor: Jan Dědek

Katedra: Katedra Softwarového Inženýrství

Vedoucí disertační práce: Prof. RNDr. Peter Vojtáš, DrSc.

Abstrakt:

Klíčová slova:

Title: Semantic Annotations

Author: Jan Dědek

Department: Department of Software Engineering

Supervisor: Prof. RNDr. Peter Vojtáš, DrSc.

Abstract:

Keywords:

Contents

1. Introduction

Four separate topics are presented in the present thesis and the discipline of Information Extraction is the central point of them. Each topic represents one particular aspect of the Information Extraction discipline.

The first two topics are focused on our information extraction methods based on deep language parsing. The first topic relates to how deep language parsing was used in our first method in combination with manually designed extraction rules.

The second topic deals with an alternative extraction method based on machine learning. An inductive procedure was developed based on Inductive Logic Programming, which allows automated learning of extraction rules from a learning collection.

The idea of the Semantic Web was the strongest motivation of our research from the very beginning. We wanted to exploit information extraction techniques to speed up the semantic web evolution. The third topic of the thesis presents even more than that. The core of the extraction method was experimentally reimplemented using semantic web technologies. Therefore not only the result of information extraction but also the extraction procedure itself is realized using semantic web technologies. The main advantage of this approach is the possibility to save the extraction rules in so called shareable extraction ontologies.

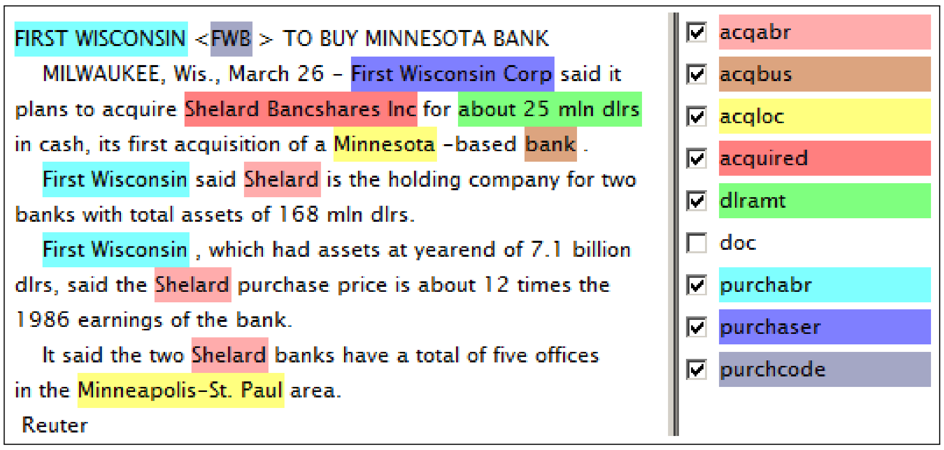
The last topic of this thesis is the most distant from the original information extraction topic. We have included it because it represents an important part of our research and considerable effort was spent on it. The topic deals with document classification and fuzzy logic. We are investigating the possibility of using information obtained by information extraction techniques to document classification. Our implementation of so called Fuzzy ILP Classifier was experimentally used for the purpose of document classification.

1.1 Motivation

The basic motivation of our research can be illustrated with three images or schemas that are presented in Figures 1.1, 1.2 and 1.3. The first two figures show some texts with several pieces of information decorated in it. If you show such images to a human, he or she will be shortly able to find such pieces of information in any other text of the same kind. But can this relatively simple task do a computer as well? Figure 1.3 represents our first ideas when we started to look for the answer. The figure shows a linguistic tree obtained by automated linguistic analysis of the last sentence of the second figure (Figure 1.2). It already contain lots of indications (decorated with orange tags) of where to look for the wanted piece of information, in this case, the amount of 8,000 Czech Crowns representing the total damage sought by the accident reported in the text.

The main motivation for creating our extraction methods was an attempt to use deep linguistic analysis for this task. Especially for the Czech language with free word order this seemed reasonable. It is much more straightforward to design extraction rules on the basis of linguistic dependency trees than to struggle with the surface structure of text. In a dependency tree, the position of a word is determined by its syntactic (analytical trees) or even semantic role (tectogrammatical trees). So the extraction rules might not be dramatically affected by minor variations of the word order.

Besides that information extraction and annotation is very interesting and challenging problem, it is also particularly useful. This period can be characterized by information overload and information extraction can provide partial answer to that. It provides fine


FIRST WISCONSIN <FWB > TO BUY MINNESOTA BANK
MILWAUKEE, Wis., March 26 - First Wisconsin Corp said it plans to acquire Shelard Bancshares Inc for about 25 mln dlrs in cash, its first acquisition of a Minnesota -based bank .
First Wisconsin said Shelard is the holding company for two banks with total assets of 168 mln dlrs.
First Wisconsin , which had assets at yearend of 7.1 billion dlrs, said the Shelard purchase price is about 12 times the 1986 earnings of the bank.
It said the two Shelard banks have a total of five offices in the Minneapolis-St. Paul area.
Reuter

☑ acqabr
☑ acqbus
☑ acqloc
☑ acquired
☑ dlramt
☐ doc
☑ purchabr
☑ purchaser
☑ purchcode

Figure 1.1: Corporate Acquisition Events annotations

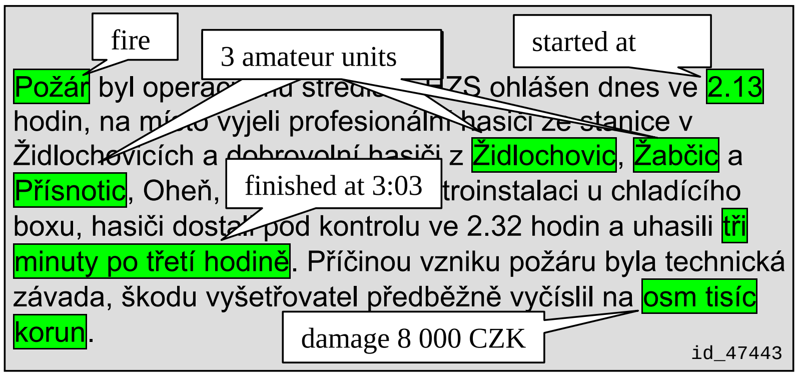
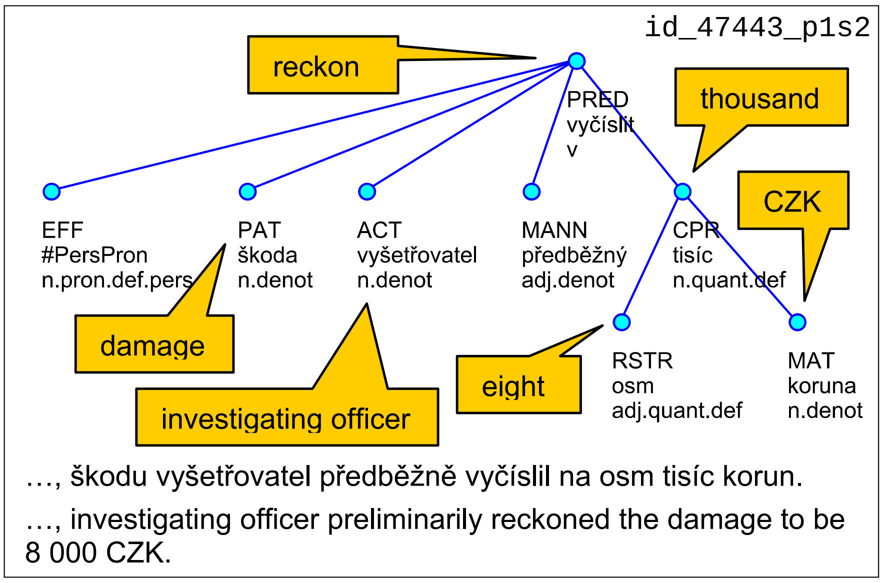

Požár byl operač[illegible] střed[illegible]ZS ohlášen dnes ve 2.13 hodin, na m[illegible]to vyjeli profesionální hasiči ze stanice v Židlochovicích a dobrovolní hasiči z Židlochovic, Žabčic a Přísnotic, Oheň, [illegible]troinstalaci u chladícího boxu, hasiči dosta[illegible] pod kontrolu ve 2.32 hodin a uhasili tři minuty po třetí hodině. Příčinou vzniku požáru byla technická závada, škodu vyšetřovatel předběžně vyčíslil na osm tisíc korun.

Figure 1.2: Fireman Events annotations



…, škodu vyšetřovatel předběžně vyčíslil na osm tisíc korun.
…, investigating officer preliminarily reckoned the damage to be 8 000 CZK.

Figure 1.3: Example of a linguistic tree of one analyzed sentence.

grained indexing of documents, which supports precise search and document filtering. Navigation within individual documents can be faster and reading can be more effective. Other software programs can use the extracted information and perform additional computations resulting in summaries and answers integrated from different sources. The effort in this direction will hopefully culminate in the realization of the idea of the Semantic Web, when all the information will be available in a machine workable form and the whole (semantic) web could be used as a huge knowledge base.

1.2 Main Goals and Contributions

To be written!!!

—-

The main *contributions* presented in this chapter are *formal models*, prototype *implementation* and experimental *evaluation* of the Fuzzy ILP Classifier.

1.3 Organization

Rather than presenting individual topics or approaches of this thesis separately in distinct chapters, we decided to organize this document according to common aspects of these approaches and to dedicate individual chapters to these aspects instead of individual approaches. This way, all the (four) approaches are described in parallel in each chapter.

Chapter 2 provides definitions of the individual problems and consequent tasks solved in this thesis. Chapter 3 contains description of the most related work of other scientists. Chapter 4 introduces the most important third party tools and resources that were used in our research. Chapter 5 describes solutions, used models and methods of the individual approaches presented in this thesis. Chapter 6 provides details about implementation of all the approaches. Chapter 7 describes all datasets that were used in our experiments. Chapter 8 describes all experiments that we performed mostly for evaluation of the approaches. Finally, Chapter 9 concludes the thesis.

2. Problems and Consequent Tasks Definitions

As already said in introduction, four separate topics are presented in this thesis. This chapter provides definitions of the main problems connected with these topics and consequent tasks that have to be addressed to solve these problems.

2.1 Information Extraction

2.1.1 The Problem

The basic problem addressed by information extraction approaches can be formulated as follows. We have a large collection of texts (or a source of texts) and we want to have a machine readable (understandable, workable) or structured form of the information present in these texts. Why? Because additional postprocessing of the information is needed.

Let us for example have texts about acquisition events. Each text describes a single acquisition event and you can find answers to following questions in these texts.

- What was the object of the acquisition?
- Who was the buyer?
- What was the deal amount?
- What was the acquisition date?

If we put the corresponding information into a relational table then we can easily obtain statistics like numbers of acquisitions per month of the year or a list of ten most valuable (largest deal amount) acquired subjects, etc. This would be impossible if the information were kept in the textual form only.

Document search and indexing is another important purpose of information extraction. Let us imagine a person interested in articles about acquisitions made in January 2009 where the deal amount was between 100 and 500 million dollars. Keyword search and indexing can not satisfy this need accurately. But if we have corresponding machine readable information, it is easy to create a simple search engine supporting this kind of queries.

The idea of the Semantic Web [Berners-Lee *et al.*, 2001] brings us to even a bigger problem that could be solved by information extraction approaches. On the Semantic Web, as much information as possible should be in structured machine readable form. But absolute majority of ordinary content of the present day web is understandable only by humans. Machines can use only very limited part of the present day web. This issue is often called "Semantic Web Bottleneck" [Konstantinou *et al.*, 2010].

2.1.2 Consequent Tasks

The definition of the problem presented in the previous section was general. We did not specify any particular kind of information to be extracted or target structure for capturing it. This is exactly the point, where different kinds of information extraction approaches differ. Currently, the variety of information extraction approaches is huge. In this thesis, we will focus on a small subset only.

It was already mentioned in the previous section that the source information is captured in plain text. Let us specify it more precisely:

We are interested in cases where information is expressed in plain text in natural language. We do not consider additional structural information or formatting instructions that may be potentially available e.g. through HTML or XML tags.

We are primarily interested in texts consisting of natural language sentences. We do not consider textual fragments, tables, etc.

These two options were selected mainly based on our personal interest and background. This setting is very close to the topic of natural language understanding, very attractive problem since the very establishment of artificial intelligence. [1] And this setting is also very common in practice, e.g. in news articles, scientific papers, blog posts, official statements and reports, offers, requests, advertisements, etc.

After the specification of input format, let us specify the kind of information to be extracted and the target structure for capturing it. Higher complexity of the extracted information makes the task more difficult. The entire complexity of human language is still far beyond the boundary of machine workability. In practice, extraction of very simple facts represented by plain relations already provides significant help. For example only one relation (e.g. “acquisition”) with four arguments (“object”, “buyer”, “deal amount” and “date”) would be sufficient for capturing the information about acquisition events mentioned above. In this thesis, we concentrate on these simple relational facts, not considering any of the wide range of possibilities that the human language offers like expression of tense, causality, modality, general statements with quantifiers, etc.

Stating that we will extract “simple relational facts” is still not precise enough and several well established information extraction tasks conform to this statement. We will describe them in Sections 2.1.4–2.1.8, but before that, it is necessary to explain how the term “document annotation” will be used in this thesis.

2.1.3 Document Annotation

Document Annotation is a term, which usually refers to the process of putting annotations to a document. In this thesis, the term annotations will always refer to a special kind of annotations used in the GATE framework. These annotations refer to (usually shorter) segments of text and they provide a label (or annotation type) and arbitrary set of features to each such segment (or simply annotation). Each annotation occupies a continuous portion of text between the beginning and the end of the annotation. These annotations can be easily visualized using colored background in text. Different annotation types are usually decorated with different colors; see Figure 1.1 for an example. Section 4.5.1 provides some technical details about this kind of annotations.

There is a very small difference between the process of document annotation and information extraction. In fact they are equivalent in the case we are describing. Because annotations can be reconstructed from extracted relational facts and relational facts can be reconstructed from annotations. There are only two conditions that have to hold true:

(1) It has to be always possible to determine a portion of text representing any extracted fact.

(2) Each annotation has to keep all its relational arguments as annotation features.

Relation name can be saved as annotation label and vice versa.

[1] See e.g. in the Wikipedia article Natural language understanding: `http://en.wikipedia.org/wiki/Natural_language_understanding#History`

2.1.4 Entity Recognition

Entity Recognition or Named Entity Recognition corresponds to the extraction task of identification of significant entities (people, organizations, locations, chemicals, genes, etc.) that are present in text. From the annotation perspective, this is the simplest annotation task: just to mark these entities in text and assign correct labels to them. From the relational perspective, this task corresponds with unary relation extraction.

It can be for example illustrated on following sentence:

> Google just announced that it is acquiring Motorola Mobility.

There are two entities mentioned in this sentence: Google and Motorola Mobility. Both entities have to be extracted and put into correct unary relations, e.g. "organization(Google)" or "company(Google)" depending on the used vocabulary. Or, in the annotation case, they have to be marked in text and annotated with corresponding label ("organization" or "company").

2.1.5 Relation Extraction

Relation Extraction is an extraction task that usually comes after entity recognition. When all the significant entities are identified, the task is to connect together those entities that are connected in text and to assign the correct label (relation name) to that connection. Let us again use the example sentence about Google acquiring Motorola. The extracted relation should be connecting the two entities (in the right order) and the label would be something like: "acquiredBy", "purchasing" or "takingOver" (depending on the system vocabulary; note the dependency between the label and the relation orientation.)

2.1.6 Event Extraction

In literature, Relation Extraction usually refers to binary relations only and Event Extraction is used for relations (events) with more arguments. Individual events have to be correctly identified in text and arguments have to be assigned to them in proper roles. For example an acquisition event can have roles like purchaser, seller, acquired, dollar amount, etc. We have to extend our running example with Motorola to demonstrate event extraction on it:

> Google just announced that it is acquiring Motorola Mobility. The search and online advertising company is buying the company for approximately $12.5 billion (or $40 per share), in cash.

In this case, both sentences would be covered by the acquisition event with attached arguments: purchaser(Google), acquired(Motorola Mobility) and dollar_amount($12.5 billion).

2.1.7 Event Extraction Encoded as Entity Recognition

In practice, there are quite common cases when only a single event is reported in each document. In this case it is not necessary to annotate the exact location of the event in the document and mere identification of event roles is sufficient. Technically, annotation of such events looks like the same as annotation of ordinary entities of an entity recognition task. The difference is only in labels of these entities because they correspond with event roles. A proper example would look like a combination of the examples that we have used for the demonstration of Entity Recognition and Event Extraction tasks. Google, Motorola and '$12.5 billion' will be annotated the same as in Event Extraction – purchaser(Google), acquired(Motorola Mobility), dollar_amount($12.5 billion) – but they will be not linked to any particular event because they belong to the implicit event identified by the current document.

Both manually annotated datasets described in this thesis are of this kind of event extraction; see details in Sections 7.3.2 and 7.3.3.

2.1.8 Instance Resolution

Instance Resolution further extends entity recognition. It aims at linking a particular entity to its unique representative (or identifier) such that the same entities have always the same identifier and, vice versa, different entities have always different identifiers. Disambiguation is the main task that has to be solved by instance resolution.

It can be illustrated on some ambiguous entity, for example "George Bush". General entity recognition system just marks the string and assigns a corresponding label (e.g. person, politician or president – depending on the granularity of the system) to it. In extraction case, such system just puts the string into the corresponding relation.

Instance resolution is more difficult. Instance resolution system has to select the right representative for that entity – George W. Bush (junior) or George H. W. Bush (senior) that will be probably both available in the system's database. Similarly, instance resolution system has to assign the same identifier in cases where for example shortcuts are used, e.g. "George W. Bush" and "G. W. Bush" should be linked to the same identifier.

2.1.9 Summary

The problem of information extraction consists in obtaining machine workable form of information that was previously available in textual form only. We have specified that we are interested only in extraction of simple relational facts and that the extracted facts can be kept either as relational data or they can be of the form of document annotations. Depending on the complexity of extracted information, four basic extraction tasks were defined:

- Entity Recognition,
- Relation Extraction,
- Event Extraction and
- Instance Resolution.

2.2 Machine Learning for Information Extraction

2.2.1 The Problem

Development of an information extraction system is always, more or less, dependent on a concrete domain and extraction tasks. It is always necessary to adapt the system when it should be used in a new domain with new extraction tasks. The difficulty of such adaptation varies. In some cases the whole system has to be redesigned and reimplemented. Many systems are based on some kind of extraction rules or extraction models and only these rules or models have to be adapted in such case. But still, only highly qualified experts can do such adaptation. These experts have to know the extraction system in detail as well as the domain, target data (texts) and extraction tasks. Such qualification is sometimes (e.g. in biomedical domains) unreachable in one person and difficult cooperation of a domain expert with a system expert is necessary.

2.2.2 Consequent Tasks

Usage of machine learning techniques can address this problem in such a way that the system is capable to adapt itself. But a learning collection of example texts with gold standard annotations has to be provided. The effort needed to build such collection is still big and demanding, but such gold standard collection is not dependent on any concrete extraction system and expert knowledge related to extraction systems is not needed for its construction.

Another important purpose of machine learning is that extraction rules or models constructed by machine learning techniques are often more successful than those designed manually.

2.3 Extraction Ontologies

2.3.1 The Problem

Information extraction and automated semantic annotation of text are usually done by complex systems and all these systems use some kind of model that represents the actual task and its solution. The model is usually represented as a set of some kind of extraction rules (e.g., regular expressions), gazetteer lists[2] or it is based on some statistical measurements and probability assertions (classification algorithms like Support Vector Machines (SVM), Maximum Entropy Models, Decision Trees, Hidden Markov Models (HMM), Conditional Random Fields (CRF), etc.)

Before the first usage of an information extraction system, such model is either created by a human designer or it is learned from training dataset. Then, in the actual extraction/annotation process, the model is used as a configuration or as a parameter of the particular extraction/annotation system. These models are usually stored in proprietary formats and they are accessible only by the corresponding system.

In the environment of the Semantic Web it is essential that information is shareable and some ontology based IE systems keep the model in so called extraction ontologies [Embley *et al.*, 2002].

> Extraction ontologies should serve as a wrapper for documents of a narrow domain of interest. When we apply an extraction ontology to a document, the ontology identifies objects and relationships present in the document and it associates them with the corresponding ontology terms and thus wraps the document so that it is understandable in terms of the ontology [Embley *et al.*, 2002].

In practice the extraction ontologies are usually strongly dependent on a particular extraction/annotation system and cannot be used separately. The strong dependency of

[2]See Section 4.5.2 for details about the term gazetteer list.

an extraction ontology on the corresponding system makes it very difficult to share. When an extraction ontology cannot be used outside the system there is also no need to keep the ontology in a standard ontology format (RDF or OWL).

2.3.2 Consequent Tasks

The cause of the problem is that a particular extraction model can only be used and interpreted by the corresponding extraction tool. If an extraction ontology should be shareable, there has to be a commonly used tool that is able to interpret the extraction model encoded by the extraction ontology. For example Semantic Web reasoners can play the role of commonly used tools that can interpret shareable extraction ontologies.

Although it is probably always possible to encode an extraction model using a standard ontology language, only certain way of encoding makes it possible to interpret such model by a standard reasoner in the same way as if the original extraction tool was used. The difference is in semantics. It is not sufficient to encode just the model's data, it is also necessary to encode the semantics of the model. Only then the reasoner is able to interpret the model in the same way as the original tool. If the process of information extraction or semantic annotation should be performed by an ordinary Semantic Web reasoner then only means of semantic inference are available and the extraction process must be correspondingly semantically described.

2.4 Document Classification

2.4.1 The Problem

Similarly to information extraction, document classification solves a problem when we have a large collection of documents and we do not exactly know what the individual documents are about. Document classification helps in cases when we want to just pick such documents that belong to certain category, e.g. to pick only football news from a general collection of sport news articles.

2.4.2 Consequent Tasks

The task of document classification can be formulated as follows. Having a textual document and a (usually small) set of target categories, the task is to decide to which category the document belongs.

In the case of the present thesis, we are especially interested in cases when the set of categories is ordered or, in other words, we can always decide which one of any two categories is higher. This kind of categories is mostly used as rankings and individual categories then correspond to ranking degrees, e.g. ranking of seriousness of a traffic accident, which is used in classification experiments of the present thesis.

3. Related Work

This chapter shortly introduces mostly related work of other researchers. The chapter is split to three main sections. Section 3.3 is dedicated to information extraction approaches, Section 3.2 to extraction ontologies and Section 3.3 to document classification.

3.1 Information Extraction Approaches

3.1.1 Deep Linguistic Parsing and Information Extraction

> The choice of the actual learning algorithm depends on the type of structural information available. For example, deep syntactic information provided by current parsers for new types of corpora such as biomedical text is seldom reliable, since most parsers have been trained on different types of narrative. If reliable syntactic information is lacking, sequences of words around and between the two entities can be used as alternative useful discriminators. [Bunescu, 2007]

Since that time, the situation has improved and deep linguistic parsing is often used even for biomedical texts (because new biomedical corpora are available for retraining of the parsers; see e.g. in [Buyko and Hahn, 2010]) and dependency graphs constitute the fundamental data structure for syntactic structuring and subsequent knowledge extraction from natural language documents in many contemporary approaches to information extraction (see details about individual information extraction systems in Section 3.1.2). Currently, quite a lot of parsers can be used for generation of these structures from natural language texts.

Besides the possibility of using different parsers there are also various language dependency representations (LDR) such as the Stanford [de Marneffe *et al.*, 2006] and CoNLL-X [Johansson and Nugues, 2007] dependencies and the Functional Generative Description (FGD) [Sgall *et al.*, 1986] studied mainly in Prague. All the dependency representations are very similar from the structural point of view; they differ mainly in the number of different dependency kinds they offer. FGD provides also additional node attributes and so called layered approach (see in Section 4.1) and is the only representation available for Czech.

It is also worthy to investigate the impact of usage of different LDRs on the performance of information extraction. Buyko and Hahn [2010] compared the impact of different LDRs and parsers in their IE system, namely Stanford and CoNLL-X dependencies and usage of different trimming operations. Their findings are definitely not supporting one choice against another because one representation was better for some tasks and another one for other tasks, see examples in their paper. More important seems to be the quality of used parser.

3.1.2 IE Systems Based on Deep Language Parsing

We describe in this section several information extraction systems based on deep language parsing. The systems differ greatly in the manner of using LDR.

Rule Based Systems

There are many systems using hand crafted extraction rules based on LDR. These systems need assistance from a human expert that is able to design the extraction rules manually. The advantage of these systems is that there is no need of learning (or training) data collection. For example Fundel *et al.* [2007] used a simple set of rules and the Stanford parser[1] for biomedical relation extraction. Shallow and deep parsers were used by Yakushiji *et al.* [2001] in combination with mapping rules from linguistic expressions to biomedical events.

Similarity Based Systems

Several information extraction systems do some kind of similarity search based on the syntactic structure, e.g. [Etzioni *et al.*, 2008] and [Wang and Neumann, 2007].

Biomedical Domain

The Stanford format is widely used in the biomedical domain (e.g., by Miyao *et al.* [2008], (Yakushiji *et al.* [2001]) or Clegg and Shepherd [2005]).

Classical Machine Learning Systems

Classical machine learning (ML) approaches rely on the existence of a learning collection. They usually use LDR just for construction of learning features for propositional learners like decision trees, neural networks, support vector machines (SVM), etc. Learning features are selected manually when the system is being adapted to new domain or extraction task. For example in [Bunescu and Mooney, 2007], learning features based on so called dependency paths constructed from LDR are used for relation extraction. Similar approach was used in [Buyko *et al.*, 2009] for biomedical event extraction.

[Li *et al.*, 2009] described machine learning facilities available in GATE. This approach is an example of classical adaptation of propositional learners for information extraction. It should be noted that GATE itself does not provide any prebuilt functions for working with LDR but they can be added as they were in our case (see in Section 6.2.2). This is also the software component (GATE Machine Learning PR) to that we have compared our solution.

Inductive Systems

There are also systems using some inductive technique e.g. Inductive Logic Programming (ILP, see also Section 4.7) to induce learning features or extraction rules automatically from learning collection. In these cases it is neither necessary to manually design extraction rules nor select the right learning features. For example Ramakrishnan *et al.* [2007] used the dependency parser MINIPAR[2] [Lin, 2003] and ILP for construction of both:

1. learning features for SVM classifier and
2. plain extraction rules.

They compared the success of the two approaches and discovered that the extraction model constructed by SVM (based on the induced learning features) was more successful than

[1] `http://nlp.stanford.edu/software/lex-parser.shtml`
[2] `http://webdocs.cs.ualberta.ca/~lindek/minipar.htm`

the plain extraction rules directly constructed by ILP.

3.1.3 Inductive Logic Programming and Information Extraction

There are many users of ILP in the linguistic and information extraction area. For example in [Konstantopoulos, 2003] ILP was used for shallow parsing and phonotactics. Authors of [Junker *et al.*, 1999] summarized some basic principles of using ILP for learning from text without any linguistic preprocessing. One of the most related approaches to ours can be found in [Aitken, 2002]. The authors use ILP for extraction of information about chemical compounds and other concepts related to global warming and they try to express the extracted information in terms of ontology. They use only the part of speech analysis and named entity recognition in the preprocessing step. But their inductive procedure uses also additional domain knowledge for the extraction. In [Ramakrishnan *et al.*, 2007] ILP was used to construct good features for propositional learners like SVM to do information extraction. It was discovered that this approach is a little bit more successful than a direct use of ILP but it is also more complicated. The later two approaches could be also employed in our solution.

3.1.4 Directly Comparable Systems

Thanks to the fact that evaluation of our extraction method based on ILP was performed also on a commonly used dataset, its performance can be compared with other information extraction systems that were evaluated on the dataset as well. Details about the results of the comparison will be presented in Section 8.2.6. Brief introduction of the directly comparable systems will be present in this section.

PAUM

The PAUM (Perceptron Algorithm with Uneven Margins) algorithm [Li *et al.*, 2002] is one of machine learning alternatives provided by GATE. The algorithm represents a slight modification of the classical Perceptron [Rosenblatt, 1957] used in neural networks and extended by SVM [Cortes and Vapnik, 1995]. PAUM belongs to the category of classical propositional learners working on a set of learning features manually extracted from text.

Thanks to the easy accessibility of PAUM in GATE, all our machine learning experiments could be performed with PAUM in the same time as our extraction method based on ILP. Hence these two methods were directly compared with absolutely equal conditions (the same learning and evaluation sets) and statistically significant results were recorded; see the details in Section 8.2.

SRV

Freitag [1999]

HMM

Freitag and McCallum [1999]

Elie

Finn and Kushmerick [2004]

SVM with ILP Feature Induction

Ramakrishnan *et al.* [2007]

3.1.5 Semantic Annotation

Last category of information extraction related work goes in the direction of semantics and ontologies. Bontcheva *et al.* [2004] described ontology features in GATE. They can be easily used to populate an ontology with the extracted data. We discus this topic later in Section 5.2.6.

See also references to related work connected with ontologies and information extraction in the following section.

3.2 Extraction Ontologies

Ontology-based Information Extraction (OBIE) [Wimalasuriya and Dou, 2010] or Ontology-driven Information Extraction [Yildiz and Miksch, 2007] has recently emerged as a subfield of information extraction. Furthermore, Web Information Extraction [Chang *et al.*, 2006] is a closely related discipline. Many extraction and annotation tools can be found in the above mentioned surveys ([Chang *et al.*, 2006]), many of the tools also use an ontology as the output format, but almost all of them store their extraction models in proprietary formats and the models are accessible only by the corresponding tool.

In the literature we have found only two approaches that use extraction ontologies. The former one was published by D. Embley [Embley *et al.*, 2002; Embley, 2004] and the later one – IE system Ex[3] was developped by M. Labský [Labský *et al.*, 2009]. But in both cases the extraction ontologies are dependent on the particular tool and they are kept in XML files with a proprietary structure.

By contrast authors of [Wimalasuriya and Dou, 2010] (a recent survey of OBIE systems) do not agree with allowing for extraction rules to be a part of an ontology. They use two arguments against that:

1. Extraction rules are known to contain errors (because they are never 100% accurate), and objections can be raised on their inclusion in ontologies in terms of formality and accuracy.

2. It is hard to argue that linguistic extraction rules should be considered a part of an ontology while information extractors based on other IE techniques (such as SVM, HMM, CRF, etc. classifiers used to identify instances of a class when classification is used as the IE technique) should be kept out of it: all IE techniques perform the same task with comparable effectiveness (generally successful but not 100% accurate). But the techniques advocated for the inclusion of linguistic rules in ontologies cannot accommodate such IE techniques.

   The authors then conclude that either all information extractors (that use different IE techniques) should be included in the ontologies or none should be included.

Concerning the first argument, we have to take into account that extraction ontologies are not ordinary ontologies, it should be agreed that they do not contain 100% accurate knowledge. Also the estimated accuracy of the extraction rules can be saved in the extrac-

[3] `http://eso.vse.cz/~labsky/ex/`

tion ontology and it can then help potential users to decide how much they will trust the extraction ontology.

Concerning the second argument, we agree that in the case of complex classification based models (SVM, HMM, CRF, etc.) serialization of such model to RDF does not make much sense (cf. the next section). But on the other hand we think that there are cases when shareable extraction ontologies can be useful and in the context of Linked Data[4] providing shareable descriptions of information extraction rules may be valuable. It is also possible that new standard ways how to encode such models to an ontology will appear in the future.

This short section briefly reminds main ontology definitions because they are touched and in a sense misused in this chapter. The most widely agreed definitions of an ontology emphasize the shared aspect of ontologies:

> An ontology is a formal specification of a shared conceptualization. [Borst, 1997]
>
> An ontology is a formal, explicit specification of a shared conceptualization. [Studer *et al.*, 1998]

Of course the word 'shareable' has different meaning from 'shared'. (Something that is shareable is not necessarily shared, but on the other hand something that is shared should be shareable.) We do not think that shareable extraction ontologies will contain shared knowledge about how to extract data from documents in certain domain. This is for example not true for all extraction models artificially learned from a training corpus. Here shareable simply means that the extraction rules can be shared amongst software agents and can be used separately from the original tool. This is the deviation in use of the term 'ontology' in the context of extraction ontologies in this chapter (similarly for document ontologies, see in Section 5.3.1).

3.3 Document Classification

3.3.1 General Document Classification

There are plenty of systems dealing with text mining and text classification. In [Reformat *et al.*, 2008] the authors use ontology modeling to enhance text identification. The authors of [Chong *et al.*, 2005] use preprocessed data from National Automotive Sampling System and test various soft computing methods to model severity of injuries (some hybrid methods showed best performance). Methods of Information Retrieval (IR) are numerous, with extraction mainly based on key word search and similarities. The Connection of IR and text mining techniques with web information retrieval can be found in the chapter "Opinion Mining" in the book of Bing Liu [Liu, 2007].

3.3.2 ML Classification with Monotonicity Constraint

The Fuzzy ILP Classifier can be seen as an ordinary classifier for data with the monotonicity constraint (the target class attribute has to be monotonizable – a natural ordering has to exist for the target class attribute). There are several other approaches addressing the classification problems with the monotonicity constraint.

[4]`http://linkeddata.org/`

The CART-like algorithm for decision tree construction does not guarantee a resulting monotone tree even on a monotone dataset. The algorithm can be modified [Bioch and Popova, 2009] to provide a monotone tree on the dataset by adding the corner elements of a node with an appropriate class label to the existing data whenever necessary.

An interesting approach is presented in [Kotlowski and Slowinski, 2009]: first, the dataset is “corrected” to be monotone (a minimal number of target class labels is changed to get a monotone dataset), then a learning algorithm (linear programming boosting in the cited paper) is applied.

Several other approaches to monotone classification have been presented, including instance based learning [Lievens *et al.*, 2008] and rough sets [Bioch and Popova, 2000].

4. Third Party Tools and Resources

In our solution we have exploited several tools and formalisms. These can be divided into two groups: linguistics and (inductive) logic programming. First we describe the linguistic tools and formalisms, the rest will follow.

4.1 Prague Dependency Treebank (PDT)

There exist several projects[1] closely related to the "Prague school of dependency linguistics" and the Institute of Formal and Applied Linguistics[2] in Prague, Czech Republic (ÚFAL). All the projects share common methodology and principles concerning the formal representation of natural language based on the theory of Functional Generative Description (FGD) [Sgall *et al.*, 1986]. FGD provides a formal framework for natural language representation. It is viewed as a system of layers (stratification) expressing the relations of forms and functions. Special attention is devoted to the layers of language meaning described inter alia in dependency terms. We will refer to these principles and related formalism as "PDT principles and formalisms" in the present work because the PDT projects are the most fundamental and the elaborate annotation guidelines (see bellow) were compiled within these projects. The most relevant principles for the present work will be briefly described in this section.

4.1.1 Layers of Dependency Analysis in PDT

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.

In PDT, text is split into individual sentences and dependency trees are built form them. Nodes of the trees are represented by individual words or tokens and edges represent their linguistic dependencies (simplified speaking, see bellow). Among others, two most important kinds of dependencies are distinguished: syntactical dependencies and "underlying (deep) structure" dependencies. Syntactical dependencies are often called *analytical* and the latter are called *tectogrammatical* dependencies. These two kinds of dependencies form two kinds of dependency trees: *analytical trees* and *tectogrammatical trees*. The situation is usually described using a concept of different layers (or levels) of annotation. It is illustrated by Figure 4.1. The description starts at the bottom with the surface structure of a sentence (*w-layer*); note that there is a spelling error – missing space between words 'do' and 'lesa' (into forest) in the example sentence. The lowest level of linguistic annotation (*m-layer*) represents morphology. Word forms are disambiguated and correct lemmas (dictionary form) and morphological tags are assigned at this level. The second level of annotation is called analytical (*a-layer*). At this level the syntactical dependencies of words are captured (e.g. subject, predicate, object, attribute, adverbial, etc.) The top level of annotation is called tectogrammatical (*t-layer*), sometimes also called "layer of deep syntax". At this level some tokens (e.g. tokens without lexical meaning) are leaved out or "merged" together (e.g. prepositions are merged with referred words). Also some

[1]Prague Dependency Treebank (PDT 1.0, PDT 2.0 [Hajič *et al.*, 2006]), Prague English Dependency Treebank (PEDT 1.0), Prague Czech-English Dependency Treebank (PCEDT 1.0)

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

nodes without a morphological level counterpart are inserted to the tectogrammatical tree at certain occasions (e.g. a node representing omitted subject – on Figure 4.1 labeled as "#PersPron").

More over all the layers are interconnected and additional linguistic features are assigned to tree nodes. Among others: *morphological tag* and *lemma* is assigned at the morphological level; *analytical function* (the actual kind of the particular analytical dependency, e.g. predicate, subject, object, etc.) is assigned to dependent nodes at the analytical level; *semantic parts of speech* + *grammatemes* (e.g. *definite quantificational semantic noun* (n.quant.def) + *number* and *gender*, etc.) and *tectogrammatical functor* (e.g. actor (ACT), patient (PAT), addressee (ADDR), temporal: when (TWHEN), directional: to (DIR3), etc.) is assigned at the tectogrammatical level.

Detailed information can be found:

- Homepage of the PDT 2.0 project: `http://ufal.mff.cuni.cz/pdt2.0/`
- Annotation guidelines:
  - Morphological Layer Annotation [Zeman *et al.*, 2005]
  - Analytical Layer Annotation [Hajičová *et al.*, 1999]
  - Tectogrammatical Layer Annotation [Mikulová *et al.*, 2006]
  - Tectogrammatical Layer Annotation for English (PEDT 1.0) [Cinková *et al.*, 2006]

4.1.2 Why Tectogrammatical Dependencies?

The practice has shown that representing only syntactical roles of tokens present in a sentence is not sufficient to capture the actual meaning of the sentence. Therefore the tectogrammatical level of representation was established.

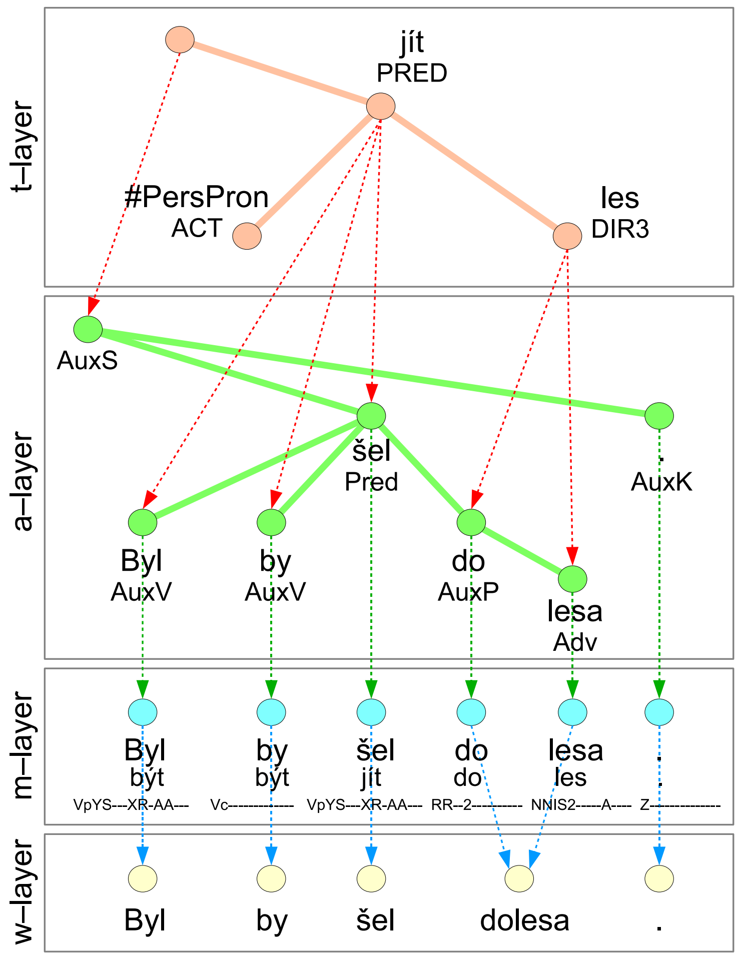
Or according to Klimeš [2006]:

> Annotation of a sentence at this [tectogrammatical] 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. This can be used in areas such as machine translation and information retrieval, however it can help other tasks as well, e.g. text synthesis.

One important thing about t-layer is that it is designed as unambiguous from the viewpoint of language meaning [Sgall *et al.*, 1986], which means that synonymous sentences should have the same single representation on t-layer. Obviously this property is very beneficial to information extraction because t-layer provides certain generalization of synonymous phrases and extraction techniques do not have to handle so many irregularities.

4.2 PDT Tools and Resources

In this section several linguistic tools and resources that are being developed at ÚFAL will be described. These tools and resources are closely connected with the PDT projects and they have been used also in the present work for various purposes.



Sample sentence (in Czech):		Byl	by	šel	dolesa.
English translation (lit.):	[He]	would	have	gone	intoforest.

Figure 4.1: Layers of linguistic annotation in PDT

4.2.1 Linguistics Analysis

Linguistic tools that were used for automated (machine) linguistic annotations of texts will be briefly described in this section. These tools are used as a processing chain and at the end of the chain they produce tectogrammatical dependency trees.

Tokenization and Segmentation

On the beginning of text analysis the input text is divided into tokens (words and punctuation); this is called *tokenization.* Sequences of tokens are then (or simultaneously) divided into sentences; this is called *segmentation.* Note that although the task seems quite simple, especially segmentation is not trivial and painful errors occur at this early stage e.g. caused by abbreviations ended by full stop in the middle of a sentence.

There are several tools available for Czech and English. The oldest (Czech) one can be found on the PDT 2.0 CD-ROM Hajič *et al.* [2006] and several choices are provided by TectoMT (see Section 4.2.3) and GATE (see Section 4.5). The best choice for Czech is probably TextSeg [Češka, 2006], which is available through TectoMT.

Morphological Analysis

Because Czech is a language with rich inflections, morphological analysis (or at least lemmatization) is an important means to success of any NLP task (starting with key word search and indexing.) In PDT morphological analysis is a necessary precondition for analytical analysis (next section).

The task of morphological analysis is for given word form in given context to select the right pair of lemma (dictionary form) and morphological tag. In PDT the tag includes part of speech (POS) and other linguistic categories like gender, number, grammatical case, tense, etc., see morphological annotation guidelines for details.

For Czech two main tools are available: Feature-based tagger[3] by Hajič [2000] and perceptron-based tagger Morče[4] by Votrubec [2006]. Morče is several years newer and it achieved few points better accuracy on PDT 2.0 (94.04% vs. 95.12% see in [Spoustová *et al.*, 2007]). Both tools are available through TectoMT, Morče also for English.

Analytical Analysis

The task of Analytical analysis is to build up a syntactical dependency tree (analytical tree) from a morphologically analyzed sentence and to assign right kind of dependency (analytical function) to every edge of the tree.

There were experiments with almost all well known parsers on the Czech PDT data[5]. But two of them have proved in practice: Czech adaptation of Collins' parser [Collins *et al.*, 1999] and Czech adaptation of McDonald's MST parser [Novák and Žabokrtsky, 2007]. Again the second tool is few years newer and few points better in accuracy (80.9% vs. 84.7% on PDT 2.0[5]). The Collins' parser can be found on the PDT 2.0 CD-ROM and the MST parser is available through TectoMT.

The analytical analysis of English is not very well established because PEDT 1.0[6] contains only tectogrammatical level of annotation and there is no other English treebank

[3]`http://ufal.mff.cuni.cz/tools.html/fbtag.html`
[4]`http://ufal.mff.cuni.cz/morce/`
[5]See details at `http://ufal.mff.cuni.cz/czech-parsing/`
[6]`http://ufal.mff.cuni.cz/pedt/`

for the analytical level. The English analytical analysis is regarded rather as a part of English tectogrammatical analysis; see details in [Klimeš, 2007].

Tectogrammatical Analysis

During tectogrammatical analysis analytical trees are transformed to tectogrammatical ones. Merging, omitting and inserting of tree nodes takes place as well as assignment of all the complex linguistic information of the tectogrammatical level (semantic parts of speech, grammatemes, tectogrammatical functors, etc.)

The tectogrammatical analysis can by performed by a variety of TectoMT *blocks* (depending on the amount of requested linguistic information, for example only tectogrammatical functors can be assigned.) Better results can be obtained through transformation-based tools developed by Václav Klimeš: [Klimeš, 2006] for Czech and [Klimeš, 2007] for English; they are also available thorough TectoMT.

4.2.2 Tree Editor TrEd, Btred

TrEd is the ÚFAL key tool for work with dependency based linguistic annotations. It provides a comfortable GUI for navigation, viewing and editing of linguistic trees at different levels of annotation, for different languages and different treebank schemas. TrEd is implemented in Perl and it is available for a variety of platforms (Windows, Unix, Linux, Mac OS X).

Homepage of the project: `http://ufal.mff.cuni.cz/~pajas/tred/`

Btred

TrEd can be also controlled using a powerful set of Perl macros and there is also a non-interactive version of TrEd called Btred, which allows batch evaluation of user macros on an arbitrary set of annotated documents.

Btred/ntred tutorial: `http://ufal.mff.cuni.cz/~pajas/tred/bn-tutorial.html`

PML Tree Query

PML Tree Query (PML-TQ) [Pajas and Štěpánek, 2009] is a TrEd based module for searching through a treebank using a complex tree based query language.

Homepage of the project: `http://ufal.mff.cuni.cz/~pajas/pmltq/`

4.2.3 TectoMT

TectoMT [Žabokrtský *et al.*, 2008] is a Czech project that contains many linguistic tools for different languages including Czech and English; all the tools are based on the dependency based linguistic theory and formalism of PDT. It is implemented in Perl; highly exploiting TrEd libraries. The recommended platform is Linux. It is primarily aimed at machine translation but it can also facilitate development of software solutions of other NLP tasks.

We have used a majority of applicable tools from TectoMT (e.g. tokeniser, sentence splitter, morphological, analytical and tectogrammatical analyzers for Czech and English). We have also developed TectoMT wrapper for GATE, which makes it possible to use TectoMT tools inside GATE, see details in Section 6.2.1.

Similarly to GATE, TectoMT supports building of application pipelines (*scenarios*) composed of so called *blocks* – processing units responsible for single independent task (like tokenization, parsing, etc.)

Homepage of the project: `http://ufal.mff.cuni.cz/tectomt/`

4.2.4 Netgraph

Netgraph [Mírovský, 2006] is a linguistic tool used for searching through a syntactically annotated corpus of a natural language (corpus of linguistic dependency trees). Besides the searching capabilities it also provides a GUI for viewing of dependency trees. Both of these features were exploited in the present work.

Netgraph implementation is client-server based and a special query language is used for searching. The query language allows putting restrictions on the shape of a tree and on values of attributes of an arbitrary tree node. Besides that nodes of a query can be marked as optional (not necessarily present in a matching tree) and names can be assigned to query nodes. Naming of query nodes then allows putting restrictions based on referenced nodes (Give me all trees where there is a node with two children and both children have the same lemma.) See also Section 6.1.2, it provides additional information about the query language including example Netgraph queries.

Currently Netgraph is replaced by PML Tree Query (see Section 4.2.2) and the Netgraph development is "discontinued". We use Netgraph in the present work because PML Tree Query is quite young and it was not available when our development started and because Netgraph is written in Java, the language used for GATE as well as for our implementation. The usage of Java also allowed integration of Netgraph inside GATE as a handy viewer of dependency trees, see details in Section 6.2.3.

Homepage of the project: `http://quest.ms.mff.cuni.cz/netgraph/`

4.3 Czech WordNet

Figure 5.3, presented and described in later sections, shows, that it would be useful to gather words with similar meanings in our extraction rules. For example, the rule (Figure 5.3) contains long disjunctions of similar words (nodes with numbers 1 and 4). These disjunctions could be replaced with some kind of expression telling that we are looking for any word from some semantic category (e.g. human beings). For this purpose we wanted to use the Czech WordNet [Pala and Smrž, 2004].

After we have explored the records of the Czech WordNet (CzWN) related to the domain of our interest (car accidents, etc.) we have decided not to involve CzWN in the extraction process. The reason is that the coverage of the vocabulary of our domain is rather poor and the semantic connections of words are sometimes unfortunately missing (e.g. car brands and models). A sample from the Czech WordNet can found in the appendix, Section A.1. But we can supply the missing information to CzWN or we can build up a new domain-specific word-net based on the ground of CzWN.

Availability: `http://catalog.elra.info/product_info.php?products_id=1089`
or `http://hdl.handle.net/11858/00-097C-0000-0001-4880-3`

4.4 Other dependency representations

4.4.1 CoNLL'X dependencies

This dependency tree format was used in the CoNLL'X Shared Tasks on multi-lingual dependency parsing [Buchholz and Marsi, 2006]. It has been adopted by most native dependency parsers and was originally obtained from Penn Treebank (PTB) trees using constituent-to-dependency conversion[7] [Johansson and Nugues, 2007].

4.4.2 Stanford dependencies

This format was proposed by de Marneffe *et al.* [2006] for semantics-sensitive applications using dependency representations, and can be obtained using the Stanford tools[8] from PTB trees.

4.5 GATE

GATE [Cunningham *et al.*, 2002] is probably the most widely used tool for text processing. In our solution the capabilities of document and annotation management, utility resources for annotation processing, JAPE grammar rules [Cunningham *et al.*, 2000], machine learning facilities and performance evaluation tools are the most helpful features of GATE that we have used.

Homepage of the project: `http://gate.ac.uk/`

4.5.1 GATE Annotations

Contrary to PDT, GATE annotations[9] are rather simple and minimalistic. They are designed as labeled segments of text. A single annotation is described by its label (*annotation type*) and starting and ending character offset. Each annotation has a unique identifier (integer ID) and an arbitrary set of *features*[10] (name-value pairs) can be assigned to it.

For example in a sentence "Hamish Cunningham leads the GATE team.", "Hamish Cunningham" can be annotated with a GATE annotation starting at character 0, ending at character 16, annotation type: "Person", with three features: "firstName=Hamish; surname= Cunningham; gender=male"; because it is the only annotation, the ID would most probably be 0.

Although the GATE annotation approach seems quite simple, very complex structures can be encoded this way (for example an annotation feature can contain a reference to another annotation using its ID), but such usage of GATE annotations is always tricky to some degree and it is always necessary to establish a convention about that. In Section 6.2.2 encoding of PDT dependency annotations in GATE will be presented.

4.5.2 Gazetteer Lists

Gazetteers provide a list of known entities for a particular category, such as all counties of the world or all human diseases and are often used in information extraction. See also the

[7]`http://nlp.cs.lth.se/software/treebank_converter/`
[8]`http://nlp.stanford.edu/software/lex-parser.shtml`
[9]`http://gate.ac.uk/userguide/sec:corpora:dags`
[10]`http://gate.ac.uk/userguide/sec:corpora:features`

GATE documentation chapter about gazetteers[11].

For flexitive languages like Czech, gazetteers can not be used directly because the term lookup without lemmatization would result in poor performance. Using GATE Flexible Gazetteer[12] this problem can be elegantly solved and terms from gazetteer list are then matched against tokens lemmas instead of their original forms. This also implies that the gazetteer's terms have to be in the form of lemmas and lemmatization has to be preformed on the analyzed documents before the gazetteer application.

4.5.3 Machine Learning in GATE

Figure 4.2 presents a general machine learning and annotation process in GATE and the same schema apply also to majority of information extraction approaches based on machine learning. An adaptation of this schema will be presented in Section 5.2, where our method based on deep language parsing and Inductive Logic Programming will be presented.

The schema presets two sub-processes or phases:

- Learning phase represented by arrows with numbers (1-6).
- Application phase represented by arrows with Latin characters (a-e).

Arrows with Greek characters (α-γ) represent configuration of individual components.

[11]http://gate.ac.uk/userguide/chap:gazetteers
[12]http://gate.ac.uk/userguide/sec:gazetteers:flexgazetteer

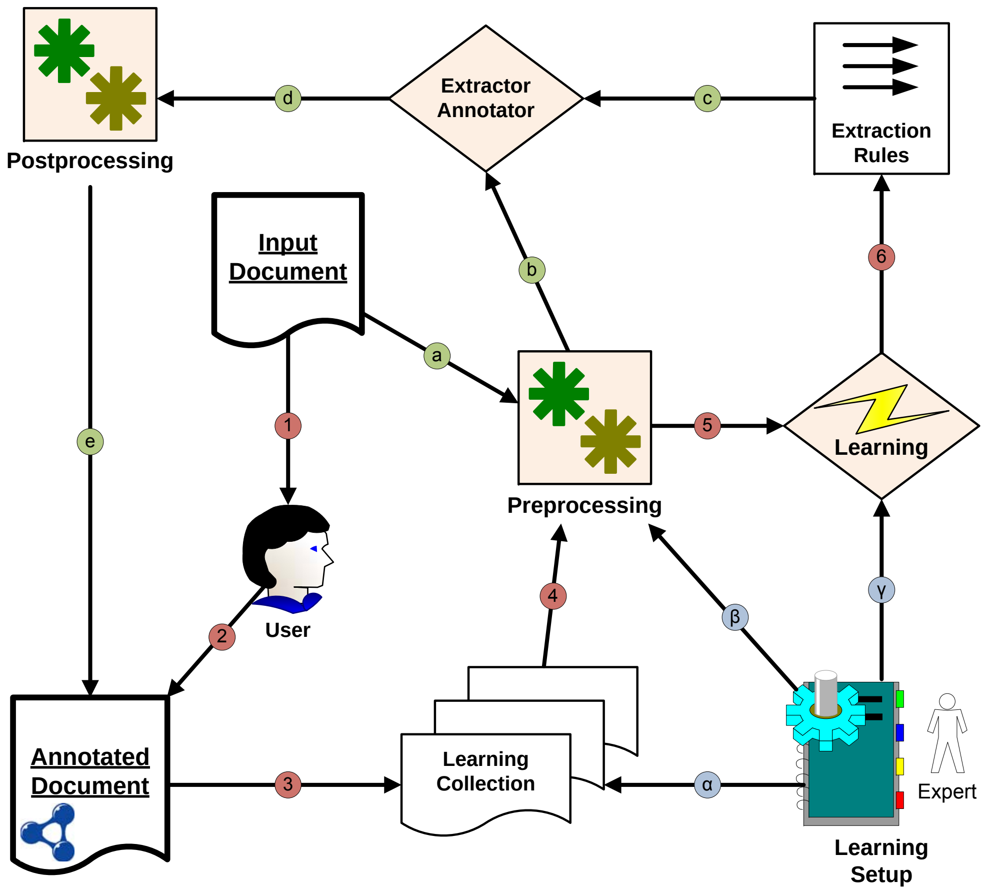


Figure 4.2: Machine learning and annotation process – general overview

During the **learning phase**, input documents are presented to users or human annotators (1) and they perform manual annotation on them (2). Annotated documents are collected and they form a learning collection (2). The size of learning collection and selection of documents is configured in learning setup by an expert user or administrator (α). Before a learning algorithm can be applied on the learning collection (5), preprocessing of learning documents takes place (4). The learning phase ends up with learned extraction rules or other kind of machine learning model (6). Document preprocessing (β) as well as learning algorithm (γ) have to be properly configured by the administrator because different kinds of preprocessing and different setting of learning parameters are suitable to different extraction tasks.

During the **application phase**, an input document goes through the same preprocessing (a) as learning documents. Extraction or annotation procedure applies the previously learned extraction rules (c) on the preprocessed document (b). The result document is usually not yet in the final form and postprocessing has to be performed on it (d). The final annotated document (e) contains the same form of annotations as those created by human annotator.

There is one configuration arrow missing in the schema. This arrow should go to the postprocessing procedure because different kinds of postprocessing suit to different extraction tasks and the actual choice of postprocessing steps is highly dependent on the preprocessing steps performed before. For example "root preprocessing" has to be followed by "subtree postprocessing"; see details in Section 5.2.4.

4.6 Named Entity Recognition

In our solution, various tools performing Named Entity Recognition[13] were used. Some of them are available thorough TectoMT (e.g. Stanford Named Entity Recognizer [Finkel *et al.*, 2005]), some of them through GATE, mainly ANNIE[14] (a Nearly-New Information Extraction System).

For Czech, we used a simple solution based on gazetteer list of all Czech cities, city parts and streets provided by the Ministry of the interior of the Czech Republic[15], although there are mature Czech named entity recognizers available, see for example work of Kravalová and Žabokrtský [2009].

4.7 Inductive Logic Programming

Inductive Logic Programming (ILP) [Muggleton, 1991] is a machine learning technique based on logic programming. Given an encoding of the known background knowledge (in our case linguistic structure of all sentences) and a set of examples represented as a logical database of facts (in our case tokens annotated with the target annotation type are positive examples and the remaining tokens negative ones), an ILP system will derive a hypothesized logic program (in our case extraction rules) which entails all the positive and none of the negative examples.

Formal definitions of ILP tasks are presented in following sections. They will be extended and used for implementation of Fuzzy ILP Classifier, see details in Section 5.4.

[13]See also Section 2.1.4 providing the definition of the problem.
[14]`http://gate.ac.uk/userguide/chap:annie`
[15]`http://www.mvcr.cz/adresa`

4.7.1 Classical ILP

In our presentation of ILP we follow Dzeroski and Lavrac [2001] and Muggleton and de Raedt [1994].

Definition 1 (Classical ILP task)**.** A set of examples $E = P \cup N$, where P contains positive and N negative examples, and background knowledge denoted by B are given. The task of ILP is to find a hypothesis H such that

$$(\forall e \in P)(B \cup H \models e)$$

and

$$(\forall e \in N)(B \cup H \not\models e).$$

Typically, E consists of ground instances of the target predicate, in our case tree nodes relevant for the praticular extraction task or accident seriousness (see examples in Figure 6.17). B typically consists of several predicates (relational tables), which describe properties of an object, in our case properties of an accident (see examples in Figure 6.18) or the structure and properties of linguistic trees. The background knowledge can also contain some rules. A hypothesis H typically consists of logic programming rules (see examples in Figure 6.20 and Figure 6.20). H added to B entails all positive examples and no negative examples. The main advantage of ILP is its multirelational character, namely, B can reside in several relational tables.

4.7.2 Fuzzy ILP

In our presentation of fuzzy ILP we follow the paper of Horváth and Vojtáš [2007] about fuzzy inductive logic programming. We use the approach of the fuzzy logic in a narrow sense, developed by Pavelka [1979] and Hájek [1998]. Formulas are of the form φ, x (φ is syntactically the same as in the classical case), they are graded by a truth value $x \in [0, 1]$. A structure $\mathcal{M}$ consists of a domain M and relations are interpreted as fuzzy (we do not consider function symbols here). The evaluation $\|\varphi\|_{\mathcal{M}}$ of a formula φ uses truth functions of many-valued connectives (our logic is extensional and/or truth functional). The satisfaction $\models_f$ is defined by

$$\mathcal{M} \models_f (\varphi, x) \; iff \;\; \|\varphi\|_{\mathcal{M}} \geq x.$$

Definition 2 (Fuzzy ILP task)**.** A fuzzy set of examples $\mathcal{E} : E \longrightarrow [0, 1]$ and a fuzzy background knowledge $\mathcal{B} : B \longrightarrow [0, 1]$ are given. The task of fuzzy ILP is to find a fuzzy hypothesis $\mathcal{H} : H \longrightarrow [0, 1]$ such that

$$(\forall e_1, e_2 \in E)(\forall \mathcal{M})(\mathcal{M} \models_f \mathcal{B} \cup \mathcal{H})$$

we have

$$\mathcal{E}(e_1) > \mathcal{E}(e_2) \Rightarrow \|e_1\|_{\mathcal{M}} \geq \|e_2\|_{\mathcal{M}}.$$

That is, it cannot happen that

$$\mathcal{E}(e_1) > \mathcal{E}(e_2) \wedge \|e_1\|_{\mathcal{M}} < \|e_2\|_{\mathcal{M}},$$

or rephrased: if $\mathcal{E}$ is rating e_1 higher than e_2, then it cannot happen that e_1 is rated worse than e_2 in a model of $\mathcal{B} \cup \mathcal{H}$.

Typically, $\mathcal{E}$ consists of ground instances of the target predicate, which are classified in truth degrees, in case of accident classification, degrees of seriousness of an accident. $\mathcal{B}$ typically consists of several fuzzy predicates (fuzzy relational tables), which describe properties of an object, in our case fuzzy properties of an accident – a degree of injury, a degree of damage, etc. A hypothesis $\mathcal{H}$ typically consists of a fuzzy logic program, which, when added to $\mathcal{B}$, prevents misclassification (what is better cannot be declared to be worse, nevertheless it can be declared as having the same degree – for more detailed discussion on this definition of fuzzy ILP we refer to the paper [Horváth and Vojtáš, 2007]).

4.7.3 Aleph – the ILP Tool

The ILP tool, which we have used in both:

- the information extraction method based on deep language parsing and ILP machine learning (Section 5.2) and
- the Fyzzy ILP Classifier (Section 5.4),

is called "A Learning Engine for Proposing Hypotheses" (Aleph v5).We consider this tool very practical. It uses quite effective method of inverse entailment [Muggleton, 1995] and keeps all handy features of a Prolog system (we have used YAP Prolog[16]) in its background.

Homepage of the project: `http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/`

4.8 Weka

Weka [Hall *et al.*, 2009] is a well known data-mining software. In our work, we have used the Weka implementation of various machine learning algorithms (Multilayer Perceptron, SVM, J48 decision trees, etc.) for the classical classification problem and we compared the performance of the Fuzzy ILP Classifier with these algorithms; see details in Section 8.4.

The **Weka Experimenter** is another very useful component of Weka. This component supports running of multiple experiments in batch mode and then an easy comparison of the performance of various algorithms and investigation of the statistical significance. All experiments (see Sections 8.2 and 8.4) presented in this thesis, where the statistical significance is provided, were performed using the Weka Experimenter

Homepage of the project: `http://www.cs.waikato.ac.nz/ml/weka/`

[16]`http://www.dcc.fc.up.pt/~vsc/Yap/`

5. Models and Methods

In this chapter, the four methods corresponding to the four topics of this thesis will be introduced and described. The chapter is structured to four main sections according to the four topics.

5.1 Manual Design of Extraction Rules

In this section, the extraction method based on manually created linguistic rules will be described. First of all a data flow schema of the extraction process will be presented. Then a description of several stages of evolution of the method will demonstrate how this method came to its existence and which decisions stood behind the development and the final implementation, which will be described in the next chapter.

The approach is based on linguistic preprocessing, oriented to text structured to individual sentences.

5.1.1 Data Flow

The method was designed as a method for extraction of information from web resources. Thus the extraction process starts on the Web. On the other hand the method was intended to serve the evolution of the Semantic Web, so the final goal of the extraction process is the extracted information stored in the form of a semantic web ontology. A schema in Figure 5.1 splits the process into four steps (phases) among five media types. The schema does not cover the extraction rules design phase; it is assumed that extraction rules were already designed by a human designer; Section 5.1.4 provides details about that. A description of individual steps follows.

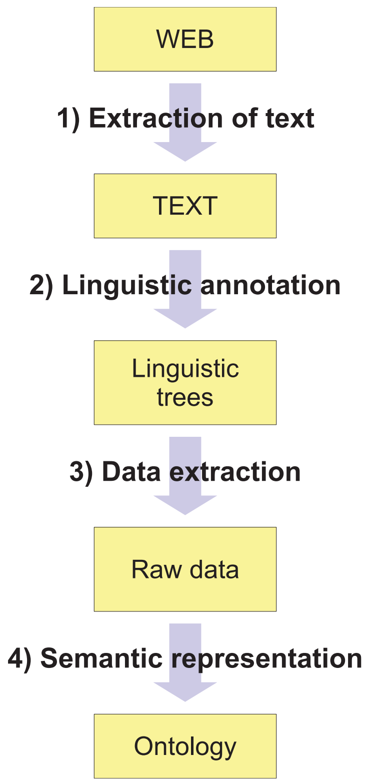
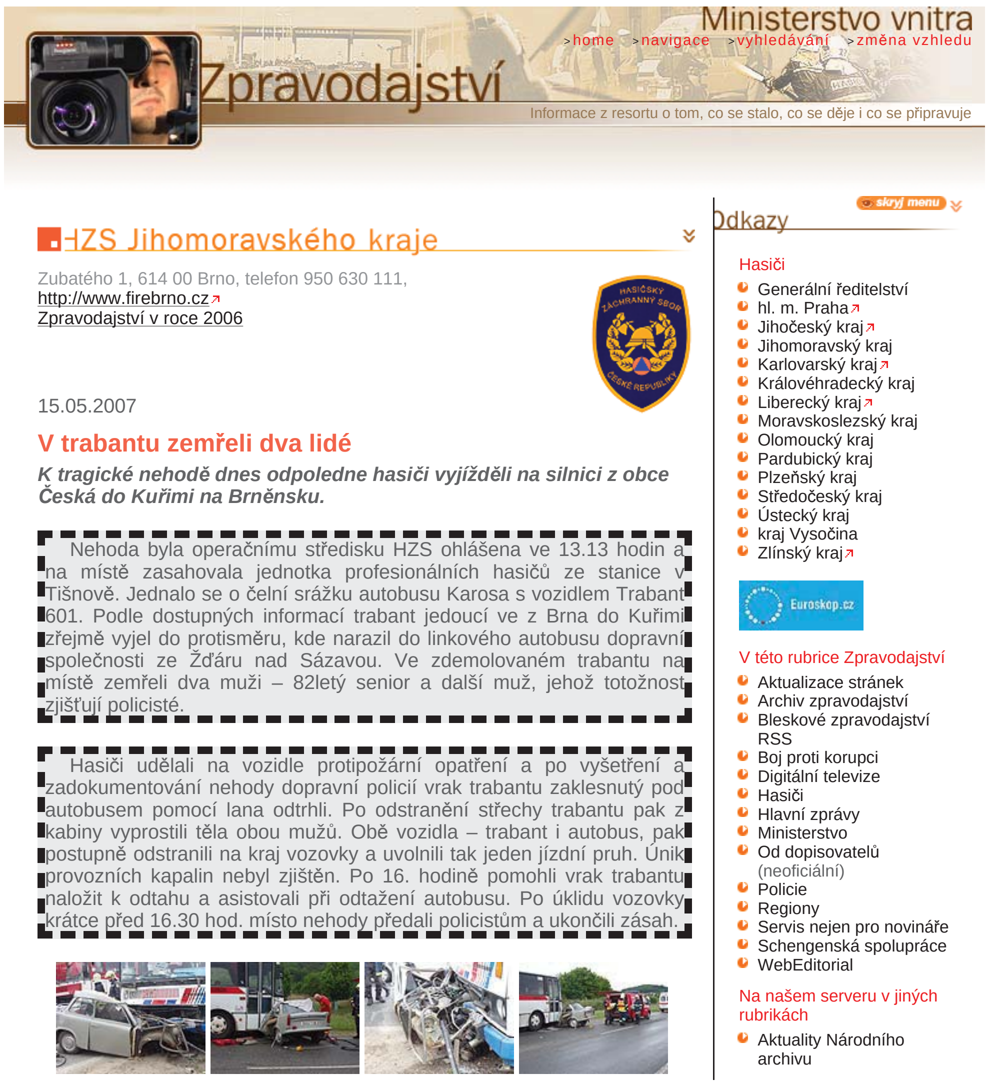


Figure 5.1: Schema of the extraction process.

1. *Extraction of text*
   First of all, target web pages have to be identified, downloaded and text has to be extracted form them. This issue is not studied in the present work. A RSS feed of the fire department web-site was used to identify and download relevant pages and the desired text (see highlighted area in the Figure 5.2) was extracted by means of a regular expression. The text is an input of the second phase.

2. *Linguistic annotation*
   In this phase the extracted text is processed by several linguistic tools. The tools analyze the text and produce corresponding set of linguistic dependency trees. There is a rich choice of linguistic tools available (see Section 4.2.1), but only PDT based tools were used in illustration examples and linguistic trees are always of the form of Tectogrammatical trees, but note that the method is general and it is not limited to the PDT presentation of linguistic dependency trees.

3. *Data extraction*
   The structure of linguistic dependency trees is used for the extraction of relevant



Ministerstvo vnitra
> home > navigace > vyhledávání > změna vzhledu
Zpravodajství
Informace z resortu o tom, co se stalo, co se děje i co se připravuje
skrýt menu

HZS Jihomoravského kraje

Zubatého 1, 614 00 Brno, telefon 950 630 111,
http://www.firebrno.cz
Zpravodajství v roce 2006

15.05.2007

V trabantu zemřeli dva lidé

K tragické nehodě dnes odpoledne hasiči vyjížděli na silnici z obce Česká do Kuřimi na Brněnsku.

Nehoda byla operačnímu středisku HZS ohlášena ve 13.13 hodin a na místě zasahovala jednotka profesionálních hasičů ze stanice v Tišnově. Jednalo se o čelní srážku autobusu Karosa s vozidlem Trabant 601. Podle dostupných informací trabant jedoucí ve z Brna do Kuřimi zřejmě vyjel do protisměru, kde narazil do linkového autobusu dopravní společnosti ze Žďáru nad Sázavou. Ve zdemolovaném trabantu na místě zemřeli dva muži – 82letý senior a další muž, jehož totožnost zjišťují policisté.

Hasiči udělali na vozidle protipožární opatření a po vyšetření a zadokumentování nehody dopravní policií vrak trabantu zaklesnutý pod autobusem pomocí lana odtrhli. Po odstranění střechy trabantu pak z kabiny vyprostili těla obou mužů. Obě vozidla – trabant i autobus, pak postupně odstranili na kraj vozovky a uvolnili tak jeden jízdní pruh. Únik provozních kapalin nebyl zjištěn. Po 16. hodině pomohli vrak trabantu naložit k odtahu a asistovali při odtažení autobusu. Po úklidu vozovky krátce před 16.30 hod. místo nehody předali policistům a ukončili zásah.

Odkazy

Hasiči
- Generální ředitelství
- hl. m. Praha
- Jihočeský kraj
- Jihomoravský kraj
- Karlovarský kraj
- Královéhradecký kraj
- Liberecký kraj
- Moravskoslezský kraj
- Olomoucký kraj
- Pardubický kraj
- Plzeňský kraj
- Středočeský kraj
- Ústecký kraj
- kraj Vysočina
- Zlínský kraj

Euroskop.cz

V této rubrice Zpravodajství
- Aktualizace stránek
- Archiv zpravodajství
- Bleskové zpravodajství RSS
- Boj proti korupci
- Digitální televize
- Hasiči
- Hlavní zprávy
- Ministerstvo
- Od dopisovatelů (neoficiální)
- Policie
- Regiony
- Servis nejen pro novináře
- Schengenská spolupráce
- WebEditorial

Na našem serveru v jiných rubrikách
- Aktuality Národního archivu

Figure 5.2: Example web page with an accident report.

structured data form the text. The extraction method used in this phase is the main topic of this section (Section 5.1) and details about the method are discuses in following subsections.

4. *Semantic representation*
   Although the output of the previous phase is already of a structured form, it is not necessarily of the form of a semantic web ontology. The output has to be converted to some ontology format (RDF, OWL) and appropriate schema for given domain and type of extracted information.

   This last step of the extraction process represents a logical distinction between two functionally different tasks of the extraction method. The first task represented by the previous (Data extraction) phase is responsible for choosing of "what" should be extracted, while the second task (Semantic representation) should determine what to do with the extracted data or how to formulate the pieces of information discovered by the Data extraction phase. The border between these two tasks is rather vague and they could be merged together, but we think that the distinction between them can help to understand the problem better.

   In the present work only a design of this phase is provided (with a small exception

– using shareable extraction ontologies, see Section 5.3) because: (1) this task seems to be strongly dependent on manual work of human designers and (2) its potential for meaningful scientific investigation seems to be rather small. Details about this step are further discussed in Section 5.1.5.

5.1.2 Evolution of the Method

Our first attempt to extract some structured data from linguistically annotated text was done in a standard procedural programming environment (more precisely in Perl, Btred). After an initial phase of development first extraction rule was created as a single executable procedure. This procedure will be described in the next chapter (Section 6.1.1) and listed in Figure 6.1. There are many drawbacks of the procedural rule design: such extraction rules are difficult to read, tedious to create, error prone, graphical or assisted design is impossible. On the other hand, this approach has the advantage of the programming language proximity. When a designer designs a procedural extraction rule he or she actually codes it in a procedural programming language and it is easy to add some additional functionality that will be executed and evaluated along with the extraction rule. Thus the designer has the full power of the programming language in hand and he or she can use it inside of the extraction rule. This possibility will be discussed later in the context of semantic interpretation of extracted data.

Dissatisfaction from tedious and time consuming design of procedural extraction rules led us to the idea of a special rule language. We were looking for a language that would allow expressing tree patterns and consequent extraction actions of extraction rules. It turned out that the Netgraph query language is very suitable for the first purpose – expressing tree patterns. An extension of the Netgraph query language to a language for extraction rules was quite simple then. See the details in the next section.

Last two steps in the evolution of the extraction method were (1) creation of machine learning procedure that is capable to learn extraction rules from manually annotated corpus (see in Section 5.2) and (2) possibility to export extraction rules to a shareable extraction ontology so the extraction rules can be evaluated on a document by an ordinary semantic web reasoner outside of the original extraction tool (see in Section 5.3).

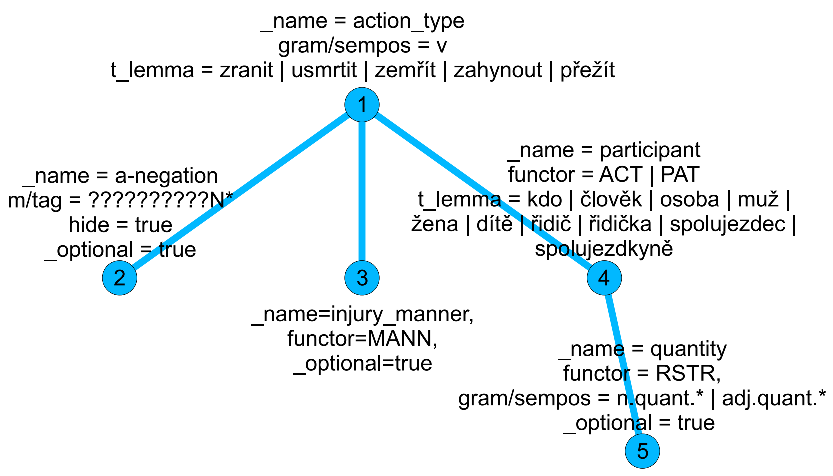
5.1.3 Netgraph Based Extraction Rules

Netgraph based extraction rules are declarative, they do not specify a sequential procedure “what to do with a linguistic tree”, they are rather based on conditions and selections similar to SQL.

Netgraph is a linguistic tool used for searching through a syntactically annotated corpus (see Section 4.2.4 for details). Netgraph queries are written in a special query language with a graphical representation. The graphical representation of a query is much better readable than its linear textual representation and we will use the graphical representation only. Figure 5.3 shows an example Netgraph query. It specifies necessary tree structure that has to be present in a matching tree and attribute restrictions that have to hold true for corresponding nodes (the restrictions are printed beside the nodes).

We adopted Netgraph queries and extended them to extraction rules that can be written in following pseudo SQL SELECT form:

```
SELECT node1_name.attr1_name, node2_name.attr2_name, ...
FROM netgraph_query
```



Transcript:

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

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

řidič driver	řidička woman driver	spolujezdec passenger	spolujezdkyně woman passenger

Figure 5.3: A manually created extraction rule investigating numbers of injuries and fatalities.

where *netgraph_query* stands for an arbitrary Netgraph query, *node1_name*, *node2_name*, etc. stand for individual names of nodes defined in *netgraph_query* and *attr1_name*, *attr2_name*, etc. stand for names of linguistic attributes whose values should be picked out from the corresponding matching tree nodes.

The extraction works as follows: the Netgraph query is evaluated by searching through a corpus of linguistic trees. Matching trees are returned and the desired information defined by the SELECT part of the extraction rule is taken from particular tree nodes and printed to the output.

Let us explain the extraction procedure in detail, using the example of extraction rule from the Figure 5.3, which is looking for information about killed and injured people during a (usually car) accident. This rule consists of five nodes. Each node of the rule will match with the corresponding node in each matching tree. So we can investigate the relevant information by reading values of linguistic attributes of matching nodes. We can find out the number (node number 5) and kind (4) of people, which were or were not (2) killed or injured (1) by an accident that is presented in the given sentence. And we can also identify the manner of injury (light or heavy) in the node number 3.

Implementation details and some additional examples of these extraction rules will be presented in the next chapter (Section 6.1).

5.1.4 Methodology for Rule Designers

The process of manual design of extraction rules is heavily dependent on skills and experience of a human designer and fulfillment of the process is quite creative task. In this section we try to pick it up as precisely as possible because we assume that a formal description of this process can help in two ways. First – a new designer can use it as a cook book and progress more quickly. Second – it can help with development of tools for assisted rule

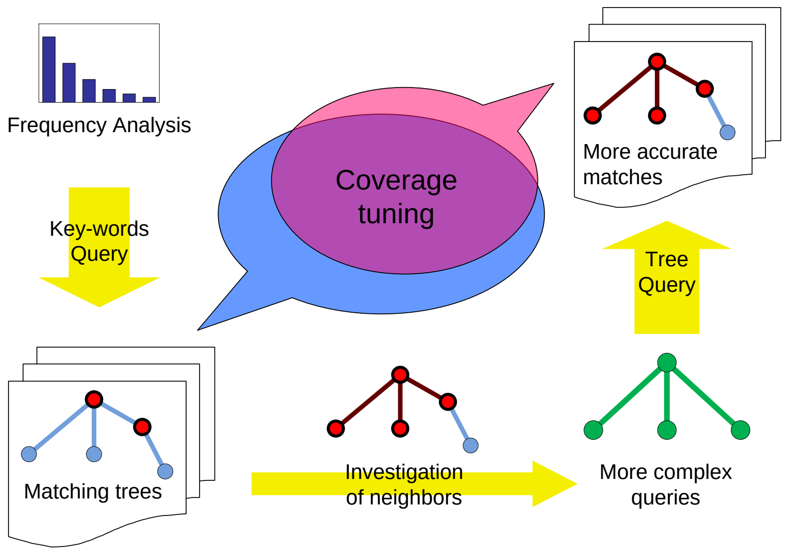


Figure 5.4: Gradual refinement of an extraction rule.

design. We will concentrate on the Netgraph based extraction rules because we think they are more useful.

The process consists of two parts: construction of a Netgraph query and semantic interpretation of the query. The semantic interpretation part will be discussed in the next section.

One obvious preposition of the procedure is that we have a collection of training texts. The procedure is demonstrated in Figure 5.4 and it starts with frequency analysis of words (their lemmas) occurring in the texts. Especially frequency analysis of verbs is very useful — meaning of a clause is usually strongly dependent on the meaning of the corresponding verb.

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

Next step of the procedure consists in **investigating trees** that are covered by key-words. The designer is examining **matching trees** — looking for positions of key-words and their **neighboring** nodes.

After that the designer can formulate an initial **Netgraph 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 refine the query and **tune the query coverage**.

There are two goals of the query tuning. The first goal is maximization of the relevance of the query. An ideal result is a query that covers all sentences expressing given type of information and no other. The second goal is to involve all important tree-nodes to the query. The 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 5.3 — each node keeps different kind of information.

5.1.5 Semantic Interpretation of Extracted Data

After the designer has successfully formulated the Netgraph query he or she has 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 output data.

We did not talk about the extraction output so far. It will be described in the next chapter. For the current description, it is sufficient to say that both methods (the procedural one and the declarative one) have, although structured but still, proprietary XML extraction output. This corresponds to the penultimate stage (raw data) of our data flow schema presented in Section 5.1.1. In this section, we will describe details about the last step of the data flow schema – semantic representation of extracted data.

In Section 6.1.3, the difference between the output of the procedural extraction rule (Section 6.1.1, Figure 6.2) and the Netgraph based extraction rule (Section 6.1.2, Figure 6.5) can be observed. The procedural one is closer to the semantics of the extracted data while the Netgraph based one is more general, rather based on the semantics of the matching tree and extraction rule. The difference is connected with the difference of the design processes of these extraction rules. While Netgraph based rules are designed in a comfortable way using a graphical tool, the procedural rules have to be coded manually in the programming language of Perl. A Perl programmer has great freedom in the design of a procedural rule and he or she can adapt the rule such that it precisely respects the semantics of extracted data. A designer of a Netgraph based rule has the only freedom in the construction of a Netgraph query and in selection particular query nodes and linguistic attributes that will be selected for the output.

As stated in Section 5.1.1 about the data flow, the goal of our extraction process is in the form of a semantic web ontology. This is not difficult in the case of procedural rules. Once the schema (or vocabulary) of the target ontology is selected, the extraction rules can be simply designed to produce the output of that form (Note that semantic web ontologies can be captured in a specific XML format.)

In the case of Netgraph based queries, the situation is more complex and different solutions can be discovered. All the solutions have one thing in common: additional manual work is necessary. The problem is basically to create a mapping of the data in one format (results of Netgraph based rules) to another format (target ontology). It can be done by a variety of technical means (coded in an arbitrary programming language, XSLT, or using a graphical mapping tool like Altova MapForce[1] or Stylus Studio[2]).

[1]`http://www.altova.com/mapforce/xml-mapping.html`
[2]`http://www.stylusstudio.com/xsd_to_xsd.html`

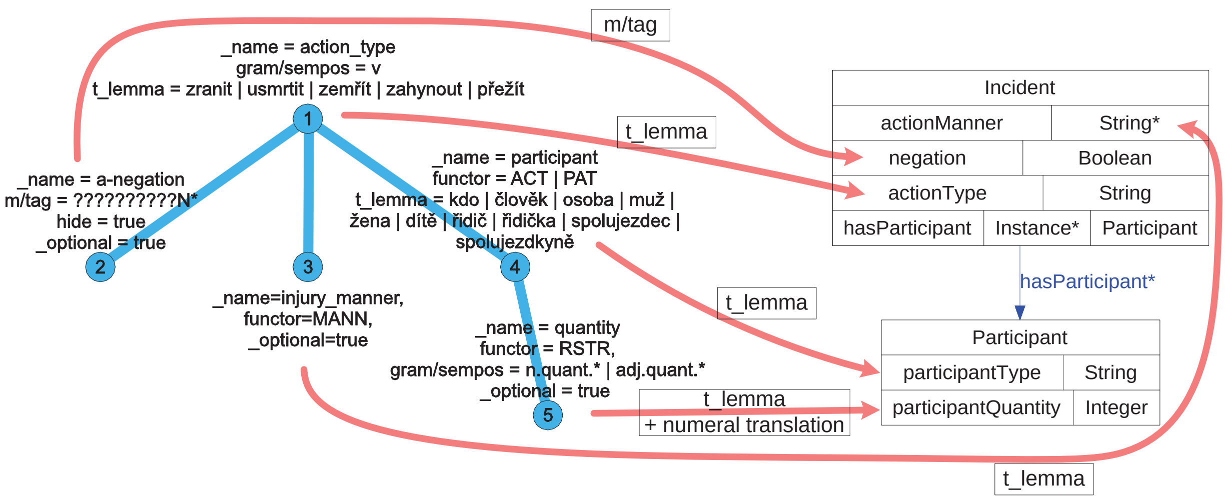


Figure 5.5: Semantic interpretation of the extraction rule.

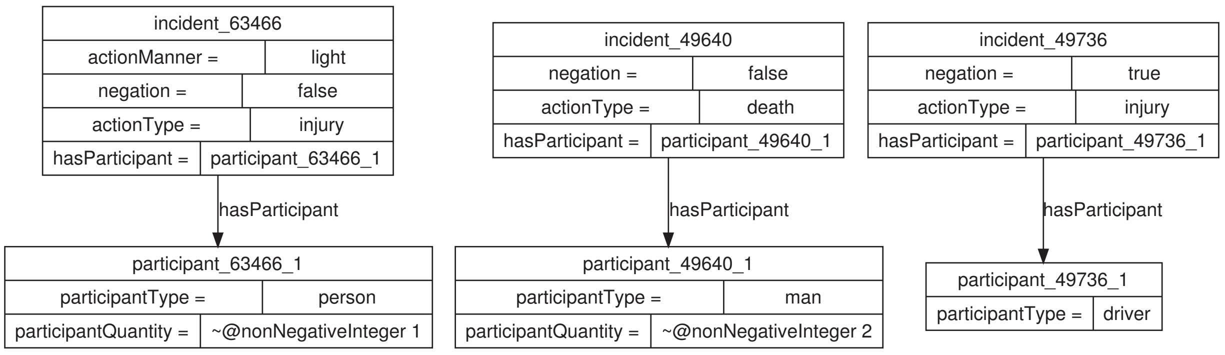


Figure 5.6: Extracted instances of the target ontology.

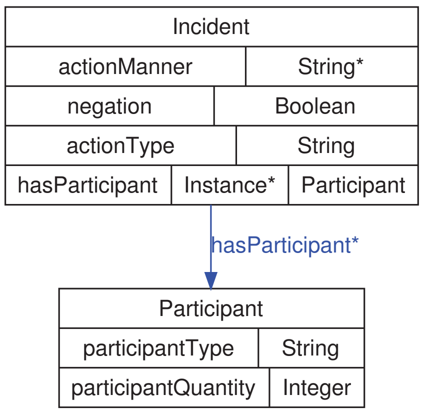


Figure 5.7: Schema of the target ontology.

Similar but in a sense different solution is to ground the mapping directly in extraction rules. Instead of creating mapping of the extraction output, extraction rules will contain also the information about the form of the extraction output. Selection of particular query nodes and linguistic attributes for the output will be extended by the specification of how the attributes will be rendered on the output. A graphical representation of such extraction rule can look like in Figure 5.5. It shows the connection between a Netgraph query on the left and an ontology instance on the right. Every node of the query can be translated to the ontology and the translation can be configured.

The configurable translations are the most interesting part of these extraction rules. The linguistic information on one side has to be converted to the ontological information on the other side. In Figure 5.5, following translation types were used: a translation of numerals to numbers, lexical translation from a source language (Czech), and detection of negation present in a

```
SELECT ?action ?participant ?participant_type ?quantity
WHERE {
        {
                ?action rdf:type :Incident;
                        :actionType "death";
                        :negation false.
        } UNION {
                ?action rdf:type :Incident;
                        :actionType "survival";
                        :negation true.
        }
        ?action :hasParticipant ?participant.
        ?participant :participantType ?participant_type.
        OPTIONAL {
                ?participant :participantQuantity ?quantity.
        }
}
```

Figure 5.8: *SPARQL* query that summarizes fatalities of particular incidents.

query node.

For better illustration, Figure 5.6 shows how the extraction output would look like in the semantic case[3]. The presented ontology was designed only for the illustration. Schema of the ontology can be seen in the Figure 5.7. It 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.

The last illustration is a SPARQL query (Figure 5.8) that would display a table of fatalities present in extracted RDF data. The query is based on the previous ontology and it demonstrates possible use of the schema and the extracted data.

5.2 Machine Learning of Extraction Rules

In this section we present our method for information extraction and annotation of texts, which is based on a deep linguistic analysis and Inductive Logic Programming (ILP) and implemented with the great help of the GATE framework. This approach is quite novel because it directly combines deep linguistic parsing with machine learning (ML). This combination and the use of ILP as a ML engine have following benefits: Manual selection of learning features is not needed. The learning procedure has full available linguistic information at its disposal and it is capable to select relevant parts itself. Extraction rules learned by ILP can be easily visualized, understood and adapted by human.

5.2.1 Data Flow

Similarly to the previous case with manually designed rules, also this extraction method was designed as a method for extraction of information from web resources and the goal is in extracted data in the form of semantic web ontology. The schema of the extraction process is different than the one presented in the previous section because it includes also the learning phase when extraction rules are learned from a learning collection. The present schema can be found on Figure 5.9, it is an adaptation of the general schema presented in Section 4.5.3. The schema is a little complicated because it illustrates two cases (or phases) in one picture:

1. the learning phase and
2. the application phase when existing extraction rules are applied to new documents and new information is being extracted.

Learning Phase

We will start the description with the learning phase and we will start on the Web (upper left corner of the schema). We need to obtain a learning collection of texts. They can be found on the web (the violet cloud), downloaded and text has to be extracted form them (the texts image). Now we need a human annotator (the worker image bellow), who will look at each of the texts and create gold standard annotations for these texts. These steps can be done using the GATE framework and we will not describe them further.

Before we can execute the learning procedure of Inductive Logic Programming (green circle), three more steps have to be done:

[3]The same data will be used in the next chapter in the example of raw extraction output (Figure 6.5).

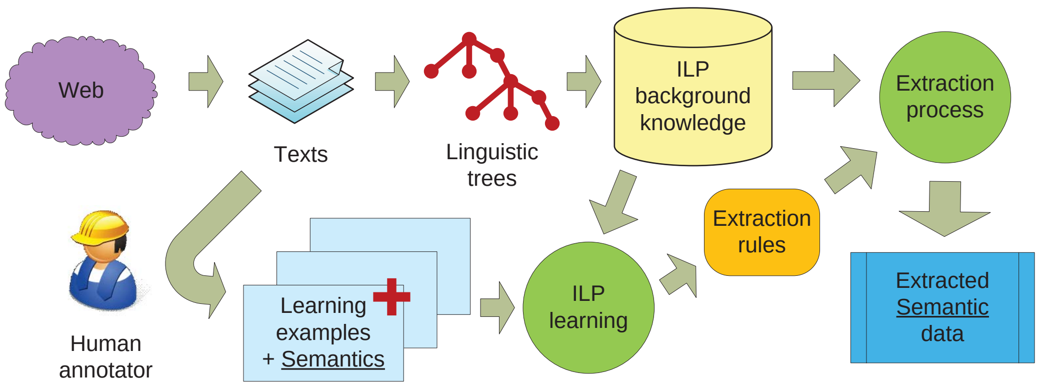


Figure 5.9: ILP data flow.

1. perform automated linguistic analysis, which will construct linguistic trees from the texts,

2. transform these linguistic trees to ILP background knowledge and

3. transform the gold standard annotations to ILP learning examples.

These steps will be described in following sections and in the next chapter about implementation (Section 6.2). The ILP learning procedure will produce extraction rules (orange oval) that will be used in the application phase.

Application Phase

Again, let us start the description on the web. Assume that target web pages for extraction have been identified, text extracted from them and linguistic trees constructed from the text. In this phase, we do not need ILP, but ordinary Logic Programming is still used for deciding which linguistic trees are covered by extraction rules. Therefore, again, the linguistic trees have to be transformed to ILP background knowledge. The previously learned extraction rules are applied on the ILP background knowledge constructed from the new trees during the main extraction process (green circle in the top right corner of the schema) and it will produce new extracted data with the same semantics that was used during manual annotation.

5.2.2 Closer Investigation

The description of the data flow schema in the previous section was quite general, not containing any implementation specific details. In this section, we want to provide more details about the actual realization of that general data flow because some interesting problems are connected with it.

First of all, the interface of this extraction method is different from the previously described method based on manually designed extraction rules. The previous method was realized as purely information extraction method while this method performs document annotation and therefore the correspondence of extracted information with its position in text has to be preserved, see details in following sections.

Another difference with the previous approach is that the previous approach is based on Netgraph and Netgraph is responsible for the management of linguistic trees in that case. The present method is based on GATE and ILP and these tools do not provide any special functions for working with linguistic trees. These functions were added by us and they provide:

- conversion of PDT linguistic trees to GATE annotations (see details in Section 6.2.2) and the possibility of calling TectoMT linguistic analysis directly from GATE (see details in Section 6.2.1) and
- integration of Prolog and ILP with GATE (see details in Section 6.2.4), which includes conversion of GATE annotations and the linguistic tree structure to ILP background knowledge (see details in Section 6.2.5).

5.2.3 Correspondence of GATE Annotations with Tree Nodes

On the one hand, our extraction method is based on linguistic trees that are composed of individual tree nodes, but on the other hand, we are performing document annotation using GATE, which means that annotations can occupy any segment of text, e.g. a segment starting or ending in the middle of some word. The correspondence of tree nodes with individual words (or tokens) of text is quite clear[4] but the learning algorithm is not capable to deal with partial word matches. This is not a problem in the case when annotations discovered by the extraction technique are put to the text because all the annotations will simply occupy whole words only. The problem can occur when ILP learning data are constructed from GATE annotations. During this construction, only those tree nodes that are completely covered by corresponding annotations will be accordingly marked in the ILP learning data.

Let us for example have a phrase “eight thousand Crowns'' and an annotation labeled as “amount'' that will almost cover the whole phrase up to the word “Crowns'', which will be covered only partly, without the ending “s''. Thus the annotated text will be “eight thousand Crown''.[5] During the construction of ILP learning data, only the tree nodes corresponding to words “eight'' and “thousand'' will be marked as “amount'', not the node that corresponds with the word “Crowns''.

Fortunately, this phenomenon does not occur very often, in fact we have not met it in our experiments so far. But there is another issue that is connected with multi-word annotations and it is quite common. See in the next section.

5.2.4 Root/Subtree Preprocessing/Postprocessing

Sometimes annotations span over more than one token. This situation complicates the process of machine learning and this situation is often called as “chunk learning”. Either we have to split a single annotation to multiple learning instances and after the application of extraction rules we have to merge them back together, or we can change the learning task from learning annotated tokens to learning borders of annotations (start tokens and end tokens). The later approach is implemented in GATE in *Batch Learning PR*[6] in the ‘SURROUND’ mode.

We have used another approach to solve this issue. Our approach is based on syntactic structure of a sentence and we call it “root/subtree preprocessing/postprocessing”. The

[4]Tectogrammatical tree nodes make it a little more difficult, see details in Section 4.1.1.

[5]Such annotation could be created by an uninformed annotator who could think that currencies should be marked as in singular form.

[6]`http://gate.ac.uk/userguide/sec:ml:batch-learning-pr`

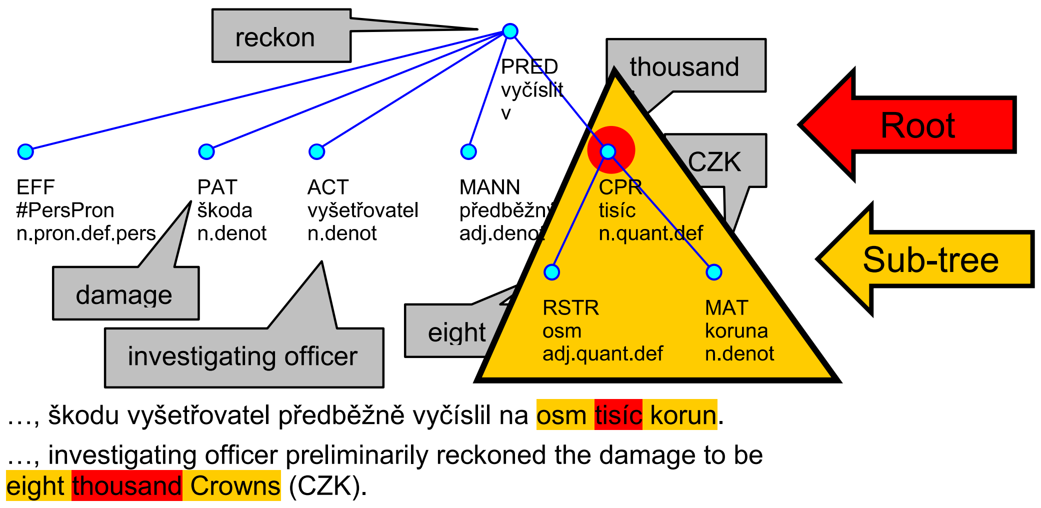


Figure 5.10: Root/Subtree Preprocessing/Postprocessing example, see also Figure 1.3 with the original tree.

idea is based on the observation that tokens of a multi-token annotation usually have a common parent node in a syntactic tree. So we can

1. extract the parent nodes (in dependency linguistics this node is also a token and it is usually one of the tokens inside the annotation),
2. learn extraction rules for parent nodes only and
3. span annotations over the whole subtrees of root tokens found during the application of extraction rules.

We call the first point as *root preprocessing* and the last point as *subtree postprocessing*. The situation is illustrated on Figure 5.10. We have successfully used this technique for the 'damage' task of our evaluation corpus (see Section 8.2 for details.)

5.2.5 Learning on Named Entity Roots

Another technique, that we have used to improve the efficiency of ILP machine learning, is closely connected with named entities. It can be used when target entities of an extraction task always (or most often) overlap with annotations of some available type (e.g. named entity). For example entities of the task 'acquired' always overlap with some named entity, or in other words, 'acquired' entities are merely a special case of named entities.

In this case, the extraction task can be rephrased as "decide which of the named entities belong to the extraction task." The collection of learning examples is reduced only to named entities and the learning algorithm is then faster and can be also more accurate.

In order to ILP had the opportunity to decide about these named entities, they have to be connected with additional information present in linguistic trees. But how to connect a potentially multi-token named entity with the nodes of linguistic trees? One possibility is to connect each named entity with all tree nodes it occupies but we have used a simpler solution when only the root token of a named entity is connected.

This solution was used for the majority of extraction tasks of the Acquisitions dataset; see evaluation details in Section 8.2.5.

5.2.6 Semantic Interpretation

Information extraction can solve the task "how to get documents annotated", but as we aim on the semantic annotation, there is a second step of "semantic interpretation" that has to be done. In this step we have to interpret the annotations in terms of a standard ontology. On a very coarse level this can be done easily. Thanks to GATE ontology tools [Bontcheva *et al.*, 2004] we can convert all the annotations to ontology instances with a quite simple JAPE [Cunningham *et al.*, 2000] rule, which takes the content of an annotation and saves it as a label of a new ontology instance or as a value of some property of a shared ontology instance. For example in our case of traffic and fire accidents, there will be a new instance of an accident class for each document and the annotations would be attached to this instance as values of its properties. Thus from all annotations of the same type, instances of the same ontology class or values of the same property would be constructed. This is very inaccurate form of semantic interpretation but still it can be useful. It is similar to the GoodRelation [Hepp, 2008] design principle of *incremental enrichment*[7]:

> "...you can still publish the data, even if not yet perfect. The Web will do the rest – new tools and people."

But of course we are not satisfied with this fashion of semantic interpretation and we plan to further develop the semantic interpretation step as a sophisticated "annotation → ontology" transformation process that we have proposed in one of our previous works [Dědek and Vojtáš, 2008].

5.3 Shareable Extraction Ontologies

In this section we present an extension of the idea of extraction ontologies that was originally presented by Embley *et al.* [2002]. We adopt the point that extraction models are kept in extraction ontologies and we add that the extraction ontologies should not be dependent on the particular extraction/annotation tool. In such case the extraction/annotation process can be done separately by an ordinary reasoner.

We present a proof of concept for the idea: a case study with our linguistically based IE engine and an experiment with several OWL reasoners. In the case study (see Section 5.3.2) the IE engine exports its extraction rules to the form of an extraction ontology. Third party linguistic tool linguistically annotates an input document and the linguistic annotations are translated to so-called document ontology. After that an ordinary OWL reasoner is used to apply the extraction ontology on the document ontology, which has the same effect as a direct application of the extraction rules on the document. The process is depicted in Figure 5.11 and it will be described in detail in Section 5.3.2.

The most closely related work was already presented in Section 3.2. The main idea will be illustrated on our case study in Section 5.3.2, its implementation in Section 6.3 and in Section 8.3 an experiment with several OWL reasoners and IE datasets will be presented. In Section 9.3.1 related issues are discussed and Section 9.3.5 concludes the discussion.

[7] `http://www.ebusiness-unibw.org/wiki/Modeling_Product_Models#Recipe:_.22Incremental_Enrichment.22`

5.3.1 Document Ontologies and Annotated Document Ontologies

The idea of shareable extraction ontologies assumes that extraction ontologies will be shareable and they can be applied on a document outside of the original extraction/annotation tool. We further assert that the extraction ontologies can be applied by ordinary reasoners. This assumption implies that both extraction ontologies and documents have to be in a reasoner readable format. In the case of contemporary OWL reasoners there are standard reasoner-readable languages: OWL and RDF in a rich variety of possible serializations (XML, Turtle, N-Triples, etc.) Besides that there exists standard ways like GRDDL or RDFa how to obtain a RDF document from an "ordinary document" (strictly speaking XHTML and XML documents).

We call 'document ontology' an ontology that formally captures content of a document. A document ontology can be for example obtained from the source document by a suitable GRDDL transformation (as in our experiment). A document ontology should contain all relevant data of a document and preferably the document could be reconstructed from the document ontology on demand.

When a reasoner is applying an extraction ontology to a document, it only has "to annotate" the corresponding document ontology, not the document itself. Here "to annotate" means to add new knowledge – new class membership or property assertions. In fact it means just to do the inference tasks prescribed by the extraction ontology on the document ontology.

Obviously when a document can be reconstructed from its document ontology (this is very often true, it is necessary just to save all words and formatting instructions) then also an annotated document can be reconstructed from its annotated document ontology.

5.3.2 The Main Idea Illustrated – a Case Study

In this section, realization of the main idea will be described and illustrated on a case study.

A schema of the case study is presented in Figure 5.11. The top row of the image illustrates how TectoMT (third party linguistic tool) linguistically annotates an input document and the linguistic annotations are translated to so-called document ontology by a GRDDL/XSLT transformation.

In the bottom of the picture our IE engine learns extraction rules and exports them to an extraction ontology. The reasoner in the middle is used to apply the extraction ontology on the document ontology and it produces the "annotated" document ontology, which was described in the previous section.

Implementation of the case study will be described in Section 6.3. It exploits the approaching support for Semantic Web Rule Language (SWRL) [Parsia *et al.*, 2005]. Although SWRL is not yet approved by W3C it is already widely supported by Semantic Web tools including many OWL reasoners. The SWRL support makes it much easier to transfer the semantics of extraction rules used by our IE tool to extraction ontology. The translation of the native extraction rules to SWRL rules that form the core of the extraction ontology will be also presented in Section 6.3.

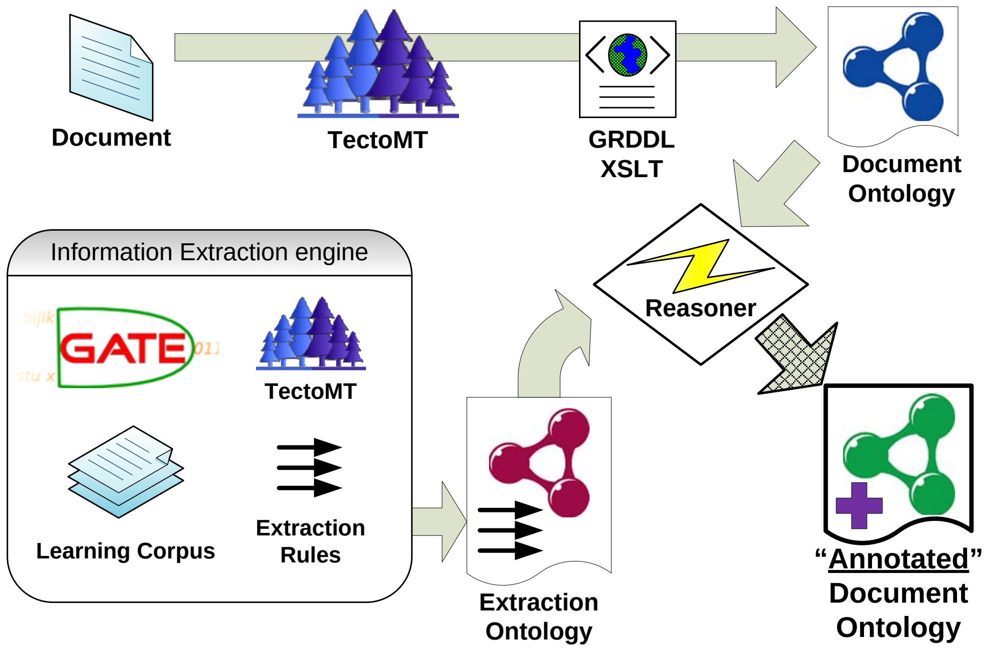


Figure 5.11: Semantic annotation driven by an extraction ontology and a reasoner – schema of the process.

5.4 Fuzzy ILP Classification

In this section, we study the problem of classification of textual reports. We are focused on the situation in which structured information extracted from the reports is used for such classification. We present a proposal and a partial implementation of an experimental classification system based on our previous work on information extraction (see Sections 5.1 and 5.2 for details) and fuzzy inductive logic programming (fuzzy ILP). Our description is based on a case study of seriousness classification of accident reports. We would like to have a tool that is able to classify the accident's degree of seriousness on the basis of information obtained through information extraction. In this section, we concentrate on the classification part and present a detailed study of the fuzzy ILP classification method (so-called 'Fuzzy ILP Classifier').

This problem represents a challenge of induction and/or mining on several occasions. First we need an inductive procedure when extracting attributes of an accident from text. Second (the subject of this section) we need an inductive procedure when trying to explain an accident's degree of seriousness by its attributes. ILP is used as the inductive procedure in both of these places.

- During the information extraction phase, we exploit the fact that ILP can work directly with multirelational data, e.g., deep syntactic (tectogrammatical) trees built from sentences of the processed text.
- During the classification phase, we are experimentally using the Fuzzy ILP Classifier because it performs quite well on our dataset (see Section 8.4) and it is naturally capable of handling fuzzy data, which occurs when the information extraction engine returns confidence probability values along with extracted data. But the description does not go so far and only the approach is fuzzy in the present demonstration.

The rest of this section is organized as follows: Design of the experimental system is presented in Section 5.4.1. Section 5.4.2 provides details about our case study, which is later used in examples. Formal models of the system (including several translations of a fuzzy ILP task to classical ILP) are presented in Section 5.4.3, followed by a description of implementation of the models in the system. In Section 6.4.1 we present the main results of the work, and then we evaluate and compare the methods with other well-known classifiers. Section 9.4 concludes the chapter.

See also Section 3.3, where some closely related works were introduced.

5.4.1 Data Flow

A general schema of the experimental system is shown in Figure 5.12. In the schema, previously developed information extraction tools based on third party linguistic analyzers are used (the upper two dashed arrows). The information extraction tools are supposed to extract structured information from the reports and the extracted information is then translated to ILP background knowledge and, along with a user rating, it is used for the classification (we assume that a small amount of learning data is annotated by a human user). The classification is based on ILP and it could be *fuzzy* or *crisp*. Crisp denotes a straightforward application of ILP and fuzzy stands for the fuzzy method (subject of the present description), see in the next sections.

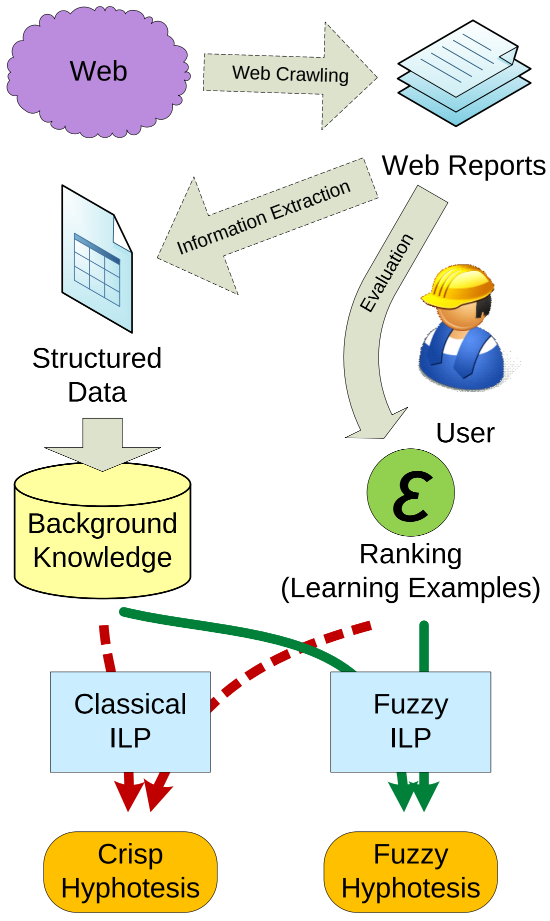


Figure 5.12: Schema of the experimental system.

5.4.2 The Case Study – Accident Seriousness Classification

The main experiment leads to the seriousness classification of an accident presented in a report.

For the experiment a classification dataset (Section 7.3.6) was built based on a collection of 50 textual reports. We have identified several features presented in these reports and manually extracted the corresponding values. To each report we have also assigned a value of overall ranking of seriousness of the presented accident, which is the target of the classification. The dataset is described in Section 7.3.6.

In the experiment, we have not used any information extracted by our automated information extraction tools. Instead, we concentrate on the classification; the actual source of the information is not so important for us and the integration step still lies ahead.

5.4.3 Translation of Fuzzy ILP Task to Several Classical ILP Tasks

Formal definitions of classical and fuzzy ILP tasks were presented in Section 4.7. In this section, additional formal constructions will be presented, because they are the basis for the implementation.

As far as there is no implementation of fuzzy ILP, we have to use a classical ILP system. Fortunately, any fuzzy ILP task can be translated to several classical ILP tasks (subject to some rounding and using a finite set of truth values).

Moreover, GAP – Generalized Annotated Programs [Kifer and Subrahmanian, 1992] are used in practice, so graded formulas will sometimes be understood as annotated (with classical connectives and with a more complex annotation of the heads of rules). This is possible because Krajci *et al.* [2004] showed that (some extension of) fuzzy logic programming is equivalent to (some restriction of) generalized annotated programs.

The proposed experimental classification system is based on two ILP classification methods – crisp and fuzzy. Technically, the difference between the approaches consists in a different setting of the underlying classical ILP tasks. In the following text, translation of the original fuzzy ILP task to the two classical ILP tasks will be presented; the first will be denoted as *crisp*, the second as *monot* (because so called monotonization technique is used). The original **fuzzy** ILP task can be informally expressed as: "From a degree of injury, a degree of damage, a degree of duration, etc. determine the degree of seriousness."

In the following text, assume that all fuzzy sets take truth values only from a finite set of truth values $T : \{0, 1\} \subseteq T \subseteq [0, 1]$.

Definition 3 (Transformation of background knowledge)**.** A fuzzy background knowledge $\mathcal{B} : B \longrightarrow [0, 1]$ is given. For each predicate $p(x)$ in B we insert an additional attribute t to express the truth value, thus we have created a new predicate $p(x, t)$. We construct two classical background knowledge sets B_T^{crisp} and B_T^{monot} as follows:

- The first (B_T^{crisp}) is a direct coding of a fuzzy value by an additional attribute: If $\mathcal{B}(p(x)) = t,\ t \in T$, then we add $p(x, t)$ to B_T^{crisp}.
- The second (B_T^{monot}) is obtained by a process called monotonization: If $\mathcal{B}(p(x)) = t,\ t \in T$, then for all $t' \in T,\ t' \leq t$ we add $p(x, t')$ to B_T^{monot}. This corresponds to the natural meaning of truth values t.

Additionally, example sets are constructed in two ways.

Definition 4 (Transformation of examples)**.** Given is a fuzzy set of examples $\mathcal{E} : E \longrightarrow [0, 1]$. For all $t \in T$ we construct two classical sets of examples E_t and $E_{\geq t}$ as follows:

- $E_t = P_t \cup N_t$, where $e \in P_t\ \ iff\ \ \mathcal{E}(e) = t$ and N_t is the rest of E.
- $E_{\geq t} = P_{\geq t} \cup N_{\geq t}$, where $e \in P_{\geq t}\ \ iff\ \ \mathcal{E}(e) \geq t$ and N_t is the rest of E.

These two translations create two classical ILP tasks for each truth value $t \in T$, the first one is crisp and the second one (*monot*) can be understood as (and translated back to) fuzzy ILP.

- The *crisp ILP task* is given by B_T^{crisp} and E_t for each $t \in T$. As a result, it produces a set of hypotheses H_t.
- The *monot ILP task* is given by B_T^{monot} and $E_{\geq t}$ for each $t \in T$. As a result, it produces a set of hypotheses $H_{\geq t}$ guaranteeing examples of a degree of at least t.

Note that among variable boundings in B there are no boundings on the truth value attribute which was added to each predicate; hence, there are no variable boundings on the truth value attribute in $H_{\geq t}$. We did not add an additional truth value attribute to the predicates in E.

Now we sketch the translation of the *monot ILP task* to GAP (fuzzy ILP) rules.

Theorem 1 (Translation of the *monot ILP task*)**.** A fuzzy ILP (or equivalent GAP) task is given by $\mathcal{E}$ and $\mathcal{B}$. Let us assume that C is the target predicate in the domain of $\mathcal{E}$ and for each $t \in T$, $H_{\geq t}$ is a correctly learned solution of the corresponding *monot ILP task* according to the definitions introduced above. We define $\mathcal{H}$ consisting of one GAP rule:

$$C(y) : u(x_1, \dots, x_m) \leftarrow B_1(y) : x_1 \,\&\dots\&\, B_m(y) : x_m,$$

here $B_1 : x_1\& \dots \&B_m : x_m$ is the enumeration of all predicates in B.

Assume that $B_1(y_1, t_1), \dots, B_n(y_n, t_n)$ are some of the predicates in B (for simplicity enumerated from 1 to n, $n \leq m$). Then for each rule

$$R = C_{\geq t}(y) \Leftarrow B_1(y, t_1), \dots, B_n(y, t_n)$$

in $H_{\geq t}$ ($C_{\geq t}$ is the monotonized target predicate) we give a constraint in the definition of u as follows:

$$U_R = u(x_1, \dots, x_m) \geq t \text{ if } x_1 \geq t_1, \dots, x_n \geq t_n.$$

Note that all x_i and y are variables, t_i and t are constants and $x_{n+1}, \dots, x_m$ have no restrictions. We learn the function u as a “monotone” completion of the rules.

We claim that if all $H_{\geq t}$ were correctly learned by a classical ILP system, then, for the minimal solution u of all constraints U_R, the rule

$$C(y) : u(x_1, \dots, x_m) \leftarrow B_1(y) : x_1 \,\&\dots\&\, B_m(y) : x_m$$

is a correct solution of the fuzzy ILP task given by $\mathcal{E}$ and $\mathcal{B}$, for all $R \in H_{\geq t}$ and $t \in T$.

Illustration Example

In our case `serious(Accident_ID)` is the fuzzy or GAP target predicate C. Let for example $t = 3$ and $H_{\geq t}$ be the same as in Figure 6.20 (the last two (25, 26) rows correspond with $H_{\geq 3}$). Then `serious_atl_3(Accident_ID)` is the monotonized target predicate $C_{\geq t}$ and B_1, B_2 and u are realized as follows:

$B_1(y, t_1)$ …`fatalities_atl(Accident_ID, 1)`,

$B_2(y, t_2)$ …`damage_atl(Accident_ID, 1500000)`,

u is an arbitrary function of m arguments (m is the number of all predicates used in background knowledge) for which following restrictions hold true:

for $t = 3$:

$u(x_1, \dots, x_m) \geq 3$ if $x_1 \geq 1$

$u(x_1, \dots, x_m) \geq 3$ if $x_2 \geq 1500000$

and similar restrictions for $t = 1, 2$.

Our presentation is slightly simplified here and we freely switch between fuzzy and GAP programs, which are known to be equivalent, see in [Krajci *et al.*, 2004].

6. Implementation

This chapter provides the most important implementation details, it is, again, divided into four main sections that correspond the four topics of this thesis.

6.1 Manual Design of Extraction Rules

6.1.1 Procedural Extraction Rules

Our earliest extraction method, which is based on procedural extraction rules, was implemented using Btred[1] API for processing linguistic trees. The extraction rules were implemented in Perl as Btred procedures. Application of these extraction rules on a corpus of linguistic trees is realized such that each procedure (or extraction rule) is executed on every available tree by Btred.

An example of such extraction rule is in Figure 6.1 and corresponding extraction output on Figure 6.2. Let us briefly describe this extraction rule. The execution is configured such that the procedure of a particular extraction rule (‘print_injured’ on the line 5 in the presented case) is repeatedly called for all nodes of all available trees and the variable ‘$this‘ always contains a reference to the actual tree node.

The procedure starts with testing if the actual node is a verb (line 6) and if it is one of the intended verbs (lines 7, 8 and 3). If these tests succeed, the rule is applied and following information is printed to the output:

<action> enclosing element and its type (line 10) : the actual verb, that triggered this extraction rule, is used as the type of the action.

<sentence> (lines 13, 14) : the full sentence and its *id*.

<negation> (lines 17-21) : test whether negation is present or not (e.g. “truck driver ***did not*** survive the accident.”)

<manner> (lines 24-29) : looks for a word expressing the manner of injury (e.g. “driver was ***seriously*** injured.”)

<participant> (lines 32-51) : looks for all participants attached to the action.

<quantity> of participants (lines 39-44) : looks for a numeral expressing the quantity of participants (e.g. “***two*** people died.”)

<full_string> representation of given participant (lines 46-48) : all words attached to the particular participant node (e.g. “***fifteen years old woman*** died.”)

6.1.2 Netgraph Based Extraction Rules

Our second extraction method, which is based on Netgraph (see Section 4.2.4 for details), is implemented in Java. Java was chosen partly because we use the Java implementation of Netgraph client as a library.

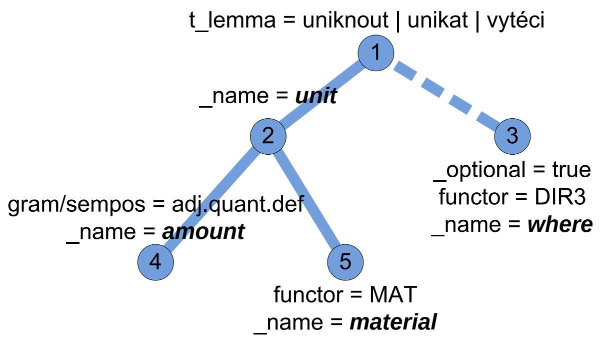
[1]See details about Btred in Section 4.2.2.

```perl
#variable $this contains currently processed node, $root current root node

my @injure_verbs = ("zranit", "usmrtit", "zemřít", "zahynout", "přežít");
# transcript:        "injure"  "kill"     "die"     "perish"    "survive"
sub print_injured {
  if ($this->{gram}{sempos} eq "v") {
    foreach my $v (@injure_verbs) {
      if ($this->{t_lemma} eq $v ) {
        #action type
        print "<action type=\"" . $this->{t_lemma} . "\">";

        #sentece
        print "<sentece>" . PML_T::GetSentenceString($root) . "</sentece>";
        print "<sentece_id>" . $root->{id} . "</sentece_id>";

        #negation
        if (test_negation($this)) {
          print "<negation>true</negation>" ;
        } else {
          print "<negation>false</negation>" ;
        }

        #manner of injury
        my @mans = find_node_by_attr_depth($this, 0, 'functor', '^MANN');
        if (@mans) {
          foreach my $m (@mans) {
            print "<manner>"; print $m->{t_lemma}; print "</manner>";
          };
        }

        #actors and patients
        my @pats = find_node_by_attr($this, 'functor', '^[PA][AC]T');
        @pats = &filter_list(\&test_person, @pats);

        foreach my $p (@pats) {
          print "<participant type=\"" . $p->{t_lemma} . "\">";

          #participants count
          my @cnt = find_node_by_attr($p, 'functor', '^RSTR');
          @cnt = &filter_list(\&test_number_lemma, @cnt);
          my $cnt1 = pop(@cnt);
          print "<quantity>" .
            &test_number($cnt1->{t_lemma}) .
            "</quantity>" if ($cnt1);

          print "<full_string>";
          print_subtree_as_text($p);
          print "</full_string>";

          print "</participant>";
        }

        #action end
        print "</action>\n";
}}}}
```

Figure 6.1: Procedurally written extraction rule in *Btred*.

```xml
<injured_result>
  <action type="zranit">
    <sentece>
      Při požáru byla jedna osoba lehce zraněna -- jednalo se
      o majitele domu, který si vykloubil rameno.
    </sentece>
    <sentece_id>T-vysocina63466.txt-001-p1s4</sentece_id>
    <negation>false</negation>
    <manner>lehký</manner>
    <participant type="osoba">
      <quantity>1</quantity>
      <full_string>jedna osoba</full_string>
    </participant>
  </action>
  <action type="zemřít">
    <sentece>
      Ve zdemolovaném trabantu na místě zemřeli dva muži -- 82letý
      senior a další muž, jehož totožnost zjišťují policisté.
    </sentece>
    <sentece_id>T-jihomoravsky49640.txt-001-p1s4</sentece_id>
    <negation>false</negation>
    <participant type="muž">
      <quantity>2</quantity>
      <full_string>dva muži</full_string>
    </participant>
  </action>
    <action type="zranit">
    <sentece>čtyřiatřicetiletý řidič nebyl zraněn.</sentece>
    <sentece_id>T-jihomoravsky49736.txt-001-p4s3</sentece_id>
    <negation>true</negation>
    <participant type="řidič">
      <full_string>čtyřiatřicetiletý řidič</full_string>
    </participant>
  </action>
</injured_result>
```

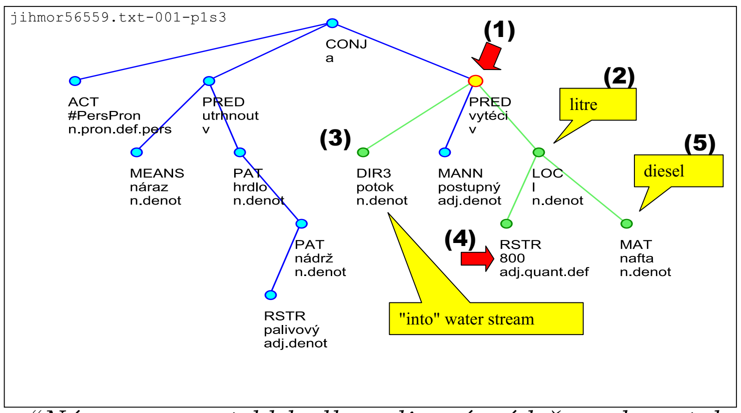
Figure 6.2: *XML* structured output of the query written in *Btred*.



Transcript:

uniknout, unikat to leak out	vytéci to flow out

Figure 6.3: A manually created extraction rule investigating dangerous liquids that spilled out into the environment.



Original sentence: *"Nárazem se utrhl hrdlo palivové nádrže a do potoka postupně vyteklo na 800 litrů nafty."*
English transcript: *"Due to the clash the throat of fuel tank tore off and 800 liters of oil (diesel) has run out to a stream."*

Figure 6.4: A tree matching with the corresponding extraction rule in Figure 6.3.

The extraction is implemented as follows: Netgraph implementation is responsible for the evaluation of the Netgraph query part of extraction rules and matching trees are then returned to our implementation, which prepares the extraction output based on the SELECT part of extraction rules.

Illustration Examples

Evaluation of the extraction rule from Figure 6.3 is illustrated on Figure 6.4. Figure 6.4 shows a linguistic tree matching with the extraction rule. Matching nodes are decorated and labeled by the numbers of corresponding query nodes.

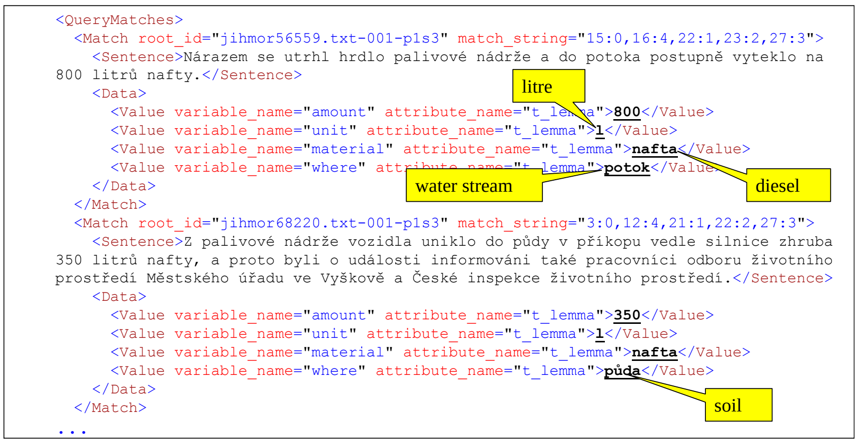
6.1.3 Extraction Output

Small pieces of extraction outputs are shown in Figure 6.5 (for the extraction rule in Figure 5.3) and in Figure 6.6 (for the extraction rule in Figure 6.3).

The former example (Figure 6.5) contains three matches of the extraction rule in three different articles. Each query match is closed in the `<Match>` element and each contains

```xml
1  <QueryMatches>
2    <Match root_id="T-vysocina63466.txt-001-p1s4" match_string="2:0,7:3,8:4,11:2">
3      <Sentence>
4        Při požáru byla jedna osoba lehce zraněna - jednalo se
5        o majitele domu, který si vykloubil rameno.
6      </Sentence>
7      <Data>
8        <Value variable_name="action_type" attribute_name="t_lemma">zranit</Value>
9        <Value variable_name="injury_manner" attribute_name="t_lemma">lehký</Value>
10       <Value variable_name="participant" attribute_name="t_lemma">osoba</Value>
11       <Value variable_name="quantity" attribute_name="t_lemma">jeden</Value>
12     </Data>
13   </Match>
14   <Match root_id="T-jihomoravsky49640.txt-001-p1s4" match_string="1:0,13:3,14:4">
15     <Sentence>
16       Ve zdemolovaném trabantu na místě zemřeli dva muži - 82letý senior
17       a další muž, jehož totožnost zjišťují policisté.
18     </Sentence>
19     <Data>
20       <Value variable_name="action_type" attribute_name="t_lemma">zemřít</Value>
21       <Value variable_name="participant" attribute_name="t_lemma">muž</Value>
22       <Value variable_name="quantity" attribute_name="t_lemma">dva</Value>
23     </Data>
24   </Match>
25   <Match root_id="T-jihomoravsky49736.txt-001-p4s3" match_string="1:0,3:3,7:1">
26     <Sentence>Čtyřiatřicetiletý řidič nebyl zraněn.</Sentence>
27     <Data>
28       <Value variable_name="action_type" attribute_name="t_lemma">zranit</Value>
29       <Value variable_name="a-negation"
30              attribute_name="m/tag">VpYS---XR-NA---</Value>
31       <Value variable_name="participant" attribute_name="t_lemma">řidič</Value>
32     </Data>
33   </Match>
34 </QueryMatches>
```

Figure 6.5: *XML* structured output of the SQL select like query. A negation can be detected from the presence of *m/tag* on the line 30.



```xml
<QueryMatches>
  <Match root_id="jihmor56559.txt-001-p1s3" match_string="15:0,16:4,22:1,23:2,27:3">
    <Sentence>Nárazem se utrhl hrdlo palivové nádrže a do potoka postupně vyteklo na
800 litrů nafty.</Sentence>
    <Data>
      <Value variable_name="amount" attribute_name="t_lemma">800</Value>
      <Value variable_name="unit" attribute_name="t_lemma">l</Value>
      <Value variable_name="material" attribute_name="t_lemma">nafta</Value>
      <Value variable_name="where" attribute_name="t_lemma">potok</Value>
    </Data>
  </Match>
  <Match root_id="jihmor68220.txt-001-p1s3" match_string="3:0,12:4,21:1,22:2,27:3">
    <Sentence>Z palivové nádrže vozidla uniklo do půdy v příkopu vedle silnice zhruba
350 litrů nafty, a proto byli o události informováni také pracovníci odboru životního
prostředí Městského úřadu ve Vyškově a České inspekce životního prostředí.</Sentence>
    <Data>
      <Value variable_name="amount" attribute_name="t_lemma">350</Value>
      <Value variable_name="unit" attribute_name="t_lemma">l</Value>
      <Value variable_name="material" attribute_name="t_lemma">nafta</Value>
      <Value variable_name="where" attribute_name="t_lemma">půda</Value>
    </Data>
  </Match>
...
```

Figure 6.6: *XML* structured output of the SQL select like query corresponding with the extraction rule in Figure 6.3 and matching tree in Figure 6.4.

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.

In the case of the example query, values identified by the variable `action_type` specify the type of the action. So in the first and third case somebody was injured (*zranit* means to injure in Czech, lines 8 and 28) and in the second case somebody died (*zemřít* means to die in Czech, line 20).

Values identified by `participant` and `quantity` contain information about participants of the action. `participant` serves for specification of the type of the participants and `quantity` values hold numbers (quantity) of the participants. So in the first action one (*jeden*, line 11) person (*osoba*, line 10) was injured and in the second action two (*dva*, line 22) men (*muž*, line 21) died.

Values identified by `a-negation` contain the information about a negation of a clause (The presence of negation is indicated by the 11th character of the position-based morphological tag, note that the corresponding node (number 2) of the extraction rule is marked as optional and the restriction on m/tag is put in the form of regular expression on the 11th character.) So we can see that the participant (driver – *řidič*, line 31) of the last action was **not** injured (lines 29-30).

The last not described attribute name is `injury_manner`. Corresponding values contain information about the manner of injury of an injury action. So in the first action of the example there was a light injury (*lehký* means light in Czech, line 9).

6.2 Machine Learning of Extraction Rules

Here we just briefly describe implementation of our system. The system consists of several modules, all integrated in GATE as processing resources (PRs).

6.2.1 TectoMT Wrapper (Linguistic Analysis)

TectoMT wrapper is a GATE component (processing resource), which takes the text of a GATE document, sends it to TectoMT linguistic analyzers, parses the results and converts the results to the form of GATE annotations. The next section provides details about how PDT annotations are represented in GATE.

Because TectoMT has to run as a separate process (it is implemented in Perl) and the initialization of TectoMT analyzers usually takes significant amount of time it would be very inefficient to start a new TectoMT instance for each document. Therefore the implementation currently offers two modes of execution: batch (TectoMTBatchAnalyser) and online (TectoMTOnlineAnalyser).

The batch mode is implemented similarly to the Batch Learning PR[2]. Batch mode of execution in the context of GATE corpus pipelines is a deviation from the standard execution mode, when documents are processed one by one. During the execution as a part of a corpus pipeline, TectoMTBatchAnalyser only accumulates documents and the whole work is done as a batch when the last document is encountered. This also implies that TectoMTBatchAnalyser has to be the last PR in the pipeline because it produces no output in the time of execution (except for the last document when the whole batch is executed).

Client-server model of implementation is used in the online mode. A separate TectoMT

[2]`http://gate.ac.uk/userguide/sec:ml:batch-learning-pr`

server process is started at the time of initialization and during the execution, GATE documents are processed in ordinary way, one by one. This means that (similarly to the previous case) each document is converted to the TectoMT readable format, sent to TectoMT and the result is converted back to GATE. The online mode of execution is a bit slower than the batch mode because additional time is spent on client-server communication (XML-RPC[3]).

6.2.2 PDT Annotations in GATE

Although GATE annotations are just highlighted pieces of text (see also Section 4.5.1) it is possible to use them to encode dependency tree structures. It is possible because each GATE annotation has a unique identifier (ID) and an arbitrary set of features (name-value pairs) can be assigned to it. The way how the PDT dependency trees are encoded in GATE is in fact the same as in the GATE wrapper for the Stanford Parser[4].

Three main constructs are used to capture an arbitrary configuration of a linguistic dependency tree:

tree nodes (usually corresponding to words (tokens) of a sentence)

edges (dependency relations between nodes)

node attributes (connected linguistic features like POS, gender, tense, case, etc.)

These constructs are encoded in GATE in the following way: tree nodes correspond to *token* annotations, node attributes are saved as token annotation features and edges are encoded as another special kind of annotations.

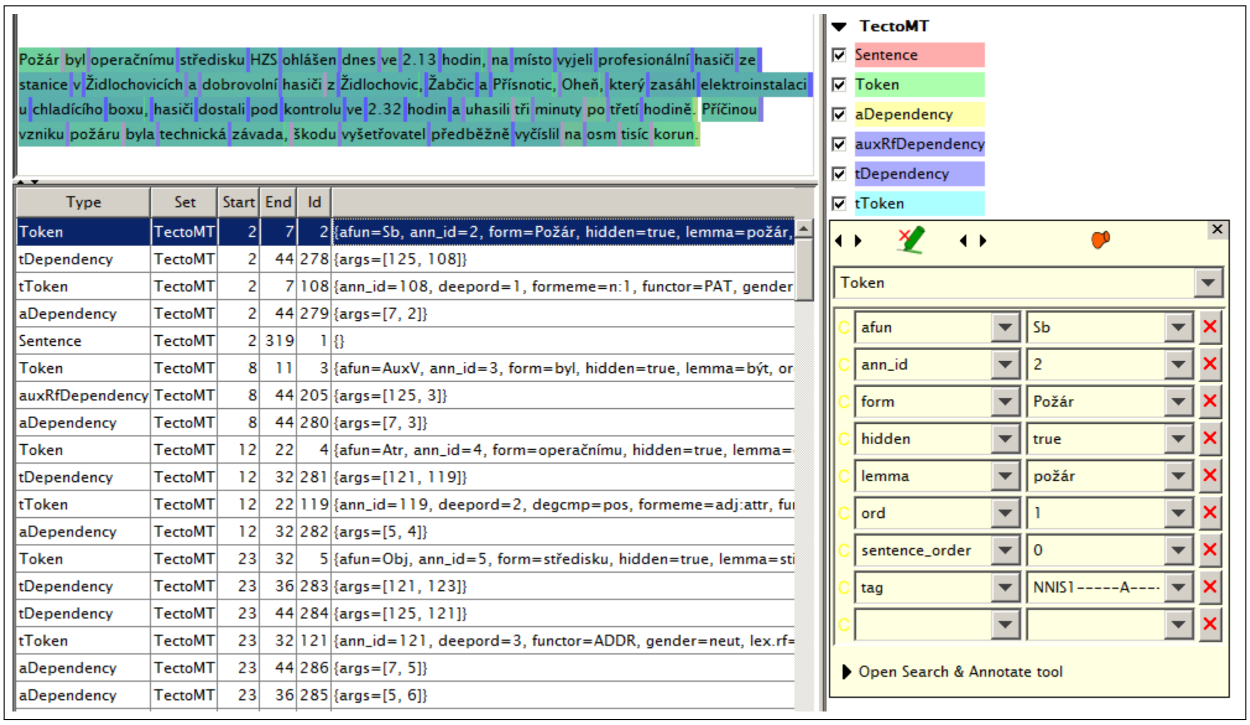
Two kinds of token annotations are used to represent two kinds of trees and tree nodes. “Token” annotation type is used for analytical tree nodes and “tToken” for tectogrammatical tree nodes.

Four kinds of edges (PDT dependencies) are implemented by the TectoMT wrapper: analytical dependencies, tectogrammatical dependencies, aux.rf (auxiliary reference) and lex.rf (main lexical reference). The last two kinds (aux.rf and lex.rf) are used to connect tectogrammatical and analytical nodes. The implementation differs according to the cardinality of a dependency type. The first three kinds are of the cardinality one-to-many (one parent node can have many children nodes) and the last one (lex.rf) if of the cardinality one-to-one (one parent node has at most one child). Because of that, lex.rf edges can be stored as features (with the name “lex.rf”) of “tToken” annotations. Note that a single GATE annotation feature can have at most one value in each annotation. In this case the annotation ID of the referenced “Token” annotation (referenced analytical node) is the value of the lex.rf feature.

One-to-many dependencies are stored as separate annotations (type names: “aDependency”, “tDependency”, “aux.rf”) with a single feature called “args”. Values of this feature are of Java type List<Integer> (list of integers). The list always contains just two items. The first one is the annotation ID of the parent annotation; the second one is the ID of the child annotation. Instead of using one list feature, two simple features (like “arg1”, “arg2” or “parentID”, “childID”) could be used, but the implementation is consistent with

[3]http://www.xmlrpc.com/

[4]http://gate.ac.uk/userguide/sec:parsers:stanford


Požár byl operačnímu středisku HZS ohlášen dnes ve 2.13 hodin, na místo vyjeli profesionální hasiči ze stanice v Židlochovicích a dobrovolní hasiči z Židlochovic, Žabčic a Přísnotic. Oheň, který zasáhl elektroinstalaci u chladícího boxu, hasiči dostali pod kontrolu ve 2.32 hodin a uhasili tři minuty po třetí hodině. Příčinou vzniku požáru byla technická závada, škodu vyšetřovatel předběžně vyčíslil na osm tisíc korun.

Type	Set	Start	End	Id	
Token	TectoMT	2	7	2	{afun=Sb, ann_id=2, form=Požár, hidden=true, lemma=požár,
tDependency	TectoMT	2	44	278	{args=[125, 108]}
tToken	TectoMT	2	7	108	{ann_id=108, deepord=1, formeme=n:1, functor=PAT, gender
aDependency	TectoMT	2	44	279	{args=[7, 2]}
Sentence	TectoMT	2	319	1	{}
Token	TectoMT	8	11	3	{afun=AuxV, ann_id=3, form=byl, hidden=true, lemma=být, or
auxRfDependency	TectoMT	8	44	205	{args=[125, 3]}
aDependency	TectoMT	8	44	280	{args=[7, 3]}
Token	TectoMT	12	22	4	{afun=Atr, ann_id=4, form=operačnímu, hidden=true, lemma=
tDependency	TectoMT	12	32	281	{args=[121, 119]}
tToken	TectoMT	12	22	119	{ann_id=119, deepord=2, degcmp=pos, formeme=adj:attr, fu
aDependency	TectoMT	12	32	282	{args=[5, 4]}
Token	TectoMT	23	32	5	{afun=Obj, ann_id=5, form=středisku, hidden=true, lemma=sti
tDependency	TectoMT	23	36	283	{args=[121, 123]}
tDependency	TectoMT	23	44	284	{args=[125, 121]}
tToken	TectoMT	23	32	121	{ann_id=121, deepord=3, functor=ADDR, gender=neut, lex.rf=
aDependency	TectoMT	23	44	286	{args=[7, 5]}
aDependency	TectoMT	23	36	285	{args=[5, 6]}

TectoMT

- ✓ Sentence
- ✓ Token
- ✓ aDependency
- ✓ auxRfDependency
- ✓ tDependency
- ✓ tToken

Token

afun	Sb
ann_id	2
form	Požár
hidden	true
lemma	požár
ord	1
sentence_order	0
tag	NNIS1-----A----

Figure 6.7: PDT annotations in GATE (screenshot).

the wrapper for the Stanford Parser[5], which uses the single list feature called "args"), thus PDT dependencies are compatible with Stanford dependencies in GATE.

It is not simple to demonstrate the GATE representation of the dependencies in a static printed form; we can only show a GATE screenshot (Figure 6.7) that partly illustrates that.

6.2.3 Netgraph Tree Viewer in GATE

Because GATE does not provide a graphical viewer of dependency trees (Only phrase structure tree viewer is available[6].) and we are quite familiar with the Java implementation of Netgraph, we decided to integrate the Netgraph viewer with GATE. This is quite handy during the adaptation of the extraction method to a new dataset and investigation of learned extraction rules and matching trees.

Figure 6.8 shows a screenshot of the integrated Netgraph viewer displaying a dependency tree obtained by the Stanford Parser (a GATE plugin). Stanford dependencies were selected for this image to demonstrate the compatibility of Stanford and PDT dependency representations in GATE. Not only the Netgraph viewer but also the whole ILP based extraction method can handle either of the dependencies.

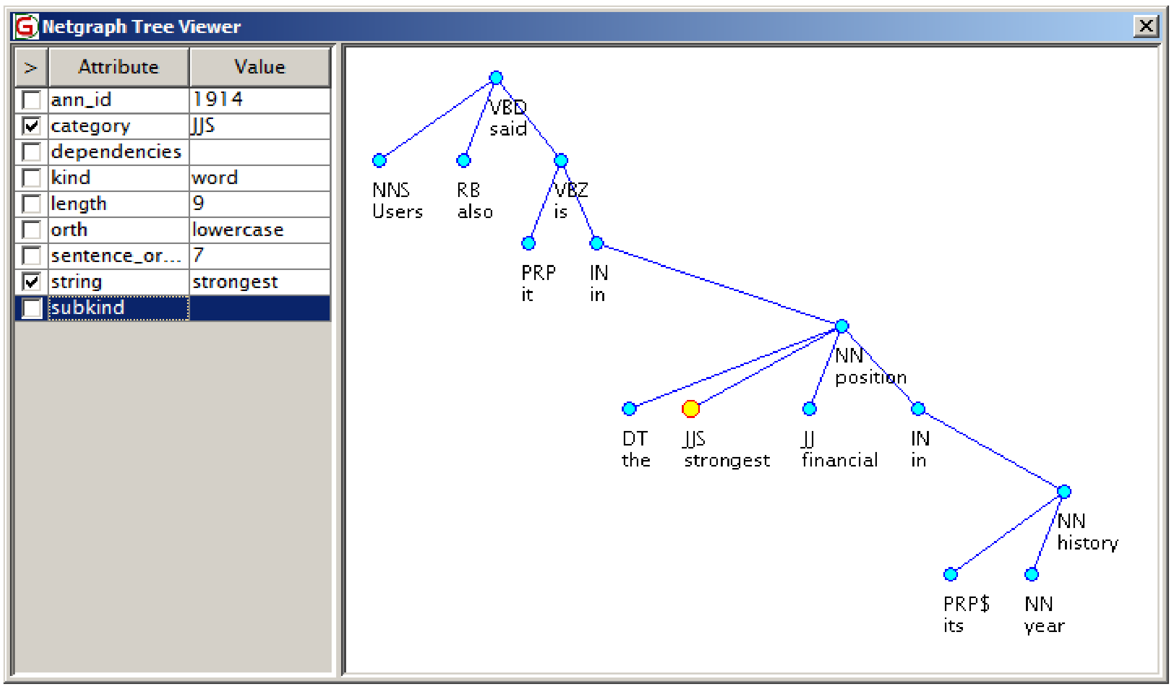
6.2.4 ILP Wrapper (Machine Learning)

After a human annotator have annotated several documents with desired target annotations, machine learning takes place. This consists of two steps:

1. learning of extraction rules from the target annotations and

2. application of the extraction rules on new documents.

[5]http://gate.ac.uk/userguide/sec:parsers:stanford
[6]http://gate.ac.uk/userguide/sec:parsers:supple:treeviewer


Netgraph Tree Viewer

>	Attribute	Value
☐	ann_id	1914
☑	category	JJS
☐	dependencies	
☐	kind	word
☐	length	9
☐	orth	lowercase
☐	sentence_or...	7
☑	string	strongest
☐	subkind	

Sentence: Users also said it is in the strongest financial position in its 24-year history.

Figure 6.8: Netgraph Tree Viewer in GATE (for Stanford Dependencies, screenshot).

In both steps the linguistic analysis has to be done before and in both steps ILP background knowledge (a logical database of facts) is constructed from linguistic structures of documents that are being processed. We call the process of background knowledge construction as *ILP serialization*; more details are presented below in Section 6.2.5.

After the ILP serialization is done, the next step depends on the phase which is being performed.

In the learning case, positive and negative examples are constructed from target annotations and the machine learning ILP inductive procedure is executed to obtain extraction rules.

In the application case a Prolog system is used to check if the extraction rules entail any of target annotation candidates.

The learning examples and annotation candidates are usually constructed from all document tokens, but it can be optionally changed to any other textual unit, for example only numerals, tectogrammatical nodes (words with lexical meaning) or named entities (see details about the technique of learning on named entity roots in Section 5.2.5) can be selected. This can be done easily with the help of *Machine Learning PR* (LM PR) from GATE[7].

ML PR provides an interface for exchange of features (including target class) between annotated texts and propositional learners in both directions – during learning as well as during application. We have used ML PR and developed our *ILP Wrapper* for it. The implementation was a little complicated because complex linguistic structures cannot be easily passed as propositional features, so in our solution we use the ML PR interface only for exchange of the class attribute and annotation id and we access the linguistic structures directly in a document.

[7] *Machine Learning PR* is an old GATE interface for ML and it is almost obsolete but in contrast to the new *Batch Learning PR* the LM PR is easy to extend for a new ML engine.

6.2.5 ILP Serialization

In this section details about conversion of linguistic trees to ILP background knowledge (a Prolog logical database of facts) will be presented. Although the construction is quite strait forward it is worth describing because it makes it easier to understand the extraction rules found by the ILP learning procedure.

As mentioned in Section 6.2.2: three main constructs are used to capture an arbitrary configuration of a dependency linguistic tree: nodes, edges and node attributes. During the process of ILP Serialization these constructs are rendered to Prolog in following way.

A unique identifier (node ID) is generated for every tree node. The identifier is based on document name and GATE annotation ID.[8] These node IDs correspond to simple Prolog atoms and they represent tree nodes in the Prolog fact database. A node type (used by the ILP learning algorithm) is assigned to a node ID by predicates `Token(NodeID)` for analytical tree nodes and `tToken(NodeID)` for tectogrammatical tree nodes.

Tree nodes are connected by edges using several binary predicates with a common form:

```
dependency_type_name(ParentNodeID, ChildNodeID)
```

Note that the parent (governor) node always occupies the first argument and the child (dependant) node the second one. Predicate name *tDependency* is used for tectogrammatical dependencies and *aDependency* for analytical ones. There are also special kinds of dependencies that connect tectogrammatical and analytical nodes: *lex.rf* (main lexical reference) and *aux.rf* (auxiliary reference), in these cases tectogrammatical node occupies the first argument and analytical the second.

Node attributes are assigned to node IDs by binary predicates of the form:

```
attribute_name(NodeID, AttributeValue)
```

There are about thirty such predicates like *t_lemma* (tectogrammatical lemma), *functor* (tectogrammatical functor), *sempos* (semantic part of speech), *negation*, *gender*, etc. but only some of them can be found in example extraction rules and we also excluded some of the attributes from serialization examples for space and simplicity reasons.

Example of a serialized tectogrammatical tree is in Figure 6.9 it is the same tree as in Figure 1.3.

Attachment of Overlapping GATE Annotations

During the development of the method, it turned out that it would be very useful to attach additional information, represented by overlapping GATE annotations, to tree nodes. This allows combination of information provided by linguistic trees with information provided by other GATE tools like gazetteers, etc.

Technically, this is realized using binary predicates with a common form:

```
overlap_OverlappingType_OverlappedType(
                            OverlappingAnnotationID, OverlappedNodeID)
```

An overlap with named entities discovers by gazetteers is the most common form of it. For tectogrammatical nodes, it has the form of `overlap_Lookup_tToken(id1,id2)`. An example can be found in Figure 8.1 on line 34.

[8] Note that node IDs based on sentence order and node deep order are used outside of GATE, see PML→RDF transformation in Section 6.3.2.

```
tToken(  id_jihomoravsky47443_243).
t_lemma( id_jihomoravsky47443_243, 'být'). %to be
functor( id_jihomoravsky47443_243, 'PRED').
sempos(  id_jihomoravsky47443_243, 'v').
tDependency( id_jihomoravsky47443_243, id_jihomoravsky47443_238).
tToken(  id_jihomoravsky47443_238).
t_lemma( id_jihomoravsky47443_238, ','). %comma
functor( id_jihomoravsky47443_238, 'APPS').
sempos(  id_jihomoravsky47443_238, 'n.denot').
gender(  id_jihomoravsky47443_238, 'nr').
tDependency( id_jihomoravsky47443_238, id_jihomoravsky47443_237).
tToken(  id_jihomoravsky47443_237).
t_lemma( id_jihomoravsky47443_237, 'vyčíslit'). %to quantify
functor( id_jihomoravsky47443_237, 'PAT').
sempos(  id_jihomoravsky47443_237, 'v').
tDependency( id_jihomoravsky47443_237, id_jihomoravsky47443_245).
tToken(  id_jihomoravsky47443_245).
t_lemma( id_jihomoravsky47443_245, 'předběžně'). %preliminarily
functor( id_jihomoravsky47443_245, 'MANN').
sempos(  id_jihomoravsky47443_245, 'adv.denot.grad.nneg').
tDependency( id_jihomoravsky47443_237, id_jihomoravsky47443_244).
tToken(  id_jihomoravsky47443_244).
t_lemma( id_jihomoravsky47443_244, 'vyšetřovatel'). %investigator
functor( id_jihomoravsky47443_244, 'ACT').
sempos(  id_jihomoravsky47443_244, 'n.denot').
gender(  id_jihomoravsky47443_244, 'anim').
tDependency( id_jihomoravsky47443_237, id_jihomoravsky47443_240).
tToken(  id_jihomoravsky47443_240).
t_lemma( id_jihomoravsky47443_240, 'osm'). %eight
functor( id_jihomoravsky47443_240, 'PAT').
sempos(  id_jihomoravsky47443_240, 'n.quant.def').
gender(  id_jihomoravsky47443_240, 'nr').
tDependency( id_jihomoravsky47443_240, id_jihomoravsky47443_242).
tToken(  id_jihomoravsky47443_242).
t_lemma( id_jihomoravsky47443_242, 'tisíc'). %thousand
functor( id_jihomoravsky47443_242, 'RSTR').
sempos(  id_jihomoravsky47443_242, 'n.quant.def').
gender(  id_jihomoravsky47443_242, 'inan').
tDependency( id_jihomoravsky47443_242, id_jihomoravsky47443_247).
tToken(  id_jihomoravsky47443_247).
t_lemma( id_jihomoravsky47443_247, 'koruna'). %crown
functor( id_jihomoravsky47443_247, 'MAT').
sempos(  id_jihomoravsky47443_247, 'n.denot').
gender(  id_jihomoravsky47443_247, 'fem').
tDependency( id_jihomoravsky47443_237, id_jihomoravsky47443_246).
tToken(  id_jihomoravsky47443_246).
t_lemma( id_jihomoravsky47443_246, 'škoda'). %damage
functor( id_jihomoravsky47443_246, 'PAT').
sempos(  id_jihomoravsky47443_246, 'n.denot').
gender(  id_jihomoravsky47443_246,'fem').
```

Figure 6.9: ILP serialization example based on the tree from Figure 1.3.

6.3 Shareable Extraction Ontologies

In this section we will present details about the implementation of our case study with shareable extraction ontologies. We have used our IE engine based on deep linguistic parsing and Inductive Logic Programming, which was described in Section 5.2. It is a complex system implemented with a great help of the GATE system and it also uses many other third party tools including several linguistic tools and a Prolog system. Installation and making the system operate is not simple. This case study should demonstrate that the extraction rules produced by the system are not dependent on the system in the sense of shareable extraction ontologies.

6.3.1 Linguistic Analysis

The IE engine needs a linguistic preprocessing (deep linguistic parsing) of documents on its input. Deep linguistic parsing brings a very complex structure to the text and the structure serves as a footing for the construction and application of extraction rules. We usually use TectoMT system to do the linguistic preprocessing, see Section 4.2.3 for details. The output linguistic annotations of the TectoMT system are stored (along with the text of the source document) in XML files in so called Prague Markup Language (PML). PML is a very complex language (or XML schema) that is able to express many linguistic elements and features present in text. For the IE engine a tree dependency structure of words in sentences is the most useful one because the edges of the structure guide the extraction rules. Such (tectogrammatical) tree structure was already presented in this thesis, e.g. in Figure 1.3.

6.3.2 Data Transformation (PML to RDF)

In this case study, PML files made by TectoMT from source documents are transformed to RDF document ontologies by quite simple GRDDL/XSLT transformation[9]. Such document ontology contains the whole variety of PML in RDF format. An example of such document ontology can be seen in Figure 6.10. It shows RDF serialization of the tree presented in Figure 1.3 and transformed to ILP background knowledge in Figure 6.9. Transcript of Czech words is provided in comments.

Figure 6.11 shows an example of annotated document ontology, which was created by the application of extraction rules on the document ontology from Figure 6.10. Lines 13-15 and 23-25 are added by reasoner; lines 13-15 and 23 are not very interesting, they show some technical constructions based on the schema mapping, but lines 24 and 25 represent the result of extraction rules, these lines mark the corresponding tree node as "*mention root* of the *damage* task". The same annotated document ontology is shown on Figure 6.12 as a screenshot of the Protégé ontology editor[10].

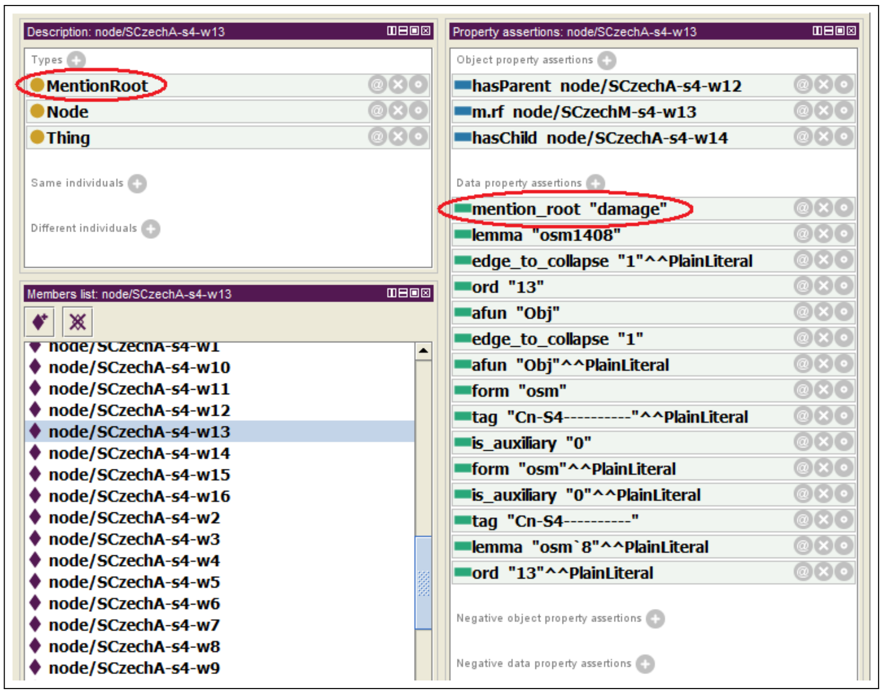
[9]The transformation is available on-line at `http://czsem.berlios.de/ontologies/tmt2rdf.xsl`
[10]`http://protege.stanford.edu/`

```
@prefix node: <http://czsem.berlios.de/ontolog.../jihomoravsky47443.owl#node/> .
@prefix pml: <http://ufal.mff.cuni.cz/pdt/pml/> .

node:SCzechT-s4-n1 rdf:type pml:Node, owl:Thing;
      pml:t_lemma "být" ; #to be
      pml:sempos "v" ; pml:verbmod "ind" ;
      pml:lex.rf node:SCzechA-s4-w4 ; pml:hasParent node:SCzechT-s4-root .
node:SCzechT-s4-n10 rdf:type pml:Node, owl:Thing;
      pml:t_lemma "vyšetřovatel" ; #investigator
      pml:negation "neg0" ; pml:sempos "n.denot" ;
      pml:gender "anim" ; pml:number "sg" ;
      pml:formeme "n:1" ; pml:functor "ACT" ;
      pml:lex.rf node:SCzechA-s4-w9 ; pml:hasParent node:SCzechT-s4-n8 .
node:SCzechT-s4-n11 rdf:type pml:Node, owl:Thing;
      pml:degcmp "pos" ; pml:t_lemma "předběžně" ; #preliminarily
      pml:formeme "adv:" ; pml:sempos "adv.denot.grad.nneg" ;
      pml:functor "MANN" ; pml:negation "neg0" ;
      pml:lex.rf node:SCzechA-s4-w10 ; pml:hasParent node:SCzechT-s4-n8 .
node:SCzechT-s4-n12 rdf:type pml:Node, owl:Thing;
      pml:t_lemma "osm" ; #eight
      pml:number "pl" ; pml:numertype "basic" ;
      pml:sempos "n.quant.def" ; pml:formeme "n:???" ;
      pml:functor "PAT" ; pml:gender "nr" ;
      pml:lex.rf node:SCzechA-s4-w13 ; pml:hasParent node:SCzechT-s4-n8 .
node:SCzechT-s4-n13 rdf:type pml:Node, owl:Thing;
      pml:t_lemma "tisíc" ; #thousand
      pml:number "sg" ; pml:functor "RSTR" ;
      pml:gender "inan" ; pml:sempos "n.quant.def" ;
      pml:numertype "basic" ; pml:formeme "n:???" ;
      pml:lex.rf node:SCzechA-s4-w14 ; pml:hasParent node:SCzechT-s4-n12 .
node:SCzechT-s4-n14 rdf:type pml:Node, owl:Thing;
      pml:t_lemma "koruna" ; #crown
      pml:gender "fem" ; pml:sempos "n.denot" ;
      pml:number "pl" ; pml:formeme "n:2" ;
      pml:functor "MAT" ; pml:negation "neg0" ;
      pml:lex.rf node:SCzechA-s4-w15 ; pml:hasParent node:SCzechT-s4-n13 .
node:SCzechT-s4-n7 rdf:type pml:Node, owl:Thing;
      pml:t_lemma "," ; #comma
      pml:gender "nr" ; pml:negation "neg0" ;
      pml:sempos "n.denot" ; pml:functor "APPS" ;
      pml:formeme "n:???" ; pml:number "nr" ;
      pml:lex.rf node:SCzechA-s4-w7 ; pml:hasParent node:SCzechT-s4-n1 .
node:SCzechT-s4-n8 rdf:type pml:Node, owl:Thing;
      pml:t_lemma "vyčíslit" ; #quantify
      pml:is_member "1" ; pml:deontmod "decl" ; pml:formeme "v:fin" ;
      pml:tense "ant" ; pml:verbmod "ind" ; pml:aspect "cpl" ;
      pml:is_clause_head "1" ; pml:functor "PAR" ;
      pml:dispmod "disp0" ; pml:sempos "v" ; pml:negation "neg0" ;
      pml:lex.rf node:SCzechA-s4-w11 ; pml:hasParent node:SCzechT-s4-n7 .
node:SCzechT-s4-n9 rdf:type pml:Node, owl:Thing;
      pml:t_lemma "škoda" ; #damage
      pml:sempos "n.denot" ; pml:functor "PAT" ; pml:gender "fem" ;
      pml:formeme "n:4" ; pml:number "sg" ; pml:negation "neg0" ;
      pml:lex.rf node:SCzechA-s4-w8 ; pml:hasParent node:SCzechT-s4-n8 .
```

Figure 6.10: Document ontology (RDF serialization of linguistic trees), based on Figure 1.3
.

```
@prefix node: <http://czsem.berlios.de/ontologies/.../jihomoravsky47443.owl#node/> .
@prefix pml: <http://ufal.mff.cuni.cz/pdt/pml/> .
# Mapping Ontology
@prefix PML2GATE: <http://czsem.berlios.de/ontologies/PML2GATE_ontology_utils.owl#> .

node:SCzechT-s4-n12 rdf:type pml:Node, owl:Thing;
       pml:t_lemma "osm" ; #eight
       pml:number "pl" ; pml:numertype "basic" ;
       pml:sempos "n.quant.def" ; pml:formeme "n:???" ;
       pml:functor "PAT" ; pml:gender "nr" ;
       pml:lex.rf node:SCzechA-s4-w13 ; pml:hasParent node:SCzechT-s4-n8 ;
################### Added by Reasoner ############################################
       rdf:type PML2GATE:TNode ; #Tectogrammatical node
       PML2GATE:hasChild node:SCzechT-s4-n13 ;
       pml:tDependency node:SCzechT-s4-n13 .
#################################################################################
node:SCzechA-s4-w13 rdf:type pml:Node, owl:Thing;
       pml:is_auxiliary "0" ; pml:edge_to_collapse "1" ;
       pml:ord "13" ; pml:tag "Cn-S4----------" ;
       pml:afun "Obj" ; pml:form "osm" ; pml:lemma "osm\1408" ;
       pml:m.rf node:SCzechM-s4-w13 ; pml:hasParent node:SCzechA-s4-w12 ;
################### Added by Reasoner ############################################
       PML2GATE:hasChild node:SCzechA-s4-w14 ;
       pml:mention_root "damage" ;
       rdf:type PML2GATE:MentionRoot .
#################################################################################
```

Figure 6.11: Annotated document ontology example


Description: node/SCzechA-s4-w13

Types
- MentionRoot
- Node
- Thing

Same individuals

Different individuals

Members list: node/SCzechA-s4-w13
- node/SCzechA-s4-w1
- node/SCzechA-s4-w10
- node/SCzechA-s4-w11
- node/SCzechA-s4-w12
- node/SCzechA-s4-w13
- node/SCzechA-s4-w14
- node/SCzechA-s4-w15
- node/SCzechA-s4-w16
- node/SCzechA-s4-w2
- node/SCzechA-s4-w3
- node/SCzechA-s4-w4
- node/SCzechA-s4-w5
- node/SCzechA-s4-w6
- node/SCzechA-s4-w7
- node/SCzechA-s4-w8
- node/SCzechA-s4-w9

Property assertions: node/SCzechA-s4-w13

Object property assertions
- hasParent node/SCzechA-s4-w12
- m.rf node/SCzechM-s4-w13
- hasChild node/SCzechA-s4-w14

Data property assertions
- mention_root "damage"
- lemma "osm1408"
- edge_to_collapse "1"^^PlainLiteral
- ord "13"
- afun "Obj"
- edge_to_collapse "1"
- afun "Obj"^^PlainLiteral
- form "osm"
- tag "Cn-S4----------"^^PlainLiteral
- is_auxiliary "0"
- form "osm"^^PlainLiteral
- is_auxiliary "0"^^PlainLiteral
- tag "Cn-S4----------"
- lemma "osm`8"^^PlainLiteral
- ord "13"^^PlainLiteral

Negative object property assertions

Negative data property assertions

Figure 6.12: Annotated document ontology in Protégé ontology editor

6.3.3 Rule Transformations

```
%[Rule 1] [Pos cover = 14 Neg cover = 0]
mention_root(damage,A) :-
   'lex.rf'(B,A), sempos(B,'n.quant.def'), tDependency(C,B), tDependency(C,D),
   t_lemma(D,'vyšetřovatel'). %investigator

%[Rule 2] [Pos cover = 13 Neg cover = 0]
mention_root(damage,A) :-
   'lex.rf'(B,A), functor(B,'TOWH'), tDependency(C,B), tDependency(C,D),
   t_lemma(D,'škoda'). %damage
```

Figure 6.13: Examples of extraction rules in the native Prolog format.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Ontology [
  <!ENTITY pml "http://ufal.mff.cuni.cz/pdt/pml/" >
]>
<Ontology xmlns="http://www.w3.org/2002/07/owl#"
  ontologyIRI="http://czsem.berlios.de/ontologies/...rules.owl">
  <DLSafeRule>
    <Body>
      <ObjectPropertyAtom> <ObjectProperty IRI="&pml;lex.rf"/>
        <Variable IRI="urn:swrl#b"/> <Variable IRI="urn:swrl#a"/>
      </ObjectPropertyAtom>
      <DataPropertyAtom> <DataProperty IRI="&pml;sempos"/>
        <Variable IRI="urn:swrl#b"/> <Literal>n.quant.def</Literal>
      </DataPropertyAtom>
      <ObjectPropertyAtom> <ObjectProperty IRI="&pml;tDependency"/>
        <Variable IRI="urn:swrl#c"/> <Variable IRI="urn:swrl#b"/>
      </ObjectPropertyAtom>
      <ObjectPropertyAtom> <ObjectProperty IRI="&pml;tDependency"/>
        <Variable IRI="urn:swrl#c"/> <Variable IRI="urn:swrl#d"/>
      </ObjectPropertyAtom>
      <DataPropertyAtom> <DataProperty IRI="&pml;t_lemma"/>
        <Variable IRI="urn:swrl#d"/> <Literal>vyšetřovatel</Literal>
      </DataPropertyAtom>
    </Body>
    <Head>
      <DataPropertyAtom> <DataProperty IRI="&pml;mention_root" />
        <Literal>damage</Literal> <Variable IRI="urn:swrl#a" />
      </DataPropertyAtom>
    </Head>
  </DLSafeRule>
</Ontology>
```

Figure 6.14: Rule 1 in the OWL/XML syntax for Rules in OWL 2 [Glimm *et al.*, 2009].

Extraction rules produced by the IE engine are natively kept in a Prolog format; examples can be seen in Figure 6.13. The engine is capable to export them to the OWL/XML[11] syntax for rules in OWL 2 [Glimm *et al.*, 2009] (see examples in Figure 6.14). The complete Prolog procedure performing the export is listed in the appendix, Section A.2.

The OWL/XML rules can be parsed by OWL API[12] 3.1 and exported to RDF/SWRL, which is very widely supported and hopefully becoming a W3C recommendation. We do not provide examples of rules in this format because it occupies lots of space and it is very difficult to follow.

OWL/XML rules can be also parsed by the Protégé ontology editor and Figure 6.15 shows the example rules in Protégé 4 – Rules View's format.

The last rule example can be seen in Figure 6.16, it shows a rule in the Jena rules format[13]. Conversion to Jena rules was necessary because it is the only format that Jena can parse, see details about our usage of Jena in Section 8.3. The Jena rules were obtained using following transformation process:

1. OWL/XML → RDF/SWRL conversion using OWL API and
2. RDF/SWRL → Jena rules conversion using SweetRules[14].

The presented rules belong to the group of so called DL-Safe rules [Motik *et al.*, 2005] so the decidability of OWL reasoning is kept.

[11]`http://www.w3.org/TR/owl-xmlsyntax/`
[12]`http://owlapi.sourceforge.net/`
[13]`http://jena.sourceforge.net/inference/#RULEsyntax`
[14]`http://sweetrules.semwebcentral.org/`

```
#[Rule 1]
lex.rf(?b, ?a), sempos(?b, "n.quant.def"), tDependency(?c, ?b),
tDependency(?c, ?d), t_lemma(?d, "vyšetřovatel") #investigator
    -> mention_root(?a, "damage")

#[Rule 2]
lex.rf(?b, ?a), functor(?b, "TOWH"), tDependency(?c, ?b),
tDependency(?c, ?d), t_lemma(?d, "škoda") #damage
    -> mention_root(?a, "damage")
```

Figure 6.15: Examples of extraction rules in Protégé 4 – Rules View's format.

```
@prefix pml: <http://ufal.mff.cuni.cz/pdt/pml/>.
[rule-75:
        ( ?b pml:lex.rf ?a )
        ( ?b pml:sempos 'n.quant.def' )
        ( ?c pml:tDependency ?b )
        ( ?c pml:tDependency ?d )
        ( ?d pml:t_lemma 'vyšetřovatel' )
     ->
        ( ?a pml:mention_root 'damage' )
]
```

Figure 6.16: Rule 1 in the Jena rules syntax.

6.4 Fuzzy ILP Classification

In the experimental system we use two inductive logic approaches: crisp and fuzzy (as described above). Technically, the difference between the approaches consists in a different setting of the underlying *ILP task*. Both can be done with a classical ILP tool (Aleph (see Section 4.7.3) was used in the final implementation).

We have compared results of the crisp and fuzzy approaches with other classification methods and, in our experiment, the fuzzy approach produced better results than many other methods, including the crisp one. See Section 6.4.1 for details.

To use ILP for a classification task we have to translate the input data to the Prolog-like logic representation, as it was already described in previous sections. Here we will describe implementation details of constructing crisp and fuzzy knowledge bases and example sets.

In construction of a crisp example set E_t, the target predicate is denoted as `serious_t`. The letter `t` stands for the actual seriousness degree. We use multiple unary predicates `serious_0`, `serious_1`, etc., instead of one binary predicate `serious(ID,Degree)`. These two cases are equivalent and we have decided to use the unary option because a visual distinction between the multiple ILP tasks is then clearer.

In construction of a fuzzy (or monotonized) example set $E_{\geq t}$, the target predicate is denoted as `serious_atl_t`, see examples in Figure 6.17.

In construction of a crisp background knowledge B_T^{crisp}, we use a simple translation of the attribute names to the names of predicates and fill them with actual values. It is illustrated in Figure 6.18).

In construction of a monotonized background knowledge B_T^{monot} we reuse the crisp background knowledge and add monotonization rules. An example for predicate `damage` is shown in Figure 6.19. The first rule deals with `unknown` values (Section 7.3.6 deals with unknown values in the dataset) and the second does the monotonization.

Negations used in Figure 6.19 and Figure 6.21 are the standard Prolog *negations as failure*.

Once we have learning examples and background knowledge, we can run the ILP inductive procedure and obtain learned rules (a learned hypothesis). According to the kind of the ILP task (crisp or monotonized), we obtain the corresponding kind (crisp or mono-

```
% Crisp Learning examples E_t

serious_2(id_47443). %positive

serious_0(id_47443). %negative
serious_1(id_47443). %negative
serious_3(id_47443). %negative

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Monotonized Learning examples E_≥t

serious_atl_0(id_47443). %positive
serious_atl_1(id_47443). %positive
serious_atl_2(id_47443). %positive

serious_atl_3(id_47443). %negative
```

Figure 6.17: Learning examples.

```
size(id_47443, 427).
type(id_47443, fire).
damage(id_47443, 8000).
dur_minutes(id_47443, 50).
fatalities(id_47443, 0).
injuries(id_47443, 0).
cars(id_47443, 0).
amateur_units(id_47443, 3).
professional_units(id_47443, 1).
pipes(id_47443, unknown).
lather(id_47443, 0).
aqualung(id_47443, 0).
fan(id_47443, 0).
```

Figure 6.18: B_T^{crisp} – crisp attributes.

```
damage_atl(ID,N) :- damage(ID,N), not(integer(N)). %unknown values

damage_atl(ID,N) :- damage(ID,N2), integer(N2), %numeric values
                    damage(N), integer(N), N2>=N.
```

Figure 6.19: Monotonization of attributes (damage_atl ← damage).

```
serious_0(A) :- fatalities(A,0), injuries(A,0), cars(A,1), amateur_units(A,0), lather(A,0).
serious_0(A) :- fatalities(A,0), cars(A,0), amateur_units(A,0), professional_units(A,1).
serious_1(A) :- amateur_units(A,1).
serious_1(A) :- damage(A,300000).
serious_1(A) :- type(A,fire), amateur_units(A,0), pipes(A,2).
serious_1(A) :- type(A,car_accident),dur_minutes(A,unknown),fatalities(A,0),injuries(A,1).
serious_2(A) :- lather(A,unknown).
serious_2(A) :- cars(A,0), lather(A,0), aqualung(A,1), fan(A,0).
serious_2(A) :- amateur_units(A,2).
serious_3(A) :- fatalities(A,2).
serious_3(A) :- type(A,fire), dur_minutes(A,unknown), cars(A,0), fan(A,0).
serious_3(A) :- injuries(A,2), cars(A,2).
serious_3(A) :- fatalities(A,1).

serious_atl_0(A).
serious_atl_1(A) :- injuries_atl(A,1).
serious_atl_1(A) :- dur_minutes_atl(A,21), pipes_atl(A,1), aqualung_atl(A,0).
serious_atl_1(A) :- damage_atl(A,8000), amateur_units_atl(A,3).
serious_atl_1(A) :- dur_minutes_atl(A,197).
serious_atl_1(A) :- dur_minutes_atl(A,unknown).
serious_atl_2(A) :- dur_minutes_atl(A,50), pipes_atl(A,3).
serious_atl_2(A) :- size_atl(A,1364), injuries_atl(A,1).
serious_atl_2(A) :- fatalities_atl(A,1).
serious_atl_2(A) :- size_atl(A,1106), professional_units_atl(A,3).
serious_atl_3(A) :- fatalities_atl(A,1).
serious_atl_3(A) :- damage_atl(A,1500000).
```

Figure 6.20: Crisp and monotonized hypotheses.

```
serious_0(ID) :- serious_atl_0(ID),
                 not(serious_atl_1(ID)), not(serious_atl_2(ID)), not(serious_atl_3(ID)).
serious_1(ID) :- serious_atl_1(ID),
                 not(serious_atl_2(ID)), not(serious_atl_3(ID)).
serious_2(ID) :- serious_atl_2(ID),
                 not(serious_atl_3(ID)).
serious_3(ID) :- serious_atl_3(ID).
```

Figure 6.21: Conversion rules for monotonized hypotheses (serious_t ← serious_atl_t).

tonized) of rules (see e.g. in Figure 6.20). But these rules cannot be used directly to solve the classification task. There are common cases when more than one rule is applicable to a single instance. So we have to select which one to use. For the monotonized hypothesis we select the one with the biggest return value; it is illustrated in Figure 6.21. Such clear criterion does not exist for the crisp hypothesis, so we simply use the first applicable rule.

In the crisp case there are often many instances which cannot be classified because there is no applicable rule. In our experiment there was about a 51% of unclassified instances (see in the next section). It could be caused by the lack of training data, but the monotonized approach does not suffer from this shortage. We can always select the bottom value.

Another advantage of the monotonized approach is that the set of positive training examples is extended by monotonization.

6.4.1 Learned Rules Examples

Figure 6.20 summarizes obtained hypotheses learned from our data:

- a crisp hypothesis learned from E_t and B_T^{crisp} (lines 1-13) and
- a monotonized hypothesis learned from $E_{\geq t}$ and B_T^{monot} (lines 15-26).

In both cases, learning examples and background knowledge have the origin in the same data (the same accidents). The hypotheses differ in the form of the ILP task (crisp and monotonized). The crisp hypothesis uses only the crisp predicates, and the monotonized hypothesis uses only the monotonized predicates.

7. Datasets

In this chapter several datasets will be described. The datasets can be divided into several groups according to several criterions:

- structure and purpose of a particular dataset (textual for IE tasks, relational for classification ML tasks and RDF datasets for Semantic Web reasoning tasks),
- presence of manual annotations,
- language in case of textual data (English and Czech),
- the origin of the dataset (third party datasets and datasets created as a part of the presented work).

The description of individual datasets will be presented at the very end of the chapter, before that, relevant common aspects of the datasets will be discussed.

7.1 Purpose and Structure

In this section a distinction of datasets according their purpose and structure will be presented. Three types of datasets will be discussed: Information Extraction Datasets, Classification Datasets and Reasoning Datasets.

7.1.1 Information Extraction Datasets

An IE dataset is made up of a set of text documents. The texts are usually manually annotated (with one exception see Section 7.3.1). The manual annotations are of the form of labels on shorter segments of the text (the length of a segment ranges between one to approximately three tokens).

An IE engine can use such dataset for training and evaluation. During evaluation, the dataset is split into two parts – training set and testing set. An IE engine is trained on the training set and success is measured on testing set by comparing the annotations returned by the IE engine and the actual annotations present in the dataset. Performance measures of precision and recall are mostly used for evaluation.

Following datasets can be regarded as IE datasets:

- Czech Fireman Reports without Annotations (Section 7.3.1)
- Czech Fireman Reports Manually Annotated (Section 7.3.2)
- Corporate Acquisition Events (Section 7.3.3)

Both the manually annotated datasets (Czech Fireman Reports Manually Annotated and Corporate Acquisition Events) are of the kind of *event extraction encoded as entity recognition* described in Section 2.1.7.

7.1.2 Reasoning Datasets

The term "Reasoning Datasets" is used here for the type of datasets which are used in Ontology Benchmarks. Ontology benchmarking is a scientific topic that is almost as old as ontologies and Semantic Web itself [Guo *et al.*, 2003]. Ontology Benchmarks serve for evaluation of ontology systems and their capabilities. There are two obvious objectives of ontology benchmarking. The first one is to verify capabilities of a tested system – weather the system is able to perform all actions prescribed by the particular benchmark. The second objective is to measure the time performance of the system. Unlike the previous two dataset types in the case of ontology benchmarking and reasoning datasets it does not make any sense to measure success of a system because there is no uncertainty in reasoning tasks. A system is either able to perform a particular action or not; it is unnecessary to measure that.

Reasoning datasets used in the present work:

- RDF Dataset Based on Czech Fireman Reports (Section 7.3.4)
- RDF Dataset Based on Corporate Acquisition Events (Section 7.3.5)

These datasets were used for evaluation of the idea of shareable extraction ontologies, see Section 8.3.

7.1.3 Classification Datasets

Classification datasets are very familiar in the community of machine learning. They are used for evaluation of propositional ML engines (like Decision Trees, Naive Bayes, Multilayer Perceptron, Support Vector Machines, etc.) on the classification task. The classification task (or problem) can be simplified way described as follows:

Given a set of objects (e.g. accidents); each object is described using a fixed set of attributes (e.g. number of fatalities, number of injuries, number of intervening units; etc.) and each object is classified into one of a fixed set of classes (not serious accident, middle serious accident, very serious accident). Based on the known classification of the objects (training set), predict the classification for new objects that have been not classified yet (testing set).

A classification dataset is made up of a relational table. Each row of the table represents a single object; each column of the table represents a single attribute. One attribute (column) is marked as a class attribute and the values of this attribute determine the target classification.

Similarly to the previous section, the dataset is split into two parts during the evaluation. A ML engine is trained on the training set and success is measured on testing set by comparing the predicted classification of the ML engine and the actual classification present in the dataset.

Classification datasets used in the present work:

- Classification Dataset Based on Czech Fireman Reports (Section 7.3.6)
- Classification Datasets from UCI ML Repository (Section 7.3.7)

These datasets were used for evaluation of the Fuzzy ILP classification method in Section 8.4.

7.2 Origin of the Datasets

The datasets can be simply divided into third party and contributed.

7.2.1 Contributed Datasets

Datasets that were partly or mostly prepared as a part the present work are:

- Czech Fireman Reports without Annotations (Section 7.3.1)
- Czech Fireman Reports Manually Annotated (Section 7.3.2)
- RDF Dataset Based on Czech Fireman Reports (Section 7.3.4)
- RDF Dataset Based on Corporate Acquisition Events (Section 7.3.5)
- Classification Dataset Based on Czech Fireman Reports (Section 7.3.6)

All the contributed datasets can be downloaded from the web site of the project; see details in Section 9.6.

7.2.2 Third Party Datasets

Datasets that were used without any additional contributions are:

- Corporate Acquisition Events (Section 7.3.3)
- Classification Datasets from UCI ML Repository (Section 7.3.7)

7.3 Individual Datasets

In this section individual datasets used in this thesis will be presented. The order of dataset presentations is the same as the order in which they were used in our work.

7.3.1 Czech Fireman Reports without Annotations

In the early beginning of our work, first IE experiments were done with textual reports from fire departments of several regions of the Czech Republic. These departments are responsible for rescue and recovery after fire, traffic and other accidents. Likewise the reports do not deal only with fire and traffic accidents but also chemical interventions, fire-fighting contests, fire drills and similar events can be found. The reports are rich in information, e.g. where and when an 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 5.2.

The dataset is made up by 814 texts of the reports collected in the time period from February to September 2007 using a RSS feed. All the reports are still available on the web site of the Ministry of Interior of the Czech Republic[1].

Following URL pattern can be used to obtain a particular report:

```
http://aplikace.mvcr.cz/archiv2008/rs_atlantic/hasici/REGION/ID.html
```

For example report 47443 (ID) from Jihomoravský region has following URL:

```
http://aplikace.mvcr.cz/archiv2008/rs_atlantic/hasici/jihomoravsky/47443.html
```

[1]`http://www.hzscr.cz/hasicien/`

	sum	avg	st. dev.	min	max	median
text size	1 298 112	1 594.7	1 142.34	57	11 438	1 290.5
num. words	195 168	239.8	172.02	9	1 750	193.0
annot. size	51 464 581	63 224.3	43 557.61	4 648	451 953	52 033.5
num. t-trees	15 208	18.7	15.02	1	143	14.0

Table 7.1: Dataset 7.3.1 – Czech Fireman Reports without Annotations – statistics

The dataset contains reports from eight Czech regions: Jihomoravský (127 reports), Královéhradecký (109 reports), Moravskoslezský (138), Olomoucký (77), Pardubický (95), Plzeňský (97), Ústecký (108) and Vysočina (63). All the reports are written in Czech language.

The dataset does not contain any manual annotations; only linguistic annotations produced by PDT tools are included in the dataset (FS format for Netgraph processing and PML format for Btred processing, see Section 4.2 for details about these tools). Table 7.1 summarizes some properties of the dataset's reports described bellow. Total sum, average value, standard deviation, minimum, maximum and median of the values can be found in the table.

Text size is the length of the text of a particular report in characters. More precisely it is the size of the corresponding ".txt" file. ISO-8859-2 (Latin-2) character encoding is used in these files so the number of characters is the same as the file size in bytes.

Number of words in a particular report is counted using UNIX command "wc -w".

Annotations size expresses the number of bytes used by the corresponding linguistic annotations of a particular report. Only annotations in FS format are counted (PML files are not included in these numbers).

Number of trees is the number of tectogrammatical trees in a particular report. The number also corresponds with the number of sentences in the report.

The dataset was used for a quantitative evaluation experiment described in Section 8.1.1.

7.3.2 Czech Fireman Reports Manually Annotated

We have created a manually annotated version of the previous dataset. This dataset was created by ourselves during the development of our IE engine based on machine learning (Section 5.2). 50 reports from the previous dataset were selected and manually annotated. Several annotation types were used; numbers of annotations per each annotation type are summarized in Table 7.2 along with the information whether the particular annotation type was used in the evaluation. See details abut the evaluation results in Section 8.2. This dataset is of the kind of *event extraction encoded as entity recognition* described in Section 2.1.7. Individual annotation types will be briefly described in the rest of this section.

'professional_unit' annotates mentions of professional fireman units that intervened in the particular accident.

'cars' annotates mentions of cars damaged during the particular accident.

'end' annotates mentions of the time when the whole operation of fireman units finished.

'end_subtree' was obtained by the subtree postprocessing of 'end' annotations. This was necessary because 'end' annotations were not annotated consistently – sometimes they included only the numeric representation of the time, sometimes they included also the unit; e.g. ‘16:10’ versus ‘16:10 hours’. ’end_subtree’ always include the time unit (if available in text).

‘start’ annotates mentions of the time when the accident started or was reported to firemen.

‘injuries’ annotates mentions of people being injured during the accident.

‘amateur_unit’ annotates mentions of voluntary fireman units that intervened in the particular accident.

‘damage’ annotates mentions of money amounts representing the total damage sought by the accident.

‘pipes’ annotates numbers of fireman pipes or fire hoses used during the accident.

‘professional_units’ annotates numbers of professional fireman units intervening during the accident.

‘fatalities’ annotates mentions of people who died during or after the accident.

‘units’ annotates numbers of all (professional or voluntary) fireman units intervening during the accident.

‘aqualung’ annotates mentions of an aqualung (or breathing apparatus) used during the accident.

‘duration’ annotates mentions of the time duration of the whole action or operation.

‘fan’ annotates mentions of a fan (or ventilator) used during the accident.

‘amateur_units’ annotates numbers of voluntary fireman units intervening during the accident.

‘lather’ annotates mentions of lather (or foam) used during the accident.

7.3.3 Corporate Acquisition Events

The second manually annotated information extraction dataset used in this thesis is called “Corporate Acquisition Events”. It is a collection of 600 news articles describing acquisition events taken from the Reuters dataset [Lewis, 1992]. We use the *Acquisitions v1.1* version[2] of the corpus.

Several types of annotations related to acquisition events are present in these articles. They include ‘purchaser’ , ‘acquired’, and ‘seller’ companies along with their abbreviated names (‘purchabr’, ‘acqabr’ and ‘sellerabr’). Some articles also mention the field ‘deal amount’. Table 7.3 provides numbers of annotations per annotation type for the whole dataset along with the information whether the particular annotation type was used in the evaluation. See details about the evaluation results in Section 8.2.

Also this dataset is of the kind of *event extraction encoded as entity recognition* described in Section 2.1.7.

[2]This version of the corpus comes from the Dot.kom (Designing infOrmation extracTion for KnOwledge

annotation type	number of annotations	used in evaluation
professional_unit	78	yes
cars	45	yes
end	42	no, end_subtree only
end_subtree	42	yes
start	42	yes
injuries	33	yes
amateur_unit	31	yes
damage	20	yes
pipes	16	no
professional_units	14	no
fatalities	11	yes
units	10	no
aqualung	8	no
duration	3	no
fan	3	no
amateur_units	1	no
lather	1	no

Table 7.2: Dataset 7.3.2 – Czech Fireman Reports Manually Annotated – manual annotations

annotation type	number of annotations	used in evaluation
acqabr	1450	yes
acqbus	264	no
acqcode	214	no
acqloc	213	no
acquired	683	yes
dlramt	283	yes
purchabr	1263	yes
purchaser	624	yes
purchcode	279	no
seller	267	yes
sellerabr	431	yes
sellercode	136	no
status	461	no

Table 7.3: Dataset 7.3.3 – Corporate Acquisition Events – manual annotations

7.3.4 RDF Dataset Based on Czech Fireman Reports

Both reasoning datasets presented in this work are based on information extraction tasks. The datasets were created for evaluation of the idea of shareable extraction ontologies (see Section 5.3). The datasets consists of so called document ontologies – RDF representation of source documents (see Section 5.3.1) and extraction ontologies – RDF representation of extraction rules (see Section 5.3). Besides that there are also small mapping ontologies that solve small differences of the schemas used in document and extraction ontologies.

The reasoning task is to combine the three kinds of ontologies and infer all instances that should be found by the extraction rules saved in the particular extraction ontology, for details see the Section 8.3.

This dataset is based on the IE dataset Czech Fireman Reports Manually Annotated (Section 7.3.2); it consists of 50 document ontologies, one mapping ontology and currently one extraction ontology, which contains extraction rules for the 'damage' task – to find an amount (in CZK - Czech Crowns) of summarized damage arisen during a reported accident.

Section 8.3.1 provides also some additional information (basic statistics) about the dataset.

7.3.5 RDF Dataset Based on Corporate Acquisition Events

This is the second reasoning dataset presented in this work. The dataset is very similar to the previous one (see details in the previous section). It is based on the information extraction dataset Corporate Acquisition Events (Section 7.3.3). It consists of 600 document ontologies, one mapping ontology and currently one extraction ontology, which contains extraction rules for the 'acquired' task.

Section 8.3.1 provides also some additional information (basic statistics) about the dataset.

7.3.6 Classification Dataset Based on Czech Fireman Reports

The development of the Fuzzy ILP Classifier was partly motivated by the possibility of doing seriousness classification of accidents presented in the reports described in previous sections. We selected the same collection of 50 web reports as in the previous section and we manually extracted values of the most interesting features (These features will be described in Section 7.3.6.) We also assigned a value of overall ranking of seriousness of the presented accident to each report. The value of seriousness ranking is the target attribute of the classification task.

Features of accidents

Table 7.4 summarizes all features (or attributes) that were obtained from accident reports. Except for the attribute `type` (type of an accident – `fire`, `car_accident` and `other`), all the attributes are numeric and therefore *monotonizable*[3]. There are cases in which the value of an attribute is unknown. We decided to make evidence of this and keep the values `unknown` in the knowledge base. A brief explanation of each attribute follows.

- `size` is length of text of a particular report.

Management) project's resources: `http://nlp.shef.ac.uk/dot.kom/resources.html`

[3]This is important, because only monotonizable attributes can be translated to fuzzy degrees; see details in Section 5.4.3.

attribute name	distinct values	missing values	monotonic
size (of file)	49	0	yes
type (of accident)	3	0	no
damage	18	30	yes
dur_minutes	30	17	yes
fatalities	4	0	yes
injuries	5	0	yes
cars	5	0	yes
amateur_units	7	1	yes
professional_units	6	1	yes
pipes	7	8	yes
lather	3	2	yes
aqualung	3	3	yes
fan	3	2	yes
ranking	14	0	yes

Table 7.4: Accident attributes.

- `damage` is an amount (in CZK – Czech Crowns) of summarized damage arisen during a reported accident.
- `dur_minutes` is time taken to handle an accident.
- `fatalities` and `injuries` are numbers of deaths and wound people sustained in an accident.
- `cars` is the number of vehicles damaged during an accident (mostly during car accidents).
- `professional_units` and `amateur_units` are numbers of fireman and volunteer units sent for a particular accident.
- `pipes` is a number of used fire hoses.
- `lather`, `aqualung` and `fan` (ventilator) indicates whether these devices were used.

A majority of accidents are of the type `fire` (52%) and `car_accident` (30%), the rest (type `other`, 18%) deals with ecological disasters, chemical accidents, etc.

Seriousness ranking

Values of the overall seriousness ranking attribute were stated from “a general impression” made by the texts with respect to particular attributes. The values have evolved to 14 distinct values in the range from 0.5 to 8. A histogram with frequencies of all these values is in Figure 7.1. We divided the values into four approximately equipotent groups (see in Figure 7.1) and these groups determine the target class attribute of the classification task.

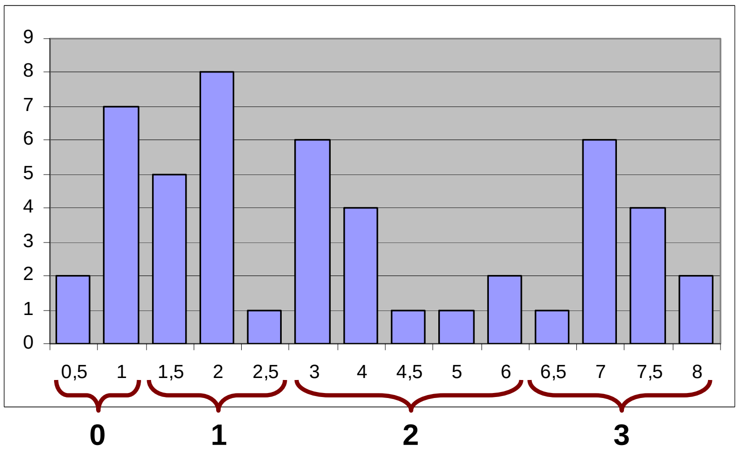


Figure 7.1: Frequencies of the seriousness ranking.

shortcut	name	url suffix[a]
car	Car Evaluation	`Car+Evaluation`
wine[b]	Wine Quality	`Wine+Quality`
cmc	Contraceptive Method Choice	`Contraceptive+Method+Choice`
tae	Teaching Assistant Evaluation	`Teaching+Assistant+Evaluation`
pop	Post-Operative Patient	`Post-Operative+Patient`
nurs	Nursery	`Nursery`

[a] The prefix is: `http://archive.ics.uci.edu/ml/datasets/`
[b] Only red wine part of the dataset was used.

Table 7.5: Classification datasets selected form the UCI ML Repository.

7.3.7 Classification Datasets from UCI ML Repository

UCI Machine Learning Repository[4] [Frank and Asuncion, 2010] is a large archive of machine learning datasets collected at the University of California, Irvine. Several classification datasets were selected and used in the evaluation of the Fuzzy ILP Classifier, see in Section 8.4.2. The list of selected datasets can be found in Table 7.5. All the datasets are monotonizable (the target attribute can be naturally ordered), so the fuzzy classifier could take advantage of that.

[4]`http://archive.ics.uci.edu/ml/index.html`

8. Experiments and Evaluation

Experiments that were performed to evaluate our methods and approaches will be presented in this chapter. Four main sections separate the experiments by topic, to which they belong.

8.1 Evaluation of Manual Rules

In this section, two experiments will be presented to explain the usefulness of our extraction method based on manually designed rules. The first one provides measurements on a higher amount of texts without manual gold standard annotations, while the second experiment was done on a small manually annotated collection.

8.1.1 Czech Fireman Quantitative

We evaluated three extraction rules (one procedural and two Netgraph based) on a set of 814 texts of news of several Czech fire departments and measured several statics. All the rules had the same goal: to find numbers of people that died or were injured during an accident. The procedural rule was the same as in Figure 6.1 and the Netgraph based rules correspond with the rule in Figure 5.3. The only difference between the Netgraph rules is that in the first case (Netgraph 1) all rule nodes except number 1 (`action_type`) were set as optional, while in the second case (Netgraph 2) also node 4 (participant) was compulsory. This little change caused some interesting effects – see below.

Table 8.1 summarizes some of the statics that were measured. Description of individual values follows.

Files The same set of 814 files (texts) was used in all the experiments.

Rule matches The presence of optional nodes in a query increases the number of possibilities how a Netgraph query can be matched to a single linguistic tree. An optional node might or might not be marked in the result. The number of possible matches is also increased if there are more compatible nodes in a candidate tree that can be matched with a single query node. This can even be true for the participant query node (4) in the case of Netgraph based rules if a sentence mentions more than one affected person. This can be marked as a mistake in the rule design – it does not count with such possibility, or it can be taken as a drawback of the current evaluation algorithm of the Netgraph based method – it should put all the possibilities to the output. Note that this issue does not concern the procedural method, which outputs all the matching participants.

Unique matched trees This number represents the number of unique trees matched by the extraction rule.

Effective rule matches Because the procedural rules and the Netgraph based rules are evaluated in a different way, the way of selection of effective matches (matches that are used for the output) is also different. In the procedural case all matches are used because every such match is tied up with a different verb in a sentence. In this case more matches per sentence (tree) are only possible for complex sentences with more verbs and therefore every match is applicable because it is connected with a different piece of information.

		extraction rule		
		procedural	Netgrpah 1	Netgraph 2
	files	814	814	814
	rule matches	472	1594	798
	unique matched trees	455	455	376
	efective rule matches	472	455	376
negation	TRUE	182	180	160
	FALSE	290	275	216
	participant	412	376	376
participant quantity	count	196	176	175
	avg	30.2	162.2	4.3
	median	2	2	2
	max	4988	27800	333
action type	přežít (to survive)	24	24	4
	usmrtit (to kill)	20	18	7
	zahynout (to perish)	8	8	6
	zemřít (to die)	24	23	14
	zranit (to injure)	396	382	345

Table 8.1: Evaluation of manually created rules (bigger dataset without manual annotations).

In the Netgraph case it is necessary to select the most relevant matches of all possible ones. The first longest (maximum of optional nodes) match for each tree is selected. This is unfortunately not optimal and also not consistent with the procedural case, but it is the easiest option for the implementation (Netgraph can be directly used in that case.)

Negation, participant, participant quantity (count), action type These values represent numbers of matching nodes (or more precisely of pieces of extracted information) of the given type. For example values of participant are the numbers of all participants (node number 4 in the Netgraph based rule) identified by the extraction rule and values of *přežít* (survive) are numbers of matching action type nodes with the value of *přežít* (survive). Note that some postprocessing of the Netgraph based output was necessary to count TRUE and FLASE negation values.

Participant quantity Values in this group are all connected with the quantity kind of information. It expresses the quantity of participants involved in the corresponding incident action. This kind of information is numeric so some numerical calculations can be made (average value, median and maximum value). Again postprocessing of the Netgraph based output was necessary to obtain these values – in this case translation of seven kinds of numerals to numbers (*jeden* - 1, *dva* - 2, *tři* - 3, *čtyři* - 4, *šest* - 6, *sedm* - 7, *osm* - 8). Note that the vale of average is strongly affected by the very few high numbers present in the results (the values 1 and 2 accounted for more than half of the results.)

	correct	missing	spurious	recall	precision	F_1
injuries	3	29	0	0.09	1	0.17
fatalities	1	10	0	0.09	1	0.17

Table 8.2: Evaluation of the manually created rule form Figure 5.3 on the manually annotated dataset.

Values of participant quantity also demonstrate several errors made by the particular extraction rules, see bellow for details.

Detected errors

Results of the extraction were investigated only partly; no formal evaluation is available in this case (except on a small evaluation set, see bellow: the second experiment). About 10-20% of information was not extracted because of errors of linguistic parsing; majority of the errors was made in long complex sentences, which are known to be difficult for linguistic processing. There was only a few of false positives. A nice example can be traced from the maximum values of participant quantity in Table 8.1. Three different numbers were found in the three experiments: 27800, 4988 and 333. The number of 27800 is actually a number of chickens that died during one of the accidents. The number was extracted by the first Netgraph based rule because the query node 4 (participant) was marked as optional and omitted during the evaluation. This node normally ensures that the participant is one of: person, man, woman, child, driver, etc. Numbers 4988 and 333 are both correct. They were both used in the same sentence, which summarized numbers of injured (4988) and killed (333) people during one whole year. Although the sentence was quite complex it was linguistically parsed correctly and also the procedural rule managed to extract both numbers correctly. The Netgraph based rule extracted only the first number in the sentence because the current evaluation algorithm does not allow multiple matches per sentence (see above the comments of rule matches and effective rule matches).

8.1.2 Czech Fireman Qualitative

In the second experiment a manually annotated collection of 50 fireman news texts was used. Having the extraction rule from the previous experiment and a set of manually annotated texts it is only natural to ask a question about the success of the extraction rule on that collection. Table 8.2 summarizes the results. These results are far from satisfactory; the recall of 0.09 is something that is far from any acceptable use. Several explanations of the issue can be provided. The extraction rule serves more for a demonstration than for exhausting coverage of all possible cases. The extraction rule looks for particular verb (to injure, to die, etc.) but the information can be also expressed by an adjective (injured driver, death passenger, etc.); another extraction rules should be constructed for these and other cases. The training collection used for the design was also of a different spectrum of texts.

On the other hand this experiment shows how a manually annotated collection contributes to the quality of extraction rules. We can never know if the extraction rule is usable until a formal evaluated is made. Also the fact that the precision is strictly 1 should be noted. This means that the extraction rule made no mistake in those cases when it provided some output.

Manual Design of Rules Using Training Data Set

	correct	missing	spurious	recall	precision	F_1
manual rules	5	2	0	0.71	1	0.83
ILP rules	5	2	0	0.71	1	0.83

Table 8.3: Evaluation of the manually created rules and ILP learned rules (manually annotated dataset was used for rule design (training half) and evaluation (testing half) – see the description of the second experiment in text.)

	correct	missing	spurious	recall	precision	F_1
manual rules	4	1	1	0.8	0.8	0.8
ILP rules	4	1	1	0.8	0.8	0.8

Table 8.4: Cross method comparison of found instances.

Next question that naturally emerges is: How would be the performance if the rules were designed with the support of a manually annotated collection? An additional experiment was made to answer that question. The collection was split into two even parts – training part and testing part. A manually created rule was designed so that it correctly matched with all annotations of the training part and then it was evaluated on the testing part. For the validity of the experiment it was necessary that the designer did not have any knowledge about the data of the testing part; that is why we used a different extraction task (damage instead of injuries and fatalities). We have also compared the performance of the manually created rule with a rule learned by the ILP machine learning engine.

The results are summarized in Table 8.3. Both kinds of rules (manually designed and learned by ILP) performed the same (recall: 0.71, precision: 1); both the methods correctly found 5 instances and they were both unable to find 2 instances. From the cross coverage comparison in Table 8.4 it is apparent that the methods agreed on 4 instances and each method was able to discovered one instance that the other did not discover. Such results could be accepted for a practical application but we must not forget the fact that the collection is very small and only single evidence is provided by the experiment, so it dies not provide any statistical significance (getting statistically significant results would require experiments with various datasets, extraction tasks and human designers.)

8.2 Evaluation of Learned Rules

In this section, we present two experiments with two manually annotated datasets. Both the datasets are of the kind of *event extraction encoded as entity recognition* described in Section 2.1.7. The performance of our method based on ILP machine learning was measured and compared with an alternative extraction method based on propositional machine learning.

Evaluation of machine learning approaches is easier than evaluation of approaches based on manual extraction rules because it is not necessary to construct extraction rules for each experiment manually. Thanks to this fact and the availability of manually annotated datasets, the evaluation is more comprehensive than in the previous section.

8.2.1 Examples of Learned Rules

In Figure 8.1, we present the most representative examples of extraction rules learned from the whole dataset Czech Fireman Reports Manually Annotated (Section 7.3.2). The

```
% [cars - Rule 3] [Pos cover = 5 Neg cover = 0]
mention(cars,A) :-
   'lex.rf'(B,A), sempos(B,'n.denot'), tDependency(C,B), t_lemma(C,vozidlo),
   functor(C,'ACT'), number(C,sg).   % vozidlo ~ vehicle

% [damage - Rule 1] [Pos cover = 14 Neg cover = 0]
mention(damage,A) :-
   'lex.rf'(B,A), sempos(B,'n.quant.def'), tDependency(C,B), tDependency(C,D),
   t_lemma(D,'vyšetřovatel').   % vyšetřovatel ~ investigating officer

% [end_subtree - Rule 7] [Pos cover = 6 Neg cover = 0]
mention(end_subtree,A) :-
   'lex.rf'(B,A), sempos(B,'n.quant.def'), tDependency(C,B), t_lemma(C,'ukončit').
            % ukončit ~ finish

% [start - Rule 2] [Pos cover = 15 Neg cover = 0]
mention(start,A) :-
   'lex.rf'(B,A), functor(B,'TWHEN'), tDependency(C,B), tDependency(C,D),
   t_lemma(D,ohlásit).   % ohlásit ~ report (e.g. a fire)

% [injuries - Rule 1] [Pos cover = 7 Neg cover = 0]
mention(injuries,A) :-
   'lex.rf'(B,A), functor(B,'PAT'), tDependency(B,C), t_lemma(C,'zraněný'),
   tDependency(D,B), aspect(D,cpl).   % zraněný ~ injured

% [fatalities - Rule 1] [Pos cover = 3 Neg cover = 0]
mention(fatalities,A) :-
   'lex.rf'(B,A), functor(B,'PAT'), tDependency(C,B), t_lemma(C,srazit).
            % srazit ~ knock down

% [professional_unit - Rule 1] [Pos cover = 17 Neg cover = 0]
mention(professional_unit,A) :-
   'lex.rf'(B,A), functor(B,'LOC'), gender(B,fem), tDependency(C,B),
   functor(C,'CONJ'), overlap_Lookup_tToken(D,B).

% [amateur_unit - Rule 1] [Pos cover = 19 Neg cover = 0]
mention(amateur_unit,A) :-
   'lex.rf'(B,A), tDependency(C,B), tDependency(D,C), tDependency(D,E),
   t_lemma(E,dobrovolný).   % dobrovolný ~ voluntary
```

Figure 8.1: Rules with largest coverage for each task learned from the whole dataset Czech Fireman Reports Manually Annotated (Section 7.3.2).

rule with largest coverage for each extraction task was selected and it is provided in the figure. Each rule demonstrates a connection of the target token, annotated as '*mention(task_name)*', with other parts of the sentence through linguistic syntax structures.

For example the second rule (*damage* task, lines 6-9) connects the node A with its tectogrammatical counterpart – numeral B (*n.quant.def*) and with node D ('vyšetřovatel') representing the investigating officer who stated the mount of damage.

8.2.2 Evaluation Methods and Measures

Evaluation experiments have to be preformed repeatedly in order to investigate statistical significance. A dataset consisting of individual documents is randomly split into the training and testing part in each experimental run. The ratio between the training and testing part depends on the size of the dataset. It is important to have enough training data and therefore larger training part and smaller testing part is used if the dataset is too small. The cross validation technique meets exactly this requirement: the more cross validation folds, the greater the ratio. In following sections presenting individual experiments, different numbers of cross validation folds are used according to the sizes of the datasets.

In each experiment run, the performance is evaluated using several performance measures; precision, recall and F_1 (harmonic mean of precision and recall) are mostly used for information extraction experiments. Partially correct (or overlapping) matches can occur when annotations do not match exactly (they are overlapping). Strict, lenient and average variants of performance measures allow dealing with partially correct matches in different ways:

- Strict measures considers all partially correct matches as incorrect (spurious).
- Lenient measures considers all partially correct matches as correct.
- Average measures takes the average of strict and lenient.

See also the GATE documentation chapter about performance evaluation[1] and few details about the statistical significance in Section 9.5.

8.2.3 Comparison with PAUM Classifier

To compare our solution with other alternatives, we took the PAUM (Perceptron Algorithm with Uneven Margins) propositional learner from GATE [Li *et al.*, 2002]. The quality of propositional learning from texts is strongly dependent on the selection of right features. We obtained quite good results with features of a window of two preceding and two following token lemmas and morphological tags. The precision was further improved by adding the feature of *analytical function*[2] from the syntactic parser and information about presence of named entities.

8.2.4 Czech Fireman Performance

Table 8.5 summarizes performance evaluation of our ILP based method and its comparison with PAUM classifier on the dataset Czech Fireman Reports Manually Annotated

[1]http://gate.ac.uk/userguide/chap:eval

[2]http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/en/a-layer/html/ch03.html#s1-list-anal-func

Strict Precision

Task	ILP	PAUM
cars	0.324 ± 0.387	0.380 ± 0.249
damage	0.901 ± 0.178	0.860 ± 0.176
end subtree	0.529 ± 0.381	0.499 ± 0.242
start	0.929 ± 0.109	0.651 ± 0.152 ●
injuries	0.667 ± 0.291	0.398 ± 0.205 ●
fatalities	0.814 ± 0.379	0.307 ± 0.390 ●
professional unit	0.500 ± 0.241	0.677 ± 0.138 ○
amateur unit	0.863 ± 0.256	0.546 ± 0.293 ●
overall	0.691 ± 0.358	0.540 ± 0.297 ●

Strict Recall

Task	ILP	PAUM
cars	0.088 ± 0.129	0.353 ± 0.231 ○
damage	0.821 ± 0.261	0.933 ± 0.148 ○
end subtree	0.231 ± 0.203	0.601 ± 0.249 ○
start	0.908 ± 0.115	0.978 ± 0.058 ○
injuries	0.574 ± 0.309	0.814 ± 0.224 ○
fatalities	0.388 ± 0.449	0.536 ± 0.452 ○
professional unit	0.506 ± 0.191	0.811 ± 0.138 ○
amateur unit	0.886 ± 0.210	0.955 ± 0.096 ○
overall	0.550 ± 0.382	0.748 ± 0.312 ○

Strict F_1

Task	ILP	PAUM
cars	0.109 ± 0.147	0.335 ± 0.205 ○
damage	0.828 ± 0.217	0.876 ± 0.131 ○
end subtree	0.283 ± 0.219	0.525 ± 0.213 ○
start	0.912 ± 0.089	0.771 ± 0.111 ●
injuries	0.543 ± 0.280	0.498 ± 0.204
fatalities	0.306 ± 0.420	0.222 ± 0.308
professional unit	0.491 ± 0.200	0.730 ± 0.118 ○
amateur unit	0.827 ± 0.253	0.634 ± 0.296 ●
overall	0.537 ± 0.369	0.574 ± 0.295 ○

○, ● statistically significant improvement or degradation

Table 8.5: Evaluation on Czech Fireman dataset

(Section 7.3.2). The table shows results of the three main evaluation measures: strict precision, strict recall and strict F_1 for each extraction task as well as overall results. 8-fold cross validation was performed 8 times in the experiment. Average values and standard deviations are printed in the table and statistical significance is indicated. Root/subtree preprocessing/postprocessing (Section 5.2.4) was performed in the first three tasks: 'cars', 'damage' and 'end subtree'.

Although the precision of the ILP method was better in the majority of tasks and also its overall precision is statically better than the precision of the PAUM method, its recall was worse in all the measurements and also F_1 score is indicating better results of the

Time Training

Task	ILP			PAUM		
cars	3:28.0	±	0:28.3	0:02.2	±	0:00.1 •
damage	0:23.4	±	0:06.2	0:02.2	±	0:00.1 •
end subtree	1:34.2	±	0:14.4	0:02.2	±	0:00.1 •
start	0:28.9	±	0:03.1	0:02.2	±	0:00.1 •
injuries	1:30.7	±	0:15.0	0:02.2	±	0:00.1 •
fatalities	0:49.2	±	0:10.6	0:02.2	±	0:00.1 •
professional unit	2:37.4	±	0:27.4	0:02.2	±	0:00.1 •
amateur unit	0:24.0	±	0:04.6	0:02.2	±	0:00.1 •
overall	1:24.5	±	1:05.8	0:02.2	±	0:00.1 •

Time Testing

Task	ILP			PAUM		
cars	0:00.9	±	0:00.2	0:00.3	±	0:00.1 •
damage	0:00.9	±	0:00.1	0:00.3	±	0:00.0 •
end subtree	0:00.9	±	0:00.2	0:00.3	±	0:00.0 •
start	0:00.9	±	0:00.1	0:00.3	±	0:00.0 •
injuries	0:00.9	±	0:00.2	0:00.3	±	0:00.1 •
fatalities	0:00.9	±	0:00.1	0:00.3	±	0:00.0 •
professional unit	0:00.9	±	0:00.1	0:00.3	±	0:00.0 •
amateur unit	0:00.9	±	0:00.1	0:00.3	±	0:00.0 •
overall	0:00.9	±	0:00.1	0:00.3	±	0:00.0 •

• statistically significant decrease (not improvement in this case)

The time in this table is cumulative (sum over all folds in one experiment run).

Table 8.6: Time spent by ML engines on the Czech Fireman dataset.

PAUM method.

Table 8.6 provides learning and application (or testing) times of the compared methods. It is clear that learning time of the ILP method is many times higher than the time of the PAUM method. For example the 'cars' task took the ILP method approximately three and half minutes while the PAUM method spent only two seconds on it. On the other hand, the testing or application time of the ILP method is not that much higher than the time of the PAUM method but we have to notice that these times do not include linguistic preprocessing and deep language parsing needed by the ILP method will increase the time significantly.

Detailed results with much more evaluation measurements including lenient variants are available in the appendix – Section B.1.

8.2.5 Acquisitions Performance

The second experiment made with our ILP based extraction method was performed on the dataset Corporate Acquisition Events (Section 7.3.3). Table 8.7 provides the same measurements as in the previous experiment. This time, 2-fold cross validation was performed 10 times; root/subtree preprocessing/postprocessing was performed in the 'dlramt' task only and the technique of learning on named entity roots (see Section 5.2.5) was used on

Strict Precision

Task	ILP			PAUM			
acqabr	0.457	±	0.034	0.408	±	0.015	●
acquired	0.376	±	0.028	0.441	±	0.027	○
dlramt	0.286	±	0.052	0.597	±	0.030	○
purchabr	0.318	±	0.050	0.390	±	0.015	○
purchaser	0.403	±	0.037	0.474	±	0.029	○
seller	0.261	±	0.071	0.245	±	0.041	
sellerabr	0.306	±	0.111	0.219	±	0.036	●
overall	0.344	±	0.088	0.396	±	0.125	○

Strict Recall

Task	ILP			PAUM			
acqabr	0.180	±	0.025	0.517	±	0.027	○
acquired	0.272	±	0.054	0.512	±	0.033	○
dlramt	0.276	±	0.043	0.740	±	0.063	○
purchabr	0.120	±	0.041	0.514	±	0.033	○
purchaser	0.340	±	0.064	0.556	±	0.028	○
seller	0.131	±	0.047	0.226	±	0.046	○
sellerabr	0.050	±	0.027	0.190	±	0.032	○
overall	0.196	±	0.106	0.465	±	0.184	○

Strict F_1

Task	ILP			PAUM			
acqabr	0.258	±	0.030	0.456	±	0.015	○
acquired	0.313	±	0.046	0.473	±	0.024	○
dlramt	0.280	±	0.046	0.659	±	0.022	○
purchabr	0.172	±	0.050	0.443	±	0.016	○
purchaser	0.367	±	0.051	0.511	±	0.022	○
seller	0.170	±	0.050	0.232	±	0.036	○
sellerabr	0.085	±	0.044	0.202	±	0.030	○
overall	0.235	±	0.101	0.425	±	0.150	○

○, ● statistically significant improvement or degradation

Table 8.7: Evaluation on Acquisitions dataset

the rest of the tasks.

On this dataset, the PAUM method performed significantly better than the ILP method. There are only three tasks ('acqabr', 'seller' and 'sellerabr'), where the ILP method achieved better precision than the PAUM method and only two of the measurements are statistically significant.

Table 8.8 provides learning and application (or testing) times on this dataset. Again, learning times of the ILP method are several times higher than the times of the PAUM method and there is not so big difference between application times, although the same notice about the necessity of deep language parsing in the preprocessing step pays also in this case.

An interesting observation can be made when comparing training times of the ILP method on tasks, where the technique of learning on named entity roots was performed,

Time Training

Task	ILP			PAUM		
acqabr	16:24.9	±	8:22.9	0:10.0	±	0:00.6 ●
acquired	10:16.0	±	7:11.1	0:09.8	±	0:00.4 ●
dlramt	23:27.6	±	15:45.7	0:10.8	±	0:02.8 ●
purchabr	16:51.6	±	9:14.7	0:10.2	±	0:01.4 ●
purchaser	12:27.1	±	7:22.6	0:10.4	±	0:01.2 ●
seller	9:12.2	±	3:55.2	0:10.1	±	0:01.0 ●
sellerabr	6:11.5	±	4:03.3	0:09.7	±	0:00.3 ●
overall	13:33.0	±	10:00.1	0:10.1	±	0:01.3 ●

Time Testing

Task	ILP			PAUM		
acqabr	0:27.4	±	0:00.8	0:13.6	±	0:00.3 ●
acquired	0:26.7	±	0:00.7	0:11.8	±	0:00.3 ●
dlramt	0:30.5	±	0:00.8	0:09.8	±	0:00.5 ●
purchabr	0:27.4	±	0:00.9	0:13.2	±	0:00.5 ●
purchaser	0:27.5	±	0:02.0	0:11.5	±	0:01.2 ●
seller	0:27.3	±	0:03.2	0:10.7	±	0:00.8 ●
sellerabr	0:26.1	±	0:00.7	0:11.5	±	0:00.3 ●
overall	0:27.6	±	0:02.0	0:11.7	±	0:01.4 ●

● statistically significant decrease (not improvement in this case)

Table 8.8: Time spent by ML engines on the Acquisitions dataset.

with the remaining task ‘dlramt’. Table 7.3 presents numbers of annotations per extraction task and it can be seen that the training time quite correlate with the number of annotations. But the task ‘dlramt’ has almost the least instances form the tasks used in the evaluation and its average training time is about two times higher than the time spent on other tasks and from there it can be seen that the technique of learning on named entity roots can save training time.

Detailed results with much more evaluation measurements including lenient variants are available in the appendix – Section B.2.

8.2.6 Comparison with Results Reported in Literature

Other researchers used the Acquisitions dataset and reported the performance of their information extraction systems in the literature. Table 8.9 provides comparison of four extraction systems (SRV [Freitag, 1999], HMM [Freitag and McCallum, 1999], Elie [Finn and Kushmerick, 2004] and SVM+ILP [Ramakrishnan *et al.*, 2007]; short description of these systems was provided in Section 3.1.4) with our results obtained in the experiment described in the previous section. Table 8.9 compares values of F_1 measure (displayed as percentage, 100-times higher values than in previous tables) for individual tasks and it also provides overall weighted average for each method obtained by following formula:

$$\mathit{overall}\ F_1 = \frac{\sum_{t \in Tasks} A_t F_{1t}}{\sum_{t \in Tasks} A_t}$$

Where A_t represents the number of annotations of the particular extraction task and F_{1t} represents the actual value of F_1 measure on the extraction task.

	Annotations		Extraction Method					
Task	ver. A	ver. B	SRV	HMM	Elie	SVM+ILP	ILP	PAUM
acquired	**683**	651	38.5	*30.9*	43.5	41.8	31.3	**47.3**
acqabr	**1450**	1494	38.1	40.1	39.7	42.6	*25.8*	**45.6**
purchaser	**624**	594	45.1	48.1	46.2	45.4	*36.7*	**51.1**
purchabr	1263	**1347**	**48.5**	n/a	28.7	35.4	*17.2*	44.3
seller	267	**707**	23.4	n/a	*15.6*	**51.5**	17.0	23.2
sellerabr	431	**458**	**25.1**	n/a	13.4	21.7	*8.5*	20.2
dlramt	**283**	206	61.8	55.3	59.0	53.0	*28.0*	**65.9**
Total/Overall	5001	**5457**	41.1	n/a	33.5	40.8	*23.9*	**44.0**

Table 8.9: Performance comparison of reported results, F_1 measure.

Unfortunately not all the results can be directly compared because experiments of Ramakrishnan *et al.* [2007] (SVM+ILP) were performed on a different version of the dataset. First two columns of Table 8.9 show the difference of the two versions of the dataset in terms of numbers of annotations per extraction task. Columns corresponding to the second version (ver. B) of the dataset have gray background for better distinction. Although not all the numbers are directly comparable, we still compare them all together because, for the majority of the tasks (except 'seller'), the difference of the datasets is very small. Maximum values are highlighted in bold and minimum in italic.

Very interesting observation relates to the results of the PAUM method, which performed the best on most tasks, which is quite surprising because the method is relatively simple and no special learning features were designed for it. On the other hand, our ILP based method (the one before the last column of the table) did not demonstrate any major improvement, although a few lower values can be found in the table ('seller' Elie and 'acquired' HMM).

The reason why we computed the overall F_1 measure as the weighted average is that Ramakrishnan *et al.* [2007] did so. Their paper provides also the overall F_1 measure for a plain ILP variant of their method, which is very close to our ILP based method. They provide following overall values of the F_1 measure:

39.8 ± 0.9 for the original SVM + ILP approach and

35.2 ± 1.5 for the plain ILP variant.

If we compare the first value with the one reported in Table 8.9 (Overall value in the SVM+ILP column) it is clear that they are not the same[3], but they are quite similar and from there we can see that also their plain ILP approach was more successful than ours (35.2 vs. 23.9 F_1 measure), although on different version of the dataset.

8.3 Evaluation of Shareable Extraction Ontologies

In this section we present an experiment that should serve as a proof of a concept that the proposed idea of independent extraction ontologies is realizable. We have selected several reasoners (namely Jena, HermiT, Pellet and FaCT++) and tested them on two slightly different datasets from two different domains and languages (see Table 8.10). This should at least partially demonstrate the universality of the proposed approach.

In both cases the task is to find all instances (corresponding to words in a document)

[3]Their values were probably collected during each experiment run based on the actual performance and annotation counts.

dataset	domain	language	number of files	dataset size (MB)	number of rules
czech_fireman	accidents	Czech	50	16	2
acquisitions	finance	English	600	126	113

Table 8.10: Description of datasets that were used.

that should be uncovered by the extraction rules. The extraction rules are saved in single extraction ontology for each dataset. The datasets are divided into individual document ontologies (owl files) corresponding to the individual documents. During the experiment the individual document ontologies are processed separately (one ontology in a step) by a selected reasoner. The total time taken to process all document ontologies of a dataset is the measured result of the reasoner for the dataset.

The actual reasoning tasks are more difficult than a simple retrieval of all facts entailed by the extraction rules. Such simple retrieval task took only a few seconds for the Acquisitions v1.1 dataset (including parsing) in the native Prolog environment that the IE engine uses. There were several more inferences needed in the reasoning tasks because the schema of the input files was a little bit different from the schema used in rules. The mapping of the schemas was captured in another "mapping" ontology that was included in the reasoning. The mapping ontology is a part of the publically available project ontologies.

8.3.1 Datasets

In the experiment we used two slightly different datasets from two different domains and languages. Table 8.10 summarizes some basic information about them. The fist dataset is called 'czech_fireman', it is based on Czech texts that are reporting on fire and traffic accidents and it was already described in Section 7.3.4. The second dataset is called "Corporate Acquisition Events", it is based on news articles describing acquisition events, taken from the Reuters dataset and it was already described in Section 7.3.5.

8.3.2 Reasoners

Four OWL reasoners were used in the experiment(namely Jena[4] ,HermiT[5] ,Pellet[6] and FaCT++[7]) and the time they spent on processing a particular dataset was measured. The time also includes time spent on parsing the input. HermiT, Pellet and FaCT++ were called through OWL API-3.1, so the same parser was used for them. Jena reasoner was used in its native environment with the Jena parser. In the early beginning of the experiment we had to exclude the FaCT++ reasoner from both tests. It turned out that FaCT++ does not work with rules[8] and it did not return any result instances. All the remaining reasoners strictly agreed on the results and returned the same sets of instances.

Also HermiT was not fully evaluated on the Acquisitions v1.1 dataset because it was too slow. The reasoner spent 13 hours of running to process only 30 of 600 files of the dataset. And it did not seem useful to let it continue. We contacted the authors of HermiT

[4]http://jena.sourceforge.net
[5]http://hermit-reasoner.com
[6]http://clarkparsia.com/pellet
[7]http://code.google.com/p/factplusplus
[8]http://en.wikipedia.org/wiki/Semantic_reasoner#Reasoner_comparison

reasoner	**czech_fireman**	stdev	**acquisitions-v1.1**	stdev
Jena	161 s	0.226	1259 s	3.579
HermiT	219 s	1.636	$\gg$ 13 hours	
Pellet	11 s	0.062	503 s	4.145
FaCT++	Does not support rules.			

Time is measured in seconds. Average values from 6 measurements. Experiment environment: Intel Core I7-920 CPU 2.67GHz, 3GB of RAM, Java SE 1.6.0_03, Windows XP.

Table 8.11: Time performance of tested reasoners on both datasets.

to clarify this issue and we were reminded that HermiT was mainly designed as a TBox reasoner and it was particularly optimized for the ontology classification problem [Glimm *et al.*, 2010]; it was never optimized for simple rule based (ABox) inferences.

8.3.3 Evaluation Results

Table 8.11 summarizes results of the experiment. The standard deviations are relatively small when compared to the differences between the average times. So there is no doubt about the order of the tested reasoners. Pellet performed the best and HermiT was the slowest amongst the tested and usable reasoners in this experiment.

From the results we can conclude that similar tasks can be satisfactorily solved by contemporary OWL reasoners because three of four tested reasoners were working correctly and two reasoners finished in bearable time.

On the other hand even the fastest system took 8.5 minutes to process 113 rules over 126MB of data. This is clearly significantly longer than a operational system would require. Contemporary Semantic Web reasoners are known still to be often quite inefficient and the experiment showed that using them today to do information extraction will result in quite poor performance. However, efficiency problems can be solved and in the context of Linked Data providing shareable descriptions of information extraction rules may be valuable.

8.4 Evaluation of Fuzzy ILP Classification

We have evaluated both ILP methods and compared them with other machine learning procedures used in data mining. To make the comparison clear and easy to perform, we have implemented an interface between the ILP methods and Weka (Section 4.8). This interface makes it possible to use the ILP methods as an ordinary Weka classifier for any[9] classification task inside the Weka software. This also makes the presented experiments easily repeatable (see Secrion 9.7 for details.)

8.4.1 Czech Fireman Performance

For our experiment we used the Weka Experimenter (see in Section 4.8) and performed an experiment in which the Crisp and Fuzzy ILP classifiers were compared with five additional classifiers:

[9]For the fuzzy ILP method, there is a requirement on the target (class) attribute: it has to be monotonizable (e.g. numeric).

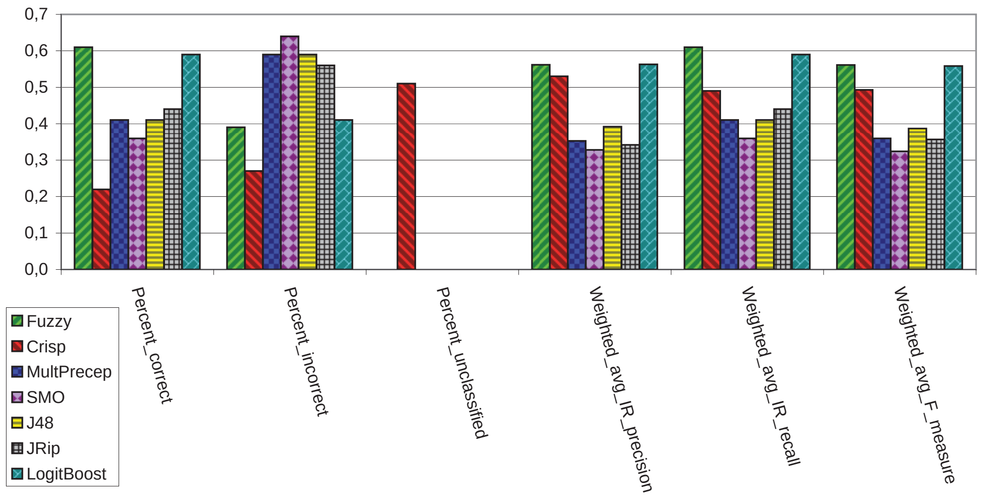


Figure 8.2: Evaluation of the methods – average values.

	Fuzzy	Crisp	MultPerc	SMO	J48	JRip	LBoost
Corr	0.61±.19	.22±.17 ●	.41±.19 ●	.36±.24 ●	.41±.22 ●	.44±.17 ●	.59±.26
Incor	.39±.19	.27±.24	.59±.19 ○	.64±.24 ○	.59±.22 ○	.56±.17 ○	.41±.26
Uncl	.00±.00	.51±.29 ○	.00±.00	.00±.00	.00±.00	.00±.00	.00±.00
Prec	.56±.24	.53±.37	.35±.20 ●	.33±.26	.39±.22	.34±.21 ●	.56±.28
Rec	.61±.19	.49±.32	.41±.19 ●	.36±.24 ●	.41±.22 ●	.44±.17 ●	.59±.26
F	.56±.20	.49±.33	.36±.19 ●	.32±.24 ●	.39±.21	.36±.19 ●	.56±.27

○, ● statistically significant increase or decrease

Legend:

Fuzzy	czsem.ILP.FuzzyILPClassifier ”
Crisp	czsem.ILP.CrispILPClassifier ”
MultPerc	functions.MultilayerPerceptron ’-L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a’
SMO	functions.SMO ’-C 1.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K \”functions.supportVector.PolyKernel -C 250007 -E 1.0\”’
J48	trees.J48 ’-C 0.25 -M 2’
JRip	rules.JRip ’-F 3 -N 2.0 -O 2 -S 1’
LBoost	meta.LogitBoost ’-P 100 -F 0 -R 1 -L -1.7976931348623157E308 -H 0.1 -S 1 -I 10 -W trees.DecisionStump’

Corr	Percent correct
Inor	Percent incorrect
Uncl	Percent unclassified
Prec	IR precision, weighted average from all classes
Rec	IR recall, weighted average from all classes
F	F measure, weighted average from all classes

Table 8.12: Evaluation of the methods in 2 times 10-fold cross validation.

dataset	Fuzzy	Crisp	MultPerc	SMO	J48	JRip	LBoost	train	test
car	.39±.03	.36±.03 ●	.53±.02 ○	.57±.01 ○	.50±.02 ○	.51±.03 ○	.54±.02○	173	1554
wine	.44±.03	.42±.02 ●	.48±.02 ○	.46±.02 ○	.47±.02 ○	.48±.03 ○	.52±.02○	160	1439
cmc	.79±.02	.77±.03 ●	.89±.02 ○	.81±.01 ○	.88±.02 ○	.82±.03 ○	.85±.02○	147	1325
tae	.50±.12	.39±.11 ●	.59±.11 ○	.55±.12 ○	.50±.12	.37±.11 ●	.55±.11○	135	15
pop	.66±.09	.54±.17 ●	.57±.13 ●	.70±.06 ○	.70±.07 ○	.70±.06 ○	.66±.10	80	9
nurs	.79±.04	.68±.06 ●	.81±.04 ○	.73±.04 ●	.80±.04 ○	.79±.06	.83±.02○	52	12907

○, ● statistically significant improvement or degradation

Legend:

train ....average number (±1) of training instances in each run
test ....average number (±1) of testing instances in each run

Learning parameters for both ILP methods:
set(noise,20). set(i,3). set(clauselength,13). set(search,heuristic). set(evalfn,wracc). set(samplesize,3).

Table 8.13: Evaluation of the methods on UCI datasets, **percent correct**, average values from 100 repetitions.

- Multilayer Perceptron [Bishop, 1996],
- Support Vector Machine classifier SMO [Keerthi *et al.*, 2001],
- J48 decision tree [Quinlan, 1993],
- JRip rules [Cohen, 1995] and
- Additive logistic regression LogitBoost [Friedman *et al.*, 2000].

We have evaluated all the methods two times by 10-fold cross validation. The obtained results (average values) are described by the graph in Figure 8.2 and in Table 8.12 (with standard deviations and marked statistically significant values).

There is no clear winner in our experiment. But the Fuzzy ILP classifier proved better results than a majority of the methods on our data and the results are statistically significant in many cases. Very good results were also obtained using LogitBoost.

8.4.2 UCI Performance

The Fuzzy ILP classifier performed quite well on our dataset, but the next question is: How is it with other data, with more learning instances, and what about the time complexity? To answer these questions we performed another experiment. We selected several datasets from the UCI repository (see Section 7.3.7) and evaluated all the methods against them.

All the selected datasets are monotonizable (the target attribute can be naturally ordered), so the fuzzy classifier could take advantage of that. Learning settings are the same as before (Table 8.12) except for settings of both ILP classifiers, which performed a little bit better with modified settings on a majority of the datasets (see in the legend of Table 8.13).

Table 8.13 compares the numbers of correctly classified instances on all the datasets. The last two columns show numbers of training and testing instances. The numbers of training instances are quite low; this is because the ILP classifiers are probably not capable of fully exploiting higher numbers of training instances and the difference between ILP classifiers and the others would be even a bit higher. This is demonstrated in Figure 8.3 (for 'nursery' dataset only). It can be seen that when the number of training instances was under about 40, the fuzzy classifier performed better than some of the others (SMO, JRip and Multilayer Perceptron), but from about 60 training instances further, both ILP classifiers performed worse than the others.

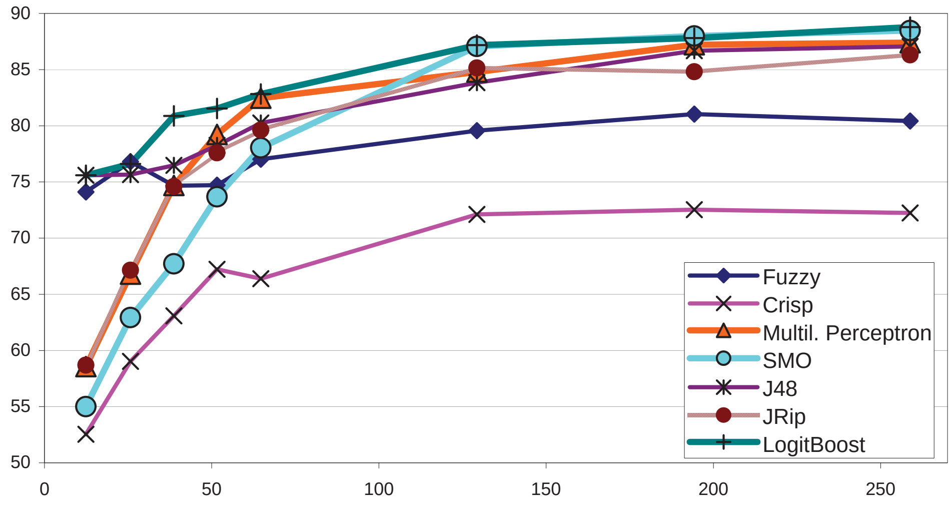


Figure 8.3: x-axis: number of training instances, y-axis: percent of correctly classified instances, average values from 10 repetitions, 'nursery' dataset.

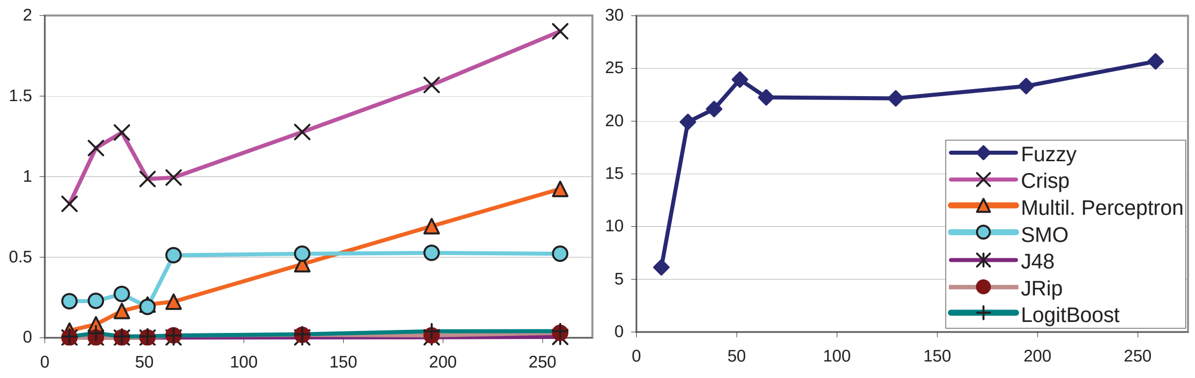


Figure 8.4: x-axis: number of training instances, y-axis: training time in seconds, average values from 10 repetitions, 'nursery' dataset.

8.4.3 UCI Time

Figure 8.4 demonstrates time complexity of the classifiers in the same experiment as in Figure 8.3. Despite the fact that the Fuzzy ILP classifier was several times slower than the Crisp ILP classifier and even more than the others, it is still computable on current processors (e.g. P9600, 2.66 GHz, which we used) and the curve of time complexity did not grow rapidly during the experiment. Because ILP is a heuristic and iterative method, the time complexity can be quite directly managed by the setting of learning parameters.

9. Conclusion

This is the last chapter of the thesis. It contains mainly concluding remarks, some additional discussions and an outlook to the future work. First four sections are dedicated to the four main topics of the thesis, followed by common remarks and discussions.

9.1 Manual Design of Extraction Rules

We have presented the development of our extraction method based on deep language parsing from its early beginning. Its evolution begun with procedurally written extraction rules followed by declarative ones and, finally, it was supported by the automated induction of extraction rules using the machine learning technique of Inductive Logic Programming.

Our extraction method based on manually designed extraction rules do not consider the annotation aspect of information extraction. Although it is possible to infer an annotation based variety of the presented method, we did not take it into account because during the development of the method the aim was more to produce structured data from text than to produce annotated documents. In the second approach the emphasis is inverted.

A deeper evaluation of the method would be definitely interesting, but at the moment, the information provided is the only available. There is no real world application of the method outside the academic ground. The method is still waiting for deep testing and further development in an extensive real world project.

9.2 Machine Learning of Extraction Rules

From our experiments can be seen that ILP is capable to find complex and meaningful rules that cover the intended information. But in terms of the performance measures the results are not better than those from a propositional learner. This is quite surprising observation mainly for Czech because it is a language with free word order and we would expect much better results of the dependency based approach than those of the position based approach, which was used by the propositional learner.

Our method is still missing an intelligent semantic interpretation procedure and it could be also evaluated on other datasets (e.g. MUC, ACE, TAC, CoNLL) and other languages. So far we also do not provide a method for classical event and relation extraction. In the present solution, we work with events in the same way as with entities; details about this variant of event extraction were provided in Section 2.1.7. The method has to be adapted for explicit learning of events and relations in the form of "subject predicate object".

Our method can also provide a comparison of different linguistic formalisms and tools and their benefit to information extraction because we could run our method using different linguistic analyzers and compare the results on the same dataset.

9.3 Shareable Extraction Ontologies

9.3.1 From Annotations to Real World Facts

In this thesis, we have described our method how to apply an extraction ontology to a document ontology and obtain so called "annotated" document ontology. To have an "annotated" document ontology is almost the same as to have an annotated document. An annotated document is useful (easier navigation, faster reading and lookup of information, possibility of structured queries on collections of such documents, etc.) but if we are interested in the actual information present in the document, if we want to know the facts that are in a document asserted about the real word things then an annotated document is not sufficient. But the conversion of an annotated document to the real world facts is not simple. There are obvious issues concerning data integration and duplicity of information. For example when in a document two mentions of people are annotated as ‘injured’, what is then the number of injured people in the corresponding accident? Are the two annotations in fact linked to the same person or not?

In the beginning of our work on the idea of shareable extraction ontologies we planned to develop it further, we wanted to cover also the step from annotated document ontologies to the real world facts. The extraction process would then end up with so called “fact ontologies”. But two main obstacles prevent us to do that.

1. Our IE engine is not yet capable to solve these data integration and duplicity of information issues and the real world facts would be quite imprecise then.
2. There are also technology problems of creating new facts (individuals) during reasoning.

Because of the decidability and finality constraints of the Description Logic Reasoning it is not possible to create new individuals during the reasoning process. There is no standard way how to do it. But there are some proprietary solutions like `swrlx:createOWLThing`[1] from the Protégé project and `makeTemp(?x)` or `makeInstance(?x, ?p, ?v)`[2] from the Jena project. And these solutions can be used in the future work.

9.3.2 How to Obtain a Document Ontology?

It is true that there are standard means how to obtain a (document) ontology from a document, but these means were not intended for a complex transformation of documents to linguistic structures. Also in our case study (Section 5.3.2), we did not show how the linguistic preprocessing could be embedded in a GRDDL transformation. In fact, our input documents were already linguistically preprocessed before the GRDDL transformation was performed. We can say that this is just a technical problem but it also illustrates that shareable extraction ontologies and probably also extraction ontologies are still rather a vision, however this vision is realizable today.

9.3.3 SPARQL Queries – Increasing Performance?

There is also a possibility to transform the extraction rules to SPARQL construct queries. This would probably rapidly increase the time performance. However a document ontology would then have to exactly fit with the schema of the extraction rules. This would be a minor problem.

The reason why we did not study this approach from the beginning is that we were interested in extraction *ontologies* and SPARQL queries are not currently regarded as a part of an ontology and nothing is suggesting it to be that way.

Anyway the performance comparison remains a valuable task for the future work.

[1]`http://protege.cim3.net/cgi-bin/wiki.pl?action=browse&id=SWRLExtensionsBuiltIns`
[2]`http://jena.sourceforge.net/inference/#RULEbuiltins`

9.3.4 Contributions for Information Extraction

Our work on extraction ontologies combines the field of ontology-based information extraction and rule-based reasoning. The aim is to show a new possibility in usage of IE tools and reasoners. The idea of shareable extraction ontologies do not bring a solution that would improve the performance of IE tools.

We also do not provide a proposal of a universal extraction ontology format (although a specific form for the rule based extraction on dependency parsed text could be inferred). This task is left for the future if a need for such activity emerges.

9.3.5 Summary

In the beginning of the description of shareable extraction ontologies, we pointed out the draw back of so called extraction ontologies – in most cases they are dependent on a particular extraction/annotation tool and they cannot be used separately.

We extended the concept of extraction ontologies by adding the shareable aspect and we introduced a new principle of making extraction ontologies independent of the original tool: the possibility of application of an extraction ontology to a document by an ordinary reasoner.

In Section 5.3.2 we presented a case study that shows that the idea of shareable extraction ontologies is realizable. We presented implementation of our IE tool that exports its extraction rules to an extraction ontology and we demonstrated how this extraction ontology can be applied to a document by a reasoner.

Moreover in Section 8.3 an experiment with several OWL reasoners was presented. The experiment evaluated the performance of contemporary OWL reasoners on IE tasks (application of extraction ontologies). A new publically available benchmark for OWL reasoning was created together with the experiment. Other reasoners can be tested this way.

9.4 Fuzzy ILP Classification

In our work on fuzzy ILP classification, we provided a design and partial implementation of a fuzzy system, which provides a fuzzy classification of textual reports. Our approach is based on usage of third party linguistic analyzers, our work on information extraction, and fuzzy inductive logic programming.

The main contributions are formal models, prototype implementation of the presented methods and evaluation experiments. The first experiment evaluated performance of the presented methods and compared them with other machine learning procedures used in data mining on our dataset. The Fuzzy ILP Classifier proved better results than a majority of the methods. The results are statistically significant in many cases. We see the advantage of the Fuzzy ILP classifier in the fact that monotonization leads to the extension of the learning domain and it utilizes the fact that the domain is or can be monotonically ordered.

In the second experiment, we evaluated all the methods on other datasets with more training instances and we also experimentally measured the time complexity of the methods. This experiment shows that the fuzzy method is suitable mainly in situations with a small amount of training instances and in cases when the target attribute mostly respects the natural order of the remaining attributes. But this did not hold true for any of the later-used datasets. When comparing the fuzzy approach with the crisp one, the fuzzy

Czsem Mining Suite installation:	`http://czsem.berlios.de/czsem_install.html`
Fuzzy ILP Classifier for Weka:	`http://www.ksi.mff.cuni.cz/~dedek/fuzzyILP/`
Ontologies: `http://czsem.berlios.de/ontologies/acquisitions-v1.1/download_instructions.html` `http://czsem.berlios.de/ontologies/czech_fireman/download_instructions.html`	
General download area: `http://developer.berlios.de/project/filelist.php?group_id=8427`	
Apache Maven repository:	`http://czsem.berlios.de/maven2/`

Table 9.1: Download links to the implementation and datasets.

approach always performed better in terms of correctness of the classification, but it was many times slower than all the methods in terms of time complexity.

9.5 Statistical Significance

The term statistical significance used in this thesis refers to the result of a pair-wise comparison of learning engines using the corrected resampled (two tailed) T-Test [Nadeau and Bengio, 2003], which is suitable for cross validation based experiments. The Weka implementation was used (see some details in Section 4.8). Test significance was 0.05 in all cases.

9.6 How to Download

The project website[3] provides several ways how to get all the presented tools running. A platform independent installer, Java binaries and source codes are provided under the GPL license.

Also majority of the datasets mentioned in this thesis are available for public download. Table 9.1 provides links to individual resources.

9.7 Repeatability of Experiments

Our implementation and datasets are publicly available. This makes our experiments repeatable according to the SIGMOD Experimental Repeatability Requirements [Manolescu *et al.*, 2008]. If you have any problems concerning the reconstruction of any of the experiments please contact the authors, they will pleased by providing an assistance.

9.8 Summary

Our experiments deal with texts in the Czech language but our method is general and it can be used with any structured linguistic representation.

[3]`http://czsem.berlios.de`

Bibliography

Stuart Aitken (2002), Learning Information Extraction Rules: An Inductive Logic Programming approach, in F. van Harmelen, editor, *Proceedings of the 15th European Conference on Artificial Intelligence*, IOS Press, Amsterdam, URL http://www.aiai.ed.ac.uk/~stuart/AKT/ilp-ie.html.

Tim Berners-Lee, James Hendler, and Ora Lassila (2001), The Semantic Web, A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities, *Scientific American*, 284(5):34–43.

Jan C. Bioch and Viara Popova (2000), Rough Sets and Ordinal Classification, in *ALT '00: Proceedings of the 11th International Conference on Algorithmic Learning Theory*, pp. 291–305, Springer-Verlag, London, UK, ISBN 3-540-41237-9.

Jan C. Bioch and Viara Popova (2009), Monotone Decision Trees and Noisy Data, Research Paper ERS-2002-53-LIS, Erasmus Research Institute of Management (ERIM), URL http://econpapers.repec.org/RePEc:dgr:eureri:2002206.

Christopher M. Bishop (1996), *Neural Networks for Pattern Recognition*, Oxford University Press, 1 edition, ISBN 0198538642, URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0198538642.

Kalina Bontcheva, Valentin Tablan, Diana Maynard, and Hamish Cunningham (2004), Evolving GATE to Meet New Challenges in Language Engineering, *Natural Language Engineering*, 10(3/4):349—373.

Willem Nico Borst (1997), *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*, Ph.D. thesis, Universiteit Twente, Enschede, URL http://doc.utwente.nl/17864/.

Sabine Buchholz and Erwin Marsi (2006), CoNLL-X shared task on multilingual dependency parsing, in *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06, pp. 149–164, Association for Computational Linguistics, Stroudsburg, PA, USA, URL http://portal.acm.org/citation.cfm?id=1596305.

Razvan Bunescu and Raymond Mooney (2007), Extracting Relations from Text: From Word Sequences to Dependency Paths, in Anne Kao and Stephen R. Poteet, editors, *Natural Language Processing and Text Mining*, chapter 3, pp. 29–44, Springer, London, ISBN 978-1-84628-175-4, doi:10.1007/978-1-84628-754-1_3, URL http://dx.doi.org/10.1007/978-1-84628-754-1_3.

Razvan Constantin Bunescu (2007), *Learning for Information Extraction: From Named Entity Recognition and Disambiguation To Relation Extraction*, Ph.D. thesis, Department of Computer Sciences, University of Texas at Austin, URL http://www.cs.utexas.edu/users/ml/papers/razvan-dissertation.pdf.

Ekaterina Buyko, Erik Faessler, Joachim Wermter, and Udo Hahn (2009), Event extraction from trimmed dependency graphs, in *BioNLP '09: Proceedings of the Workshop on BioNLP*, pp. 19–27, Association for Computational Linguistics, Morristown, NJ, USA, ISBN 978-1-932432-44-2.

Ekaterina Buyko and Udo Hahn (2010), Evaluating the impact of alternative dependency graph encodings on solving event extraction tasks, in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pp. 982–992, Association for Computational Linguistics, Stroudsburg, PA, USA, URL http://dl.acm.org/citation.cfm?id=1870658.1870754.

Chia-Hui Chang, Mohammed Kayed, Moheb Ramzy Girgis, and Khaled F. Shaalan (2006), A Survey of Web Information Extraction Systems, *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1411–1428, ISSN 1041-4347, doi:http://doi.ieeecomputersociety.org/10.1109/TKDE.2006.152.

Miao Chong, Ajith Abraham, and Marcin Paprzycki (2005), Traffic Accident Analysis Using Machine Learning Paradigms, *Informatica*, 29:89–98.

Silvie Cinková, Jan Hajič, Marie Mikulová, Lucie Mladová, Anja Nedolužko, Petr Pajas, Jarmila Panevová, Jiří Semecký, Jana Šindlerová, Josef Toman, Zdeňka Urešová, and Zdeněk Žabokrtský (2006), Annotation of English on the tectogrammatical level, Technical Report 35, UFAL MFF UK, URL http://ufal.mff.cuni.cz/pedt/papers/TR_En.pdf.

Andrew B. Clegg and Adrian J. Shepherd (2005), Evaluating and integrating treebank parsers on a biomedical corpus, in *Proceedings of the Workshop on Software*, Software '05, pp. 14–33, Association for Computational Linguistics, Stroudsburg, PA, USA, URL http://portal.acm.org/citation.cfm?id=1626317.

William W. Cohen (1995), Fast Effective Rule Induction, in *In Proceedings of the Twelfth International Conference on Machine Learning*, pp. 115–123, URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.50.8204.

Michael Collins, Jan Hajič, Eric Brill, Lance Ramshaw, and Christoph Tillmann (1999), A Statistical Parser of Czech, in *Proceedings of 37th ACL Conference*, pp. 505–512, University of Maryland, College Park, USA.

Corinna Cortes and Vladimir Vapnik (1995), Support-vector networks, *Machine Learning*, 20:273–297, ISSN 0885-6125, URL http://dx.doi.org/10.1007/BF00994018, 10.1007/BF00994018.

Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan (2002), GATE: A framework and graphical development environment for robust NLP tools and applications, in *Proceedings of the 40th Anniversary Meeting of the ACL.*

Hamish Cunningham, Diana Maynard, and Valentin Tablan (2000), JAPE: a Java Annotation Patterns Engine, Technical report, Department of Computer Science, The University of Sheffield, URL http://www.dcs.shef.ac.uk/intranet/research/resmes/CS0010.pdf.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning (2006), Generating Typed Dependency Parses from Phrase Structure Parses, in *Proceedings of the IEEE / ACL 2006 Workshop on Spoken Language Technology*, The Stanford Natural Language Processing Group, URL http://nlp.stanford.edu/pubs/LREC06_dependencies.pdf.

Jan Dědek and Peter Vojtáš (2008), Computing aggregations from linguistic web resources: a case study in Czech Republic sector/traffic accidents, in Cosmin Dini, editor, *Second International Conference on Advanced Engineering Computing and Applications in Sciences*, pp. 7–12, IEEE Computer Society, ISBN 978-0-7695-3369-8, URL `http://dx.doi.org/10.1109/ADVCOMP.2008.17`.

Saso Dzeroski and Nada Lavrac, editors (2001), *Relational Data Mining*, Springer, Berlin, URL `http://www-ai.ijs.si/SasoDzeroski/RDMBook/`.

Pavel Češka (2006), Segmentace textu, Bachelor's Thesis, MFF, Charles University in Prague.

David W. Embley (2004), Toward semantic understanding: an approach based on information extraction ontologies, in *Proceedings of the 15th Australasian database conference - Volume 27*, ADC '04, pp. 3–12, Australian Computer Society, Inc., Darlinghurst, Australia, Australia, URL `http://portal.acm.org/citation.cfm?id=1012294.1012295`.

David W. Embley, Cui Tao, and Stephen W. Liddle (2002), Automatically Extracting Ontologically Specified Data from HTML Tables of Unknown Structure, in Stefano Spaccapietra, Salvatore T. March, and Yahiko Kambayashi, editors, *ER*, volume 2503 of *Lecture Notes in Computer Science*, pp. 322–337, Springer, ISBN 3-540-44277-4.

Oren Etzioni, Michele Banko, Stephen Soderland, and Daniel S. Weld (2008), Open information extraction from the web, *Commun. ACM*, 51(12):68–74, ISSN 0001-0782, doi:http://doi.acm.org/10.1145/1409360.1409378, URL `http://portal.acm.org/citation.cfm?id=1409378`.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning (2005), Incorporating non-local information into information extraction systems by Gibbs sampling, in *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pp. 363–370, Association for Computational Linguistics, Stroudsburg, PA, USA, doi: 10.3115/1219840.1219885, URL `http://dx.doi.org/10.3115/1219840.1219885`.

Aidan Finn and Nicholas Kushmerick (2004), Multi-level Boundary Classification for Information Extraction, in Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi, editors, *ECML*, volume 3201 of *Lecture Notes in Computer Science*, pp. 111–122, Springer, ISBN 3-540-23105-6, URL `http://dx.doi.org/10.1007/978-3-540-30115-8_13`.

A. Frank and A. Asuncion (2010), UCI Machine Learning Repository, URL `http://archive.ics.uci.edu/ml`.

Dayne Freitag and Andrew K. McCallum (1999), Information Extraction with HMMs and Shrinkage, in *AAAI Workshop on Machine Learning for Information Extraction*, URL `https://www.aaai.org/Papers/Workshops/1999/WS-99-11/WS99-11-006.pdf`.

Dayne Brian Freitag (1999), *Machine learning for information extraction in informal domains*, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, USA.

J. Friedman, T. Hastie, and R. Tibshirani (2000), Additive logistic regression: a statistical view of boosting, *Annals of statistics*, 28(2):337–374.

Katrin FUNDEL, Robert KÜFFNER, and Ralf ZIMMER (2007), RelEx—Relation extraction using dependency parse trees, *Bioinformatics*, 23(3):365–371, ISSN 1367-4803, doi:http://dx.doi.org/10.1093/bioinformatics/btl616.

Birte GLIMM, Matthew HORRIDGE, Bijan PARSIA, and Peter F. PATEL-SCHNEIDER (2009), A Syntax for Rules in OWL 2, in *Proceedings of the 6th International Workshop on OWL: Experiences and Directions (OWLED 2009)*, volume 529, CEUR.

Birte GLIMM, Ian HORROCKS, Boris MOTIK, and Giorgos STOILOS (2010), Optimising Ontology Classification, in Peter F. PATEL-SCHNEIDER, Yue PAN, Pascal HITZLER, Peter MIKA, Lei ZHANG, Jeff Z. PAN, Ian HORROCKS, and Birte GLIMM, editors, *Proc. of the 9th Int. Semantic Web Conf. (ISWC 2010)*, volume 6496 of *LNCS*, pp. 225–240, Springer, Shanghai, China.

Yuanbo GUO, Jeff HEFLIN, and Zhengxiang PAN (2003), Benchmarking DAML+OIL Repositories, in Dieter FENSEL, Katia P. SYCARA, and John MYLOPOULOS, editors, *International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pp. 613–627, Springer, ISBN 3-540-20362-1.

Petr HÁJEK (1998), *Metamathematics of Fuzzy Logic*, Kluwer, ISBN 978-0-7923-5238-9.

Jan HAJIČ (2000), Morphological Tagging: Data vs. Dictionaries, in *Proceedings of the 6th Applied Natural Language Processing and the 1st NAACL Conference*, pp. 94–101, Seattle, Washington.

Jan HAJIČ, Eva HAJIČOVÁ, Jaroslava HLAVÁČOVÁ, Václav KLIMEŠ, Jiří MÍROVSKÝ, Petr PAJAS, Jan ŠTĚPÁNEK, Barbora VIDOVÁ-HLADKÁ, and Zdeněk ŽABOKRTSKÝ (2006), Prague Dependency Treebank 2.0 CD–ROM, Linguistic Data Consortium LDC2006T01, Philadelphia 2006, URL `http://ufal.mff.cuni.cz/pdt2.0/`.

Eva HAJIČOVÁ, Zdeněk KIRSCHNER, and Petr SGALL (1999), A Manual for Analytical Layer Annotation of the Prague Dependency Treebank, Technical report, ÚFAL MFF UK, Prague, Czech Republic, URL `http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/en/a-layer/html/index.html`.

Mark HALL, Eibe FRANK, Geoffrey HOLMES, Bernhard PFAHRINGER, Peter REUTEMANN, and Ian H. WITTEN (2009), The WEKA data mining software: an update, *SIGKDD Explor. Newsl.*, 11(1):10–18, ISSN 1931-0145, doi:http://doi.acm.org/10.1145/1656274.1656278.

Martin HEPP (2008), GoodRelations: An Ontology for Describing Products and Services Offers on the Web, in Aldo GANGEMI and Jérôme EUZENAT, editors, *EKAW*, volume 5268 of *Lecture Notes in Computer Science*, pp. 329–346, Springer, ISBN 978-3-540-87695-3.

Tomáš HORVÁTH and Peter VOJTÁŠ (2007), Induction of Fuzzy and Annotated Logic Programs, *ILP: 16th International Conference, ILP 2006, Santiago de Compostela, Spain, August 24-27, 2006, Revised Selected Papers*, pp. 260–274, doi:http://dx.doi.org/10.1007/978-3-540-73847-3_27.

Richard JOHANSSON and Pierre NUGUES (2007), Extended Constituent-to-dependency Conversion for English, in *Proceedings of NODALIDA 2007*, pp. 105–112, Tartu, Estonia.

Markus JUNKER, Michaell SINTEK, Michael SINTEK, and Matthias RINCK (1999), Learning for Text Categorization and Information Extraction with ILP, in *In Proc. Workshop on Learning Language in Logic*, pp. 84–93, Springer, LNCS.

S. S. KEERTHI, S. K. SHEVADE, C. BHATTACHARYYA, and K. R. K. MURTHY (2001), Improvements to Platt's SMO Algorithm for SVM Classifier Design, *Neural Computation*, 13(3):637–649.

Michael KIFER and V. S. SUBRAHMANIAN (1992), Theory of Generalized Annotated Logic Programming and its Applications, *Journal of Logic Programming*, 12:335–367, ISSN 0743-1066, doi:10.1016/0743-1066(92)90007-P, URL http://dl.acm.org/citation.cfm?id=139720.139723.

Václav KLIMEŠ (2006), Transformation-Based Tectogrammatical Analysis of Czech, in Petr SOJKA, Ivan KOPECEK, and Karel PALA, editors, *Proceedings of the 9th international conference on Text, speech and dialogue*, number 4188 in Lecture Notes in Computer Science, pp. 135–142, Springer-Verlag Berlin Heidelberg, ISBN 3-540-39090-1, ISSN 0302-9743, URL http://dx.doi.org/10.1007/11846406_17.

Václav KLIMEŠ (2007), Transformation-based tectogrammatical dependency analysis of English, in *Proceedings of the 10th international conference on Text, speech and dialogue*, Lecture Notes in Computer Science, pp. 15–22, Springer-Verlag, Berlin, Heidelberg, ISBN 3-540-74627-7, 978-3-540-74627-0, URL http://portal.acm.org/citation.cfm?id=1776334.1776341.

Nikolaos KONSTANTINOU, Dimitrios-Emmanuel SPANOS, Periklis STAVROU, and Nikolas MITROU (2010), Technically approaching the semantic web bottleneck, *Int. J. Web Eng. Technol.*, 6:83–111, ISSN 1476-1289, doi:http://dx.doi.org/10.1504/IJWET.2010.034761, URL http://dx.doi.org/10.1504/IJWET.2010.034761.

Stasinos Th. KONSTANTOPOULOS (2003), *Using ILP to Learn Local Linguistic Structures*, Ph.D. thesis, Faculteit der Wiskunde en Natuurwetenschappen, Groningen, URL http://dissertations.ub.rug.nl/faculties/science/2003/s.t.konstantopoulos/.

Wojciech KOTLOWSKI and Roman SLOWINSKI (2009), Rule learning with monotonicity constraints, in Andrea Pohoreckyj DANYLUK, Léon BOTTOU, and Michael L. LITTMAN, editors, *ICML*, volume 382 of *ACM International Conference Proceeding Series*, p. 68, ACM, ISBN 978-1-60558-516-1.

Stanislav KRAJCI, Rastislav LENCSES, and Peter VOJTAS (2004), A comparison of fuzzy and annotated logic programming, *Fuzzy Sets and Systems*, 144:173–192.

Jana KRAVALOVÁ and Zdeněk ŽABOKRTSKÝ (2009), Czech Named Entity Corpus and SVM-based Recognizer, in *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration (NEWS 2009)*, pp. 194–201, Association for Computational Linguistics, Suntec, Singapore, ISBN 978-1-932432-57-2.

Martin LABSKÝ, Vojtěch SVÁTEK, Marek NEKVASIL, and Dušan RAK (2009), The Ex Project: Web Information Extraction Using Extraction Ontologies, in Bettina BERENDT, Dunja MLADENIC, Marco DE GEMMIS, Giovanni SEMERARO, Myra SPILIOPOULOU, Gerd STUMME, Vojtěch SVÁTEK, and Filip ŽELEZNÝ, editors, *Knowledge Discovery*

Enhanced with Semantic and Social Information, volume 220 of *Studies in Computational Intelligence*, pp. 71–88, Springer Berlin / Heidelberg, URL http://dx.doi.org/10.1007/978-3-642-01891-6_5.

D.D. Lewis (1992), *Representation and learning in information retrieval*, Ph.D. thesis, University of Massachusetts.

Yaoyong Li, Kalina Bontcheva, and Hamish Cunningham (2009), Adapting SVM for Data Sparseness and Imbalance: A Case Study on Information Extraction, *Natural Language Engineering*, 15(02):241–271, URL http://journals.cambridge.org/repo_A45LfkBD.

Yaoyong Li, Hugo Zaragoza, Ralf Herbrich, John Shawe-Taylor, and Jaz S. Kandola (2002), The Perceptron Algorithm with Uneven Margins, in *ICML '02: Proceedings of the Nineteenth International Conference on Machine Learning*, pp. 379–386, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, ISBN 1-55860-873-7.

S. Lievens, Bernard De Baets, and Kim Cao-Van (2008), A probabilistic framework for the design of instance-based supervised ranking algorithms in an ordinal setting, *Annals OR*, 163(1):115–142.

Dekang Lin (2003), Dependency-Based Evaluation of Minipar, in Anne Abeillé and Nancy Ide, editors, *Treebanks*, volume 20 of *Text, Speech and Language Technology*, pp. 317–329, Springer Netherlands, ISBN 978-94-010-0201-1, URL http://dx.doi.org/10.1007/978-94-010-0201-1_18.

Bing Liu (2007), *Web Data Mining*, Springer-Verlag, ISBN 978-3-540-37881-5, URL http://dx.doi.org/10.1007/978-3-540-37882-2.

I. Manolescu, L. Afanasiev, A. Arion, J. Dittrich, S. Manegold, N. Polyzotis, K. Schnaitter, P. Senellart, S. Zoupanos, and D. Shasha (2008), The repeatability experiment of SIGMOD 2008, *SIGMOD Rec.*, 37(1):39–45, ISSN 0163-5808, doi:http://doi.acm.org/10.1145/1374780.1374791.

Marie Mikulová, Alevtina Bémová, Jan Hajič, Eva Hajičová, Jiří Havelka, Veronika Kolářová, Lucie Kučová, Markéta Lopatková, Petr Pajas, Jarmila Panevová, Magda Razímová, Petr Sgall, Jan Štěpánek, Zdeňka Urešová, Kateřina Veselá, and Zdeněk Žabokrtský (2006), A Manual for Tectogrammatical Layer Annotation of the Prague Dependency Treebank, Technical Report 30, ÚFAL MFF UK, Prague, Czech Rep., URL http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/en/t-layer/html/index.html.

Jiří Mírovský (2006), Netgraph: A Tool for Searching in Prague Dependency Treebank 2.0, in Jan Hajič and Joakim Nivre, editors, *Proceedings of the Fifth Workshop on Treebanks and Linguistic Theories (TLT)*, 5, pp. 211–222, Prague, Czech rep., ISBN 80-239-8009-2.

Yusuke Miyao, Rune Sætre, Kenji Sagae, Takuya Matsuzaki, and Jun'ichi Tsujii (2008), Task-oriented Evaluation of Syntactic Parsers and Their Representations, in *Proceedings of ACL-08: HLT*, pp. 46–54, Association for Computational Linguistics, Columbus, Ohio, URL http://www.aclweb.org/anthology-new/P/P08/P08-1006.bib.

Boris Motik, Ulrike Sattler, and Rudi Studer (2005), Query Answering for OWL-DL with rules, *Web Semantics: Science, Services and Agents on the World Wide Web*, 3:41–60, ISSN 1570-8268, doi:http://dx.doi.org/10.1016/j.websem.2005.05.001, URL `http://dx.doi.org/10.1016/j.websem.2005.05.001`.

Stephen MUGGLETON (1991), Inductive Logic Programming, *New Generation Computing*, 8(4):295–318, ISSN 0288-3635, URL `http://dx.doi.org/10.1007/BF03037089`.

Stephen MUGGLETON (1995), Inverse Entailment and Progol, *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, URL `http://citeseer.ist.psu.edu/muggleton95inverse.html`.

Stephen MUGGLETON and Luc DE RAEDT (1994), Inductive Logic Programming: Theory and Methods, *Journal of Logic Programming*, 19:629–679.

Claude NADEAU and Yoshua BENGIO (2003), Inference for the Generalization Error, *Mach. Learn.*, 52:239–281, ISSN 0885-6125, doi:10.1023/A:1024068626366, URL `http://portal.acm.org/citation.cfm?id=779909.779927`.

Václav NOVÁK and Zdeněk ŽABOKRTSKY (2007), Feature engineering in maximum spanning tree dependency parser, in *Proceedings of the 10th international conference on Text, speech and dialogue*, Lecture Notes in Computer Science, pp. 92–98, Springer-Verlag, Berlin, Heidelberg, ISBN 3-540-74627-7, 978-3-540-74627-0, URL `http://portal.acm.org/citation.cfm?id=1776334.1776350`.

Petr PAJAS and Jan ŠTĚPÁNEK (2009), System for Querying Syntactically Annotated Corpora, in Gary LEE and Sabine Schulte IM WALDE, editors, *Proceedings of the ACL-IJCNLP 2009 Software Demonstrations*, pp. 33–36, Association for Computational Linguistics, Suntec, Singapore, ISBN 1-932432-61-2.

Karel PALA and Pavel SMRŽ (2004), Building Czech Wordnet, *Romanian Journal of Information Science and Technology*, 2004(7):79–88, URL `http://www.fit.vutbr.cz/research/view_pub.php?id=7682`.

Bijan PARSIA, Evren SIRIN, Bernardo Cuenca GRAU, Edna RUCKHAUS, and Daniel HEWLETT (2005), Cautiously Approaching SWRL, *Elsevier Science*, 2005(February), URL `http://www.mindswap.org/papers/CautiousSWRL.pdf`.

Jan PAVELKA (1979), On fuzzy logic I, II, III, *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 25(3-6):45—52, 119—134, 447—464, ISSN 1521-3870, doi:10.1002/malq.19790250304, URL `http://dx.doi.org/10.1002/malq.19790250304`.

J. Ross QUINLAN (1993), *C4.5: programs for machine learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, ISBN 1-55860-238-0.

Ganesh RAMAKRISHNAN, Sachindra JOSHI, Sreeram BALAKRISHNAN, and Ashwin SRINIVASAN (2007), Using ILP to Construct Features for Information Extraction from Semi-structured Text, in Hendrik BLOCKEEL, Jan RAMON, Jude W. SHAVLIK, and Prasad TADEPALLI, editors, *ILP'07: Proceedings of the 17th International Conference on Inductive Logic Programming*, volume 4894 of *Lecture Notes in Computer Science*, pp. 211–224, Springer-Verlag, Berlin, Heidelberg, ISBN 3-540-78468-3, 978-3-540-78468-5, URL `http://dx.doi.org/10.1007/978-3-642-01891-6_5`.

Marek Reformat, Ronald R. Yager, and Zhan Li (2008), Ontology Enhanced Concept Hierarchies for Text Identification, *Journal Semantic Web Information Systems*, 4(3):16–43, ISSN 1552-6283.

F. Rosenblatt (1957), *The Perceptron, a Perceiving and Recognizing Automaton Project Para*, Cornell Aeronautical Laboratory report, Cornell Aeronautical Laboratory, URL http://books.google.cz/books?id=P_XGPgAACAAJ.

Petr Sgall, Eva Hajičová, and Jarmila Panevová (1986), *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*, Dordrecht:Reidel Publishing Company and Prague:Academia, ISBN 978-90-277-1838-9.

Drahomíra "johanka" Spoustová, Jan Hajič, Jan Votrubec, Pavel Krbec, and Pavel Květoň (2007), The Best of Two Worlds: Cooperation of Statistical and Rule-Based Taggers for Czech, in *Proceedings of the Workshop on Balto-Slavonic Natural Language Processing 2007*, pp. 67–74, Association for Computational Linguistics, Prague, Czech Republic.

Rudi Studer, V. Richard Benjamins, and Dieter Fensel (1998), Knowledge engineering: Principles and methods, *Data & Knowledge Engineering*, 25(1-2):161 – 197, ISSN 0169-023X, doi:DOI:10.1016/S0169-023X(97)00056-6, URL http://www.sciencedirect.com/science/article/B6TYX-3SYXJ6S-G/2/67ea511f5600d90a74999a9fef47ac98.

Jan Votrubec (2006), Morphological Tagging Based on Averaged Perceptron, in *WDS'06 Proceedings of Contributed Papers*, pp. 191–195, Matfyzpress, Charles University, Praha, Czechia, ISBN 80-86732-84-3, URL http://www.mff.cuni.cz/veda/konference/wds/proc/pdf06/WDS06_134_i3_Votrubec.pdf.

Rui Wang and Günter Neumann (2007), Recognizing textual entailment using sentence similarity based on dependency tree skeletons, in *RTE '07: Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pp. 36–41, Association for Computational Linguistics, Morristown, NJ, USA.

Daya C. Wimalasuriya and Dejing Dou (2010), Ontology-based information extraction: An introduction and a survey of current approaches, *Journal of Information Science*, 36(3):306–323, doi:10.1177/0165551509360123, URL http://dx.doi.org/10.1177/0165551509360123.

A. Yakushiji, Y. Tateisi, Y. Miyao, and J. Tsujii (2001), Event extraction from biomedical papers using a full parser., *Pac Symp Biocomput*, pp. 408–419, URL http://view.ncbi.nlm.nih.gov/pubmed/11262959.

Burcu Yildiz and Silvia Miksch (2007), ontoX - a method for ontology-driven information extraction, in *Proceedings of the 2007 international conference on Computational science and its applications - Volume Part III*, ICCSA'07, pp. 660–673, Springer-Verlag, Berlin, Heidelberg, ISBN 3-540-74482-5, 978-3-540-74482-5, URL http://portal.acm.org/citation.cfm?id=1793154.1793216.

Zdeněk Žabokrtský, Jan Ptáček, and Petr Pajas (2008), TectoMT: Highly Modular MT System with Tectogrammatics Used as Transfer Layer, in *Proceedings of the 3rd Workshop on Statistical Machine Translation*, pp. 167–170, ACL, Columbus, OH, USA, ISBN 978-1-932432-09-1.

Dan Zeman, Jiří Hana, Hana Hanová, Jan Hajič, Barbora Hladká, and Emil Jeřábek (2005), A Manual for Morphological Annotation, 2nd edition, Technical Report 27, ÚFAL MFF UK, Prague, Czech Republic, URL `http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/en/m-layer/html/index.html`.

List of Figures

List of Tables

Nomenclature

ANNIE	a Nearly-New Information Extraction System http://gate.ac.uk/userguide/chap:annie
API	Application Programming Interface
CRF	Conditional Random Fields
CRISP	Cross Industry Standard Process (for Data Mining) http://en.wikipedia.org/wiki/CRISP_DM
DL	Description Logic http://en.wikipedia.org/wiki/Description_logic
FGD	Functional Generative Description (of Language), see Section 4.1
FS	Feature Structure http://ufal.mff.cuni.cz/pdt/Corpora/PDT_1.0/Doc/fs.html
GAP	Generalized Annotated Programs, see Section 5.4.3
GATE	General Architecture for Text Engineering, see Section 4.5
GPL	GNU General Public License http://www.gnu.org/licenses/gpl.html
GRDDL	Gleaning Resource Descriptions from Dialects of Languages http://www.w3.org/TR/grddl/
GUI	Graphical User Interface
HMM	Hidden Markov Models
HTML	HyperText Markup Language http://www.w3.org/TR/html
IE	Information Extraction
ILP	Inductive Logic Programming
JAPE	Java Annotation Patterns Engine http://gate.ac.uk/userguide/chap:jape
LDR	Language Dependency Representation
ML	Machine Learning
MST	Minimum-Spanning Tree
OWL	Web Ontology Language http://www.w3.org/TR/owl-primer/
PAUM	Perceptron Algorithm with Uneven Margins
PDT	Prague Dependency Treebank, see Section 4.1

PEDT	Prague English Dependency Treebank
PML	Prague Markup Language `http://ufal.mff.cuni.cz/jazz/PML/`
POS	Part of Speech
PR	Processing Resource (in GATE) `http://gate.ac.uk/userguide/chap:creole-model`
PTB	Penn Treebank `http://www.cis.upenn.edu/~treebank/`
RDF	Resource Description Framework `http://www.w3.org/RDF/`
RDFa	Resource Description Framework - in - attributes `http://www.w3.org/TR/xhtml-rdfa-primer/`
RPC	Remote Procedure Call
RSS	Really Simple Syndication, or Rich Site Summary, but originally RDF Site Summary `http://www.rssboard.org/rss-specification`
SPARQL	SPARQL Query Language for RDF `http://www.w3.org/TR/rdf-sparql-query/`
SQL	Structured Query Language `http://www.iso.org/iso/catalogue_detail.htm?csnumber=45498`
SVM	Support Vector Machines
SWRL	Semantic Web Rule Language `http://www.w3.org/Submission/SWRL/`
URL	Uniform Resource Locator (RFC1738, `http://www.ietf.org/rfc/rfc1738.txt`)
W3C	World Wide Web Consortium `http://www.w3.org/`
XHTML	Extensible HyperText Markup Language `http://www.w3.org/TR/xhtml1/`
XML	Extensible Markup Language `http://www.w3.org/XML/`
XSLT	Extensible Stylesheet Language Transformations `http://www.w3.org/TR/xslt`
ÚFAL	Institute of Formal and Applied Linguistics, Prague, Czech Republic (Ústav formální a aplikované lingvistiky) `http://ufal.mff.cuni.cz/`

A. Listings

Individual listings are placed as separate sections on following pages.

A.1 Sample of Czech WordNet

This sample shows the word coverage in the domain of means of transport and vehicles. We do not provide the complete transcript of Czech words. The top subtree demonstrates the distance between the words “autobus, autokar” (bus, coach) and “kamion” (truck) in the hypernym hierarchy and the rest of the sample lists all hyponyms of the three selected words: “motorové vozidlo” (motor vehicle), “nákladní automobil” (truck) and “auto, vůz” (car).

- entita:1
  - objekt:1
    - celek:1
      - artefakt:1, výtvor:2, výrobek:2
        - vybavení:2
          - přepravní prostředek:1, transportní prostředek:1
            ▷ **veřejná doprava:1**
              ⇒ *autobus:1, autokar:1*
            ▷ **dopravní prostředek:1**
              - kolové vozidlo:1
                - samohybné vozidlo:1, vozidlo s vlastním pohonem:1
                  - motorové vozidlo:1
                    - nákladní automobil:1
                      ⇒ *kamion:1*

- motorové vozidlo:1
  - motocykl:1
  - nákladní automobil:1
  - obojživelné vozidlo:1
  - auto:1, vůz:2
  - pohřební vůz:1
  - sněžný pluh:1, pluh:2
  - golfový vozík:1

- nákladní automobil:1
  - dodávka:3
  - sklápěč:1, vyklápěcí nákladní automobil:1
  - tahač:1
  - pick-up:1, malý nákladní automobil:1
  - hasící vůz:1, požární stříkačka:1
  - rozhlasový vůz:1
  - kamion:1
  - nákladní automobil s přívěsem:1
  - popelářský vůz:1, popelářské auto:1, bobr:3

- auto:1, vůz:2
  - limuzína:1
  - elektrický vozík:1
  - závodní vůz:1, závodní automobil:1
  - sportovní vůz:1
  - kabriolet:1, sporťák:1
  - vrak:3
  - limuzína:2
  - hlídkový vůz:1, policejní vůz:1
  - sériový automobil:1
  - cestovní vůz:1
  - džíp:1
  - kupé:1
  - kabriolet:3
  - kombi:1
  - taxi:1
  - ambulance:1, sanitka:1, pohotovost:6, záchranka:1, sanita:1

A.2 Extraction rules export to OWL in Prolog.

```
serialize_rule_file(RuleFileName, OutputFileName, OntologyURI, ObjectProperties) :-
  assert(objectProperties(ObjectProperties)),
  open(OutputFileName, 'write', Stream), set_output(Stream),
  write('<?xml version="1.0" encoding="UTF-8"?>\n'),
  write('<!DOCTYPE Ontology [ <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >\n'),
  write('  <!ENTITY pml "http://ufal.mff.cuni.cz/pdt/pml/" > ]>\n'),
  write('<Ontology xmlns="http://www.w3.org/2002/07/owl#"\n'),
  format('  ontologyIRI="~a">\n',[OntologyURI]),
  consult(RuleFileName), %Each rule in the file will be processed by serialize_rule,
                         %because of the term_expansion setting on the last line.
  write('</Ontology>'),
  close(Stream).

serialize_rule(:-(H, B)) :-
  numbervars(:-(H, B), 0, _),
  write('<DLSafeRule>\n'),
    write('<Body>\n'),
      serialize_term(B),                    %Rule body
    write('\n</Body>\n'), write('<Head>\n'),
      serialize_term(H),                    %Rule head
    write('\n</Head>\n'),
  write('</DLSafeRule>\n\n'),!.

%Serialize object property
serialize_term(T):- functor(T,F,N), objectProperties(OProps), member(F, OProps),!,
  write('<ObjectPropertyAtom>'),
  format('<ObjectProperty IRI="&pml;~a"/>\n', [F]),
  serialize_arg(T, 0, N),
  write('</ObjectPropertyAtom>').

%Serialize datatype property
serialize_term(T):- functor(T,F,N),
  write('<DataPropertyAtom>'),
  format('<DataProperty IRI="&pml;~a"/>\n', [F]),
  serialize_arg(T, 0, N),
  write('</DataPropertyAtom>').

%Serialize argument list
serialize_term(','(A,B)):- serialize_term(A), write('\n'), serialize_term(B),!.

%Serialize variable
serialize_term('$VAR'(N)):- char_code('a',I), Var is I + N,
  format('<Variable IRI="urn:swrl#~c"/>\n',[Var]),!.
%Serialize simple atom or string (printed without quotes)
serialize_term(T):- atom(T), format('<Literal>~a</Literal>\n',[T]),!.
%Serialize number
serialize_term(T):- simple(T), format('<Literal>~k</Literal>\n',[T]),!.

%Serialize term arguments: serialize_arg(+Term, +CurrentArgNumber, +LastArgNumber)
serialize_arg(T, _, 0).
serialize_arg(T, N, N).
serialize_arg(T, M, N) :-  M2 is M+1, arg(M2, T, A),
  serialize_term(A), serialize_arg(T, M2, N).
                                            %All predicates loaded by consult
term_expansion(T,T):- serialize_rule(T).    %will be processed by serialize_rule.
```

B. Complete Evaluation Results

B.1 Czech Fireman

task	measured quantity		ILP avg	±	ILP stdev	PAUM avg	±	PAUM stdev	st.sig.
cars	Number	Correct	0.563	±	0.732	2.047	±	1.452	○
		Missing	4.531	±	2.569	2.359	±	1.587	●
		Spurious	2.078	±	2.080	2.594	±	1.892	
		Overlap	0.531	±	0.734	1.219	±	1.175	○
	Strict	Precision	0.324	±	0.387	0.380	±	0.249	
		Recall	0.088	±	0.129	0.353	±	0.231	○
		F_1	0.109	±	0.147	0.335	±	0.205	○
	Lenient	F_1	0.209	±	0.199	0.524	±	0.232	○
	Average	F_1	0.159	±	0.161	0.429	±	0.203	○
	Number	Training Inst	39.375	±	3.011		n/a		
		Training Docs	43.750	±	0.436		n/a		
	Time	Training	3:28.0	±	0:28.3	0:02.2	±	0:00.1	●
		Testing	0:00.9	±	0:00.2	0:00.3	±	0:00.1	●
damage	Number	Correct	2.000	±	1.069	2.313	±	1.052	○
		Missing	0.438	±	0.664	0.188	±	0.393	●
		Spurious	0.281	±	0.487	0.609	±	0.809	○
		Overlap	0.063	±	0.302	0.000	±	0.000	
	Strict	Precision	0.901	±	0.178	0.860	±	0.176	
		Recall	0.821	±	0.261	0.933	±	0.148	○
		F_1	0.828	±	0.217	0.876	±	0.131	○
	Lenient	F_1	0.847	±	0.195	0.876	±	0.131	
	Average	F_1	0.838	±	0.200	0.876	±	0.131	
	Number	Training Inst	17.500	±	1.084		n/a		
		Training Docs	43.750	±	0.436		n/a		
	Time	Training	0:23.4	±	0:06.2	0:02.2	±	0:00.1	●
		Testing	0:00.9	±	0:00.1	0:00.3	±	0:00.0	●
end subtree	Number	Correct	1.203	±	1.057	3.109	±	1.554	○
		Missing	3.766	±	1.601	1.578	±	1.124	●
		Spurious	1.063	±	1.082	2.797	±	1.945	○
		Overlap	0.281	±	0.629	0.563	±	0.664	○
	Strict	Precision	0.529	±	0.381	0.499	±	0.242	
		Recall	0.231	±	0.203	0.601	±	0.249	○
		F_1	0.283	±	0.219	0.525	±	0.213	○
	Lenient	F_1	0.343	±	0.244	0.623	±	0.169	○
	Average	F_1	0.313	±	0.220	0.574	±	0.181	○
	Number	Training Inst	36.750	±	1.852		n/a		
		Training Docs	43.750	±	0.436		n/a		
	Time	Training	1:34.2	±	0:14.4	0:02.2	±	0:00.1	●
		Testing	0:00.9	±	0:00.2	0:00.3	±	0:00.0	●

task	measured quantity		ILP			PAUM			st.sig.
			avg		stdev	avg		stdev	
start	Number	Correct	4.750	±	0.992	5.125	±	0.900	○
		Missing	0.500	±	0.617	0.125	±	0.333	●
		Spurious	0.453	±	0.733	3.141	±	2.030	○
		Overlap	0.000	±	0.000	0.000	±	0.000	
	Strict	Precision	0.929	±	0.109	0.651	±	0.152	●
		Recall	0.908	±	0.115	0.978	±	0.058	○
		F_1	0.912	±	0.089	0.771	±	0.111	●
	Lenient	F_1	0.912	±	0.089	0.771	±	0.111	●
	Average	F_1	0.912	±	0.089	0.771	±	0.111	●
	Number	Training Inst	36.750	±	0.926		n/a		
		Training Docs	43.750	±	0.436		n/a		
	Time	Training	0:28.9	±	0:03.1	0:02.2	±	0:00.1	●
		Testing	0:00.9	±	0:00.1	0:00.3	±	0:00.0	●
injuries	Number	Correct	2.031	±	1.501	3.047	±	2.089	○
		Missing	2.094	±	2.129	1.078	±	1.384	●
		Spurious	1.375	±	1.397	4.859	±	2.938	○
		Overlap	0.000	±	0.000	0.000	±	0.000	
	Strict	Precision	0.667	±	0.291	0.398	±	0.205	●
		Recall	0.574	±	0.309	0.814	±	0.224	○
		F_1	0.543	±	0.280	0.498	±	0.204	
	Lenient	F_1	0.543	±	0.280	0.498	±	0.204	
	Average	F_1	0.543	±	0.280	0.498	±	0.204	
	Number	Training Inst	28.875	±	3.129		n/a		
		Training Docs	43.750	±	0.436		n/a		
	Time	Training	1:30.7	±	0:15.0	0:02.2	±	0:00.1	●
		Testing	0:00.9	±	0:00.2	0:00.3	±	0:00.1	●
fatalities	Number	Correct	0.188	±	0.393	0.453	±	0.665	○
		Missing	1.188	±	1.097	0.922	±	1.028	●
		Spurious	0.281	±	0.654	1.688	±	1.479	○
		Overlap	0.000	±	0.000	0.000	±	0.000	
	Strict	Precision	0.814	±	0.379	0.307	±	0.390	●
		Recall	0.388	±	0.449	0.536	±	0.452	○
		F_1	0.306	±	0.420	0.222	±	0.308	
	Lenient	F_1	0.306	±	0.420	0.222	±	0.308	
	Average	F_1	0.306	±	0.420	0.222	±	0.308	
	Number	Training Inst	9.625	±	1.120		n/a		
		Training Docs	43.750	±	0.436		n/a		
	Time	Training	0:49.2	±	0:10.6	0:02.2	±	0:00.1	●
		Testing	0:00.9	±	0:00.1	0:00.3	±	0:00.0	●

task	measured quantity		ILP avg		ILP stdev	PAUM avg		PAUM stdev	st.sig.
professional unit	Number	Correct	4.891	±	2.079	7.719	±	1.830	○
		Missing	2.750	±	1.911	1.234	±	1.581	●
		Spurious	3.641	±	2.615	3.375	±	2.380	
		Overlap	2.109	±	1.404	0.797	±	0.820	●
	Strict	Precision	0.500	±	0.241	0.677	±	0.138	○
		Recall	0.506	±	0.191	0.811	±	0.138	○
		F_1	0.491	±	0.200	0.730	±	0.118	○
	Lenient	F_1	0.692	±	0.116	0.801	±	0.106	○
	Average	F_1	0.591	±	0.151	0.766	±	0.106	○
	Number	Training Inst	92.750	±	3.478		n/a		
		Training Docs	43.750	±	0.436		n/a		
	Time	Training	2:37.4	±	0:27.4	0:02.2	±	0:00.1	●
		Testing	0:00.9	±	0:00.1	0:00.3	±	0:00.0	●
amateur unit	Number	Correct	3.250	±	2.539	3.641	±	2.919	○
		Missing	0.625	±	1.134	0.234	±	0.463	●
		Spurious	0.750	±	1.447	2.625	±	1.777	○
		Overlap	0.000	±	0.000	0.000	±	0.000	
	Strict	Precision	0.863	±	0.256	0.546	±	0.293	●
		Recall	0.886	±	0.210	0.955	±	0.096	○
		F_1	0.827	±	0.253	0.634	±	0.296	●
	Lenient	F_1	0.827	±	0.253	0.634	±	0.296	●
	Average	F_1	0.827	±	0.253	0.634	±	0.296	●
	Number	Training Inst	27.125	±	3.032		n/a		
		Training Docs	43.750	±	0.436		n/a		
	Time	Training	0:24.0	±	0:04.6	0:02.2	±	0:00.1	●
		Testing	0:00.9	±	0:00.1	0:00.3	±	0:00.0	●
overall (all tasks)	Number	Correct	2.359	±	2.214	3.432	±	2.656	○
		Missing	1.986	±	2.172	0.965	±	1.316	●
		Spurious	1.240	±	1.823	2.711	±	2.294	○
		Overlap	0.373	±	0.913	0.322	±	0.713	
	Strict	Precision	0.691	±	0.358	0.540	±	0.297	●
		Recall	0.550	±	0.382	0.748	±	0.312	○
		F_1	0.537	±	0.369	0.574	±	0.295	○
	Lenient	F_1	0.585	±	0.353	0.619	±	0.284	○
	Average	F_1	0.561	±	0.357	0.596	±	0.285	○
	Number	Training Inst	36.094	±	23.599		n/a		
		Training Docs	43.750	±	0.433		n/a		
	Time	Training	1:24.5	±	1:05.8	0:02.2	±	0:00.1	●
		Testing	0:00.9	±	0:00.1	0:00.3	±	0:00.0	●

B.2 Acquisitions

task	measured quantity		ILP avg		ILP stdev	PAUM avg		PAUM stdev	st.sig.
acqabr	Number	Correct	130.500	±	18.295	374.500	±	21.681	○
		Missing	589.300	±	43.071	257.400	±	33.200	●
		Spurious	149.100	±	17.508	449.800	±	25.455	○
		Overlap	5.200	±	1.989	93.100	±	5.259	○
	Strict	Precision	0.457	±	0.034	0.408	±	0.015	●
		Recall	0.180	±	0.025	0.517	±	0.027	○
		F_1	0.258	±	0.030	0.456	±	0.015	○
	Lenient	F_1	0.268	±	0.032	0.569	±	0.009	○
	Average	F_1	0.263	±	0.031	0.513	±	0.011	○
	Number	Training Inst	419.000	±	37.848	n/a			
		Training Docs	300.000	±	0.000	n/a			
	Time	Training	16:24.9	±	8:22.9	0:10.0	±	0:00.6	●
		Testing	0:27.4	±	0:00.8	0:13.6	±	0:00.3	●
dlramt	Number	Correct	39.100	±	6.871	104.700	±	9.068	○
		Missing	50.100	±	6.385	27.700	±	8.731	●
		Spurious	45.800	±	14.906	62.300	±	13.292	○
		Overlap	52.300	±	5.355	9.100	±	2.025	●
	Strict	Precision	0.286	±	0.052	0.597	±	0.030	○
		Recall	0.276	±	0.043	0.740	±	0.063	○
		F_1	0.280	±	0.046	0.659	±	0.022	○
	Lenient	F_1	0.656	±	0.053	0.716	±	0.024	○
	Average	F_1	0.468	±	0.045	0.687	±	0.022	○
	Number	Training Inst	141.500	±	6.587	n/a			
		Training Docs	300.000	±	0.000	n/a			
	Time	Training	23:27.6	±	15:45.7	0:10.8	±	0:02.8	●
		Testing	0:30.5	±	0:00.8	0:09.8	±	0:00.5	●
acquired	Number	Correct	92.700	±	18.013	174.800	±	8.842	○
		Missing	234.000	±	23.580	135.100	±	10.989	●
		Spurious	138.200	±	22.190	191.300	±	22.765	○
		Overlap	14.800	±	4.264	31.600	±	5.522	○
	Strict	Precision	0.376	±	0.028	0.441	±	0.027	○
		Recall	0.272	±	0.054	0.512	±	0.033	○
		F_1	0.313	±	0.046	0.473	±	0.024	○
	Lenient	F_1	0.363	±	0.057	0.559	±	0.019	○
	Average	F_1	0.338	±	0.052	0.516	±	0.020	○
	Number	Training Inst	297.000	±	3.300	n/a			
		Training Docs	300.000	±	0.000	n/a			
	Time	Training	10:16.0	±	7:11.1	0:09.8	±	0:00.4	●
		Testing	0:26.7	±	0:00.7	0:11.8	±	0:00.3	●

task	measured quantity		ILP			PAUM			st.sig.
			avg		stdev	avg		stdev	
purchabr	Number	Correct	75.200	±	24.262	324.400	±	23.604	○
		Missing	548.500	±	53.817	252.600	±	26.517	●
		Spurious	150.800	±	40.157	454.400	±	43.221	○
		Overlap	7.800	±	3.795	54.500	±	6.416	○
	Strict	Precision	0.318	±	0.050	0.390	±	0.015	○
		Recall	0.120	±	0.041	0.514	±	0.033	○
		F_1	0.172	±	0.050	0.443	±	0.016	○
	Lenient	F_1	0.190	±	0.055	0.517	±	0.018	○
	Average	F_1	0.181	±	0.052	0.480	±	0.016	○
	Number	Training Inst	413.000	±	29.885		n/a		
		Training Docs	300.000	±	0.000		n/a		
	Time	Training	16:51.6	±	9:14.7	0:10.2	±	0:01.4	●
		Testing	0:27.4	±	0:00.9	0:13.2	±	0:00.5	●
purchaser	Number	Correct	105.900	±	19.490	173.300	±	9.557	○
		Missing	194.400	±	26.421	119.400	±	11.394	●
		Spurious	144.100	±	21.850	174.000	±	19.743	○
		Overlap	11.700	±	4.547	19.300	±	3.164	○
	Strict	Precision	0.403	±	0.037	0.474	±	0.029	○
		Recall	0.340	±	0.064	0.556	±	0.028	○
		F_1	0.367	±	0.051	0.511	±	0.022	○
	Lenient	F_1	0.408	±	0.061	0.568	±	0.018	○
	Average	F_1	0.387	±	0.056	0.539	±	0.020	○
	Number	Training Inst	298.500	±	11.787		n/a		
		Training Docs	300.000	±	0.000		n/a		
	Time	Training	12:27.1	±	7:22.6	0:10.4	±	0:01.2	●
		Testing	0:27.5	±	0:02.0	0:11.5	±	0:01.2	●
seller	Number	Correct	17.400	±	5.929	29.800	±	4.940	○
		Missing	113.000	±	12.257	99.200	±	14.421	●
		Spurious	46.100	±	16.954	89.000	±	19.050	○
		Overlap	3.100	±	1.197	4.500	±	1.269	
	Strict	Precision	0.261	±	0.071	0.245	±	0.041	
		Recall	0.131	±	0.047	0.226	±	0.046	○
		F_1	0.170	±	0.050	0.232	±	0.036	○
	Lenient	F_1	0.201	±	0.056	0.267	±	0.039	○
	Average	F_1	0.186	±	0.053	0.250	±	0.037	○
	Number	Training Inst	127.500	±	13.260		n/a		
		Training Docs	300.000	±	0.000		n/a		
	Time	Training	9:12.2	±	3:55.2	0:10.1	±	0:01.0	●
		Testing	0:27.3	±	0:03.2	0:10.7	±	0:00.8	●

task	measured quantity		ILP avg		ILP stdev	PAUM avg		PAUM stdev	st.sig.
sellerabr	Number	Correct	10.800	±	6.143	40.900	±	7.460	○
		Missing	203.800	±	21.928	165.800	±	21.765	●
		Spurious	22.500	±	7.778	137.400	±	16.801	○
		Overlap	0.900	±	0.876	8.800	±	2.573	○
	Strict	Precision	0.306	±	0.111	0.219	±	0.036	●
		Recall	0.050	±	0.027	0.190	±	0.032	○
		F_1	0.085	±	0.044	0.202	±	0.030	○
	Lenient	F_1	0.092	±	0.042	0.247	±	0.027	○
	Average	F_1	0.089	±	0.043	0.225	±	0.027	○
	Number	Training Inst	120.500	±	20.684	n/a			
		Training Docs	300.000	±	0.000	n/a			
	Time	Training	6:11.5	±	4:03.3	0:09.7	±	0:00.3	●
		Testing	0:26.1	±	0:00.7	0:11.5	±	0:00.3	●
overall (all tasks)	Number	Correct	67.371	±	45.432	174.629	±	124.734	○
		Missing	276.157	±	197.689	151.029	±	79.430	●
		Spurious	99.514	±	58.180	222.600	±	153.848	○
		Overlap	13.686	±	16.844	31.557	±	30.284	○
	Strict	Precision	0.344	±	0.088	0.396	±	0.125	○
		Recall	0.196	±	0.106	0.465	±	0.184	○
		F_1	0.235	±	0.101	0.425	±	0.150	○
	Lenient	F_1	0.311	±	0.180	0.492	±	0.162	○
	Average	F_1	0.273	±	0.132	0.459	±	0.156	○
	Number	Training Inst	259.571	±	123.599	n/a			
		Training Docs	300.000	±	0.000	n/a			
	Time	Training	13:33.0	±	10:00.1	0:10.1	±	0:01.3	●
		Testing	0:27.6	±	0:02.0	0:11.7	±	0:01.4	●

C. List of Author's Publications

Jan Dědek, Alan Eckhardt, Leo Galamboš, and Peter Vojtáš. Discussion on uncertainty ontology for annotation and reasoning (a position paper). In P. C. G. da Costa, editor, *URSW '08 Uncertainty Reasoning for the Semantic Web - Volume 4*, volume 423 of *CEUR Workshop Proceedings*, pages 128–129. The 7th International Semantic Web Conference, 2008. Available from: `http://c4i.gmu.edu/ursw/2008/papers/URSW2008_P2_DedekEtAl.pdf`.

Jan Dědek, Alan Eckhardt, and Peter Vojtáš. Experiments with czech linguistic data and ILP. In Filip Železný and Nada Lavrač, editors, *ILP 2008 - Inductive Logic Programming (Late Breaking Papers)*, pages 20–25, Prague, Czech Republic, 2008. Action M. ISBN 978-80-86742-26-7. Available from: `http://ida.felk.cvut.cz/ilp2008/ILP08_Late_Breaking_Papers.pdf`.

Jan Dědek. Extraction of semantic information from web resources. In Jana Šafránková and Jiří Pavlů, editors, *WDS'08 Proceedings of Contributed Papers: Part I - Mathematics and Computer Sciences*, pages 224–229, Praha, 2008. Matfyzpress. ISBN 978-80-7378-065-4. Available from: `http://www.mff.cuni.cz/veda/konference/wds/proc/pdf08/WDS08_138_i2_Dedek.pdf`.

Jan Dědek, Alan Eckhardt, Peter Vojtáš, and Leo Galamboš. Sémantický web. In Václav Řepa and Oleg Svatoš, editors, *DATAKON 2008*, pages 12–30, Brno, 2008. ISBN 978-80-7355-081-3. Available from: `http://www.datakon.cz/datakon08/zvane.html#5`.

Jan Dědek and Peter Vojtáš. Computing aggregations from linguistic web resources: a case study in czech republic sector/traffic accidents. In Cosmin Dini, editor, *Second International Conference on Advanced Engineering Computing and Applications in Sciences*, pages 7–12. IEEE Computer Society, 2008. ISBN 978-0-7695-3369-8. Available from: `http://dx.doi.org/10.1109/ADVCOMP.2008.17`.

Jan Dědek and Peter Vojtáš. Exploitation of linguistic tools in semantic extraction - a design. In Mieczysław Kłopotek, Adam Przepiórkowski, Sławomir Wierzchoń, and Krzysztof Trojanowski, editors, *Intelligent Information Systems XVI*, pages 239–247, Zakopane, Poland, 2008. Academic Publishing House EXIT. ISBN 978-83-60434-44-4. Available from: `http://iis.ipipan.waw.pl/2008/proceedings/iis08-23.pdf`.

Jan Dědek and Peter Vojtáš. Extrakce informací z textově orientovaných zdrojů webu. In Václav Snášel, editor, *Znalosti 2008*, pages 331–334, 2008. ISBN 978-80-227-2827-0. Available from: `http://znalosti2008.fiit.stuba.sk/download/articles/znalosti2008-Dedek.pdf`.

Jan Dědek and Peter Vojtáš. Linguistic extraction for semantic annotation. In Costin Badica, Giuseppe Mangioni, Vincenza Carchiolo, and Dumitru Burdescu, editors, *2nd International Symposium on Intelligent Distributed Computing*, volume 162 of *Studies in Computational Intelligence*, pages 85–94, Catania, Italy, 2008. Springer-Verlag. ISBN 978-3-540-85256-8. Available from: `http://www.springerlink.com/content/w7213j007t416132`.

Jan Dědek, Alan Eckhardt, and Peter Vojtáš. Connecting web and users. In Štuller J., Linková Z., and Kuželová D., editors, *Inteligentní modely, algoritmy, metody a nástroje pro vytváření sémantického webu*, pages 39–44, Prague, 2008. Ústav informatiky AV ČR.

Jan Dědek and Peter Vojtáš. Web information extraction for e-environment. In Jiri Hrebicek, Jiri Hradec, Emil Pelikan, Ondrej Mirovsky, Werner Pillmann, Ivan Holoubek, and Thomas Bandholtz, editors, *Proceedings of the European conference of the Czech Presidency of the Council of the EU TOWARDS eENVIRONMENT*, pages 138–143, Brno, Czech Republic, 2009. Masaryk University. ISBN 978-80-210-4824-9. Available from: `http://www.e-envi2009.org/?proceedings`.

Jan Dědek and Peter Vojtáš. Fuzzy classification of web reports with linguistic text mining. In Paolo Boldi, Giuseppe Vizzari, Gabriella Pasi, and Ricardo Baeza-Yates, editors, *Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, volume 3, pages 167–170, Los Alamitos, CA, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3801-3. Available from: `http://dx.doi.org/10.1109/WI-IAT.2009.254`.

Jan Dědek. Web information extraction systems for web semantization. In Peter Vojtáš, editor, *ITAT 2009 Information Technologies - Applications and Theory*, pages 1–6, Seňa, Slovakia, 2009. PONT Slovakia. ISBN 978-80-970179-2-7. Available from: `http://ceur-ws.org/Vol-584/paper1.pdf`.

Jan Dědek, Alan Eckhardt, and Peter Vojtáš. Web semantization – design and principles. In Václav Snášel, Piotr Szczepaniak, Ajith P. Abraham, and Janusz Kacprzyk, editors, *Proceedings of the 6th Atlantic Web Intelligence Conference - AWIC'2009*, volume 67/2009 of *Advances in Intelligent and Soft Computing*, pages 3–18, Prague, Czech Rep., 2009. Springer Verlag. ISBN 978-3-642-10686-6. Available from: `http://www.springerlink.com/content/514p404827328j55/`.

Jan Dědek, Peter Vojtáš, and Marta Vomlelová. Evaluace fuzzy ILP klasifikátoru na datech o dopravních nehodách. In Pavel Smrž, editor, *Znalosti 2010*, pages 187–190, Jindřichův Hradec, 2010. VŠE v Praze, Oeconomica. ISBN 978-80-245-1636-3.

Jan Dědek, Peter Vojtáš, and Juraj Vojtáš. Obsahuje web indikace blížící se krize? Umíme je rozeznat? In Jiří Voříšek, editor, *Systémová integrace 2010*, pages 166–175, Praha, 2010. VŠE v Praze, Oeconomica. ISBN 978-80-245-1660-8.

Jan Dědek. Towards semantic annotation supported by dependency linguistics and ILP. In *Proceedings of the 9th International Semantic Web Conference (ISWC2010), Part II*, volume 6497 of *Lecture Notes in Computer Science*, pages 297–304, Shanghai / China, 2010. Springer-Verlag Berlin Heidelberg. ISBN 978-3-642-17748-4. Available from: `http://iswc2010.semanticweb.org/accepted-papers/219`.

Jan Dědek, Peter Vojtáš, and Marta Vomlelová. Fuzzy ILP classification of web reports after linguistic text mining. *Information Processing & Management*, 48(3):438 – 450, 2012. Soft Approaches to IA on the Web. Available from: `http://www.sciencedirect.com/science/article/pii/S0306457311000264`.

Jan Dědek and Peter Vojtáš. Semantic annotation semantically: Using a shareable extraction ontology and a reasoner. In Pascal Lorenz and Eckhard Ammann, editors,