Chapter 1

Introduction

1.1 Introduction

The vast amount of information on the web increases the need of automated processing. Machine processing and machine understanding of textual information is especially difficult.

Figure 1.1 learning and annotation process in general (ILP version later, Chapter 6)

In Figure 1.4, nodes of the tectogrammatical tree are decorated. A piece of information about the damage of 8000 CZK can be found there (the three nodes on the right). We have used ILP to learn rules that are able to detect these nodes.

The extraction process requires human assistance when annotating the training data.

Note that our method is general and is not limited to Czech. It can be used with any structured linguistic representation.

1.2 Web Information Extraction Systems for Web Semantization

There exist many extraction tools that can process web pages and produce structured machine understandable data (or information) that corresponds with the content of a web page. This process is often called Web Information Extraction (WIE).

In this chapter we present a survey of web information extraction systems and we connect these systems with the problem of web semantization.

The chapter is structured as follows. First we sketch the basic ideas of semantic web and web semantization. In the next two sections methods of web information extraction will presented. Then description of our solutions (work in progress) will continue. And finally just before the conclusion we will discuss the connection of WIE systems with the problem of web semantization.

1.2.1 The Semantic Web in Use

The idea of the Semantic Web [Berners-Lee *et al.*, 2001] (World Wide Web dedicated not only to human but also to machine – software agents) is very well known today. Let us just shortly demonstrate its use with respect to the idea of Web Semantization (see in next section).

The Figure 1.5 shows a human user using the (Semantic) Web in three manners: a keyword query, a semantic query and by using a software agent. The difference between the first two manners (keyword and semantic query) can be illustrated with the question: "Give me a list of the names of E.U. heads of state." This example from interesting article [Horrocks, 2008] by Ian Horrocks shows the big difference between use of a semantic query language instead of keywords. In the semantic case you should be given exactly the list of names you were requesting without having to pore through results of (probably more then one) keyword

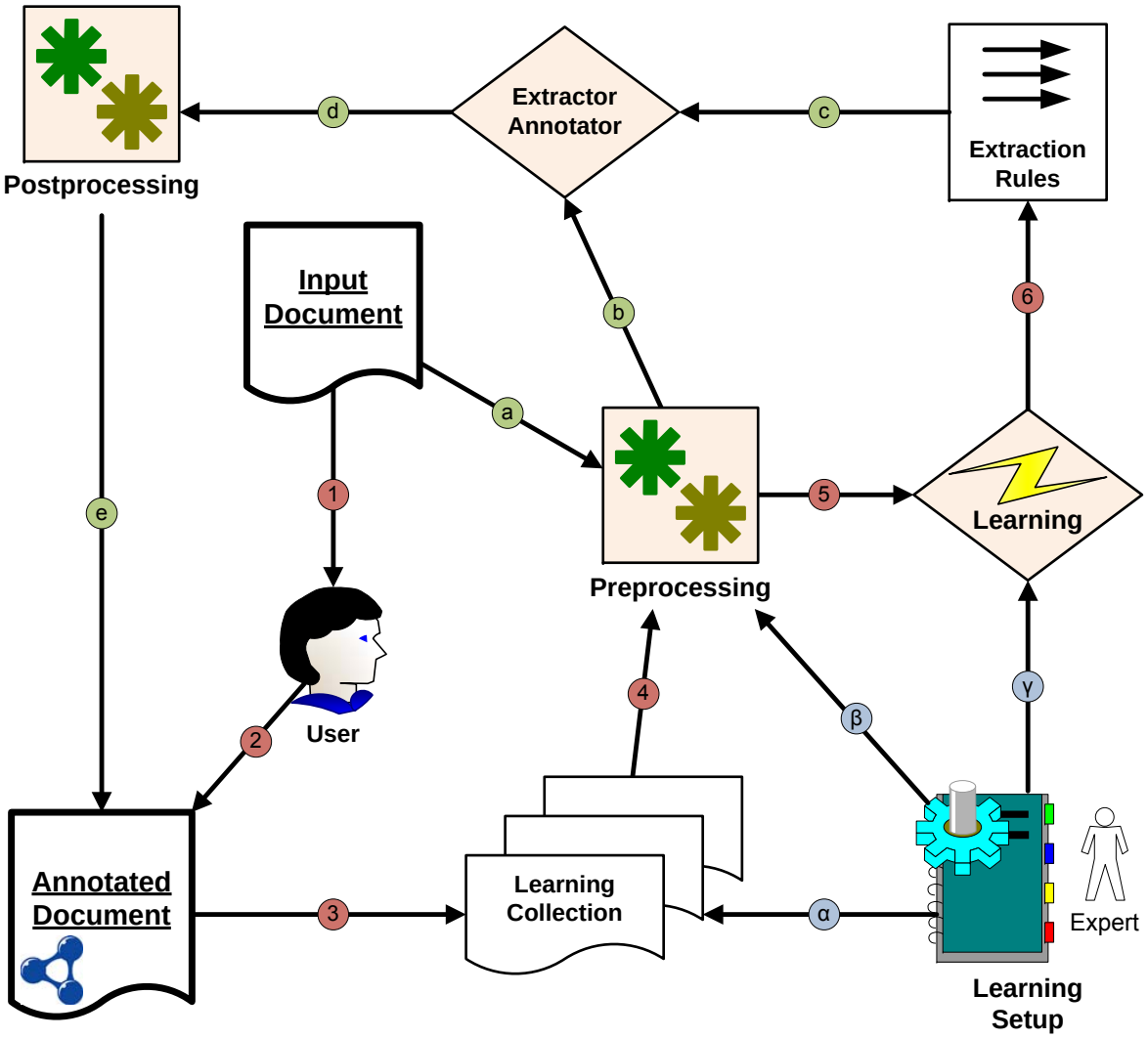


Figure 1.1: Learning and annotation process – general overview

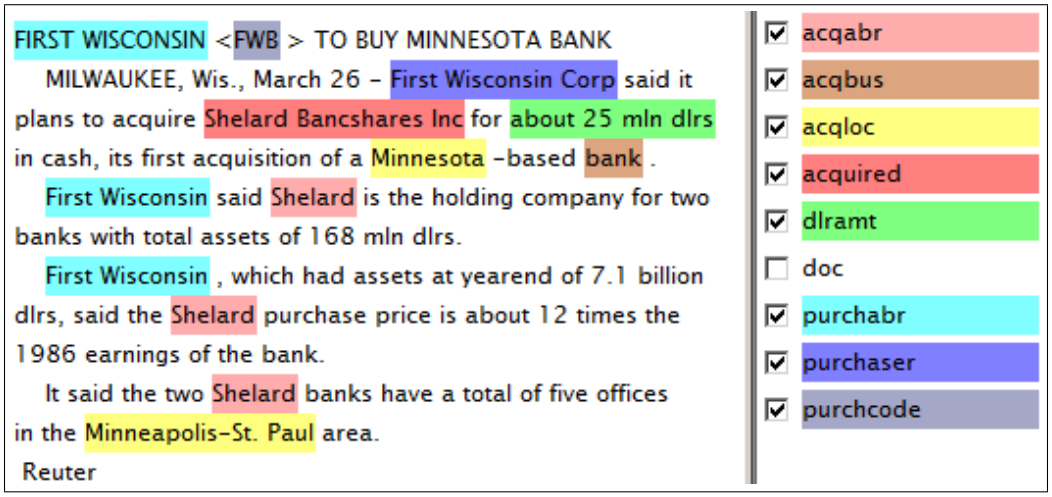


Figure 1.2: Corporate Acquisition Events annotations

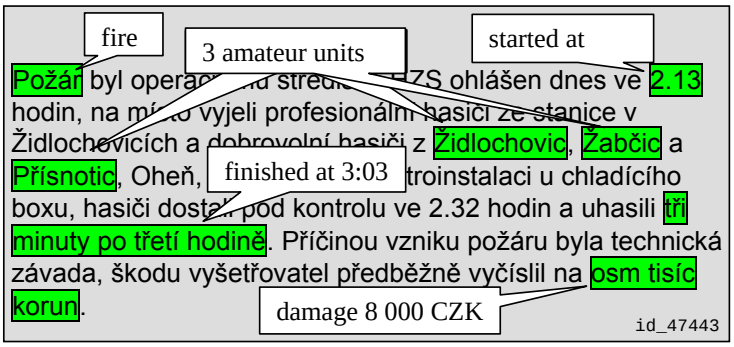


Figure 1.3: Fireman Events annotations

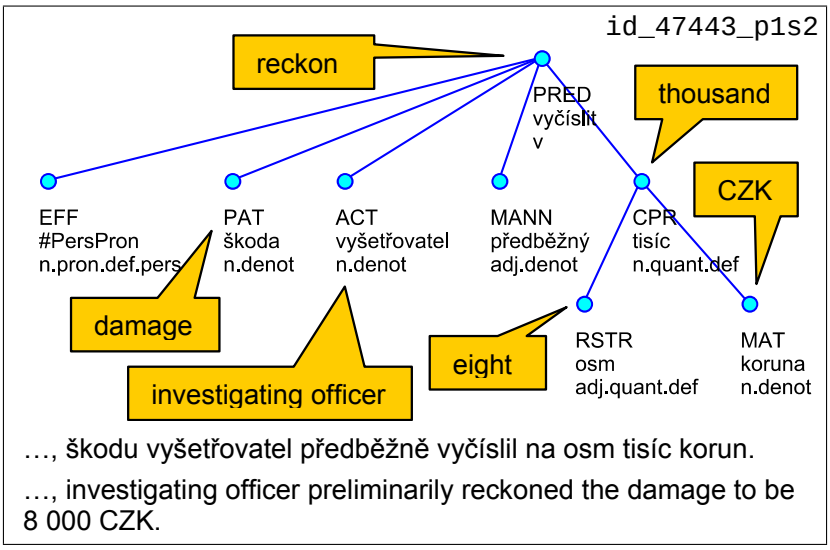


Figure 1.4: Example of a linguistic tree of one analyzed sentence.

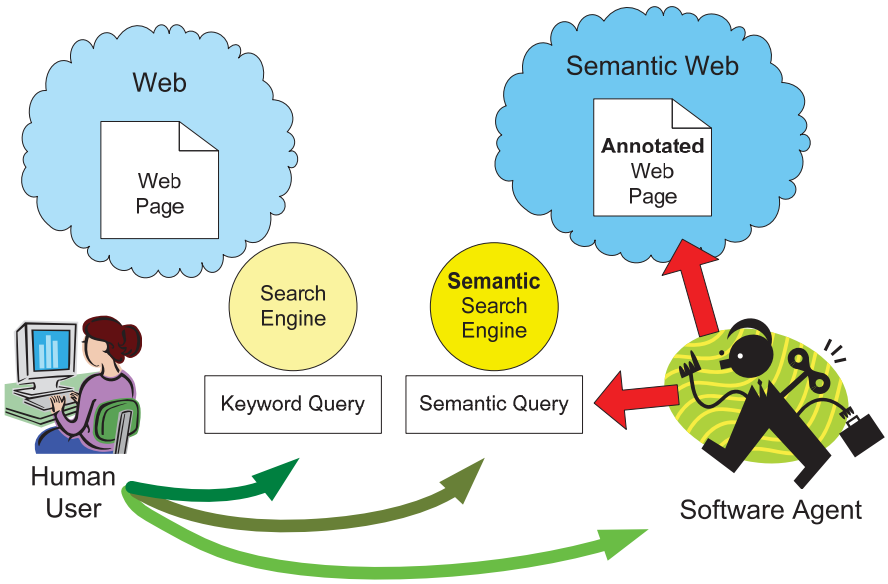


Figure 1.5: The Semantic/Semantized Web in Use

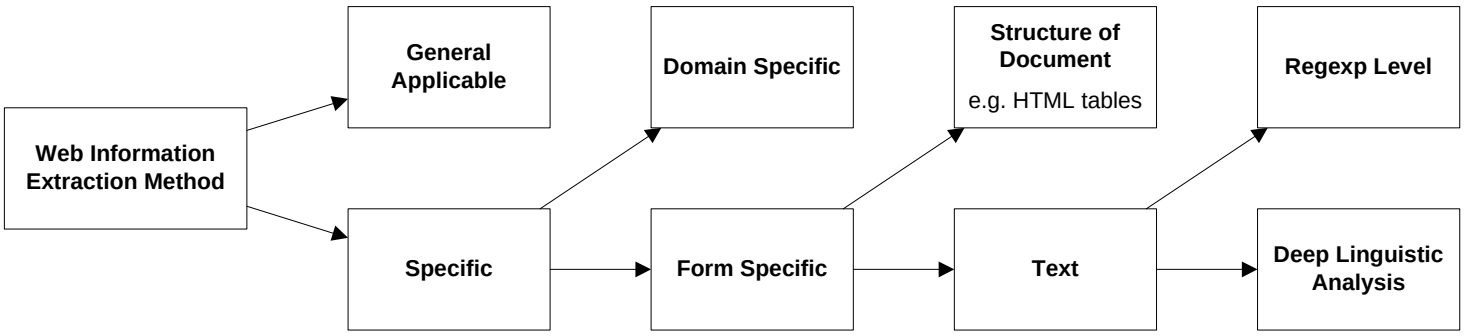


Figure 1.6: Division of extraction methods

queries. Of course the user have to know the syntax of the semantic query language or a special GUI[1] must be provided.

The last and the most important possibility (in the semantic or semantized setting) is to use some (personalized) software agent that is specialized to tasks of some kind like planning a business trip or finding the most optimal choice from all the relevant job offers, flats for rent, cars for sale, etc.

Both the semantic querying and software agents engagement is impossible without some kind of adaptation of the web of today in the semantic direction.

1.2.2 Web Semantization

The idea of Web Semantization [Dědek *et al.*, 2008b] consist in gradual enrichment of the current web content as an automated process of third party annotation tools making at least a part of today's web more suitable for machine processing and hence enabling it intelligent tools for searching and recommending things on the web (see [Berners-Lee, 2008]).

The most strait forward idea is to fill a semantic repository with some information that is automatically extracted from the web and make it available to software agents so they could access to the web of today in a semantic manner (e.g. through semantic search engine).

The idea of a semantic repository and a public service providing semantic annotations was experimentally realized in the very recognized work of IBM Almaden Research Center: the SemTag [Dill *et al.*, 2003]. This work demonstrated that an automated semantic annotation can be applied in a large scale. In their experiment they annotated about 264 million web pages and generated about 434 millions of semantic tags. They also provided the annotations as a *Semantic Label Bureau* – a HTTP server providing annotations for web documents of 3rd parties.

1.3 Web Information Extraction

The task of a web information extraction system is to transform web pages into machine-friendly structures such as relational databases. There exists a rich variety of Web Information Extraction systems. The results generated by distinct tools usually can not be directly compared since the addressed extraction tasks are different. The extraction tasks can be distinguished according several dimensions: domain specificity, automation degree, techniques used, etc. These dimensions are analyzed in detail in the recent publications [Chang *et al.*, 2006a] and [Liu, 2007]. Here we will concentrate on a little bit more specific division of WIE according to the needs of the Web Semantization (see in Sect. 1.6). The division is demonstrated on the Figure 1.6 and should not be considered as disjoint division of the methods but rather as emphasizing different aspects of the methods. For example many extraction methods are domain and form specific at the same time.

[1]Such handy GUI can be found for example in the KIM project [Popov *et al.*, 2004].

The distinguishing between general applicable methods and the others that have meaningful application only in some specific setting (specific domain, specific form of input) is very important for Web Semantization because when we try to produce annotations in large scale, we have to control which web resource is suitable for which processing method (see in Sect. 1.6).

1.3.1 General Applicable

The most significant (and probably the only one) generally applicable IE task is so called *Instance Resolution Task*. The task can be described as follows: Given a general ontology, find all the instances from the ontology that are present in the processed resource. This task is usually realized in two steps: (1) Named Entity Recognition (see in Sect. 1.4.1), (2) Disambiguation of ontology instances that can be connected with the found named entities. Success of the method can be strongly improved with coreference resolution (see in Sect. 1.4.1).

Let us mention several good representatives of this approach: the SemTag application [Dill *et al.*, 2003], the KIM project [Popov *et al.*, 2004] and the PANKOW annotation method [Cimiano *et al.*, 2004] based on smart formulation of Google API queries.

1.3.2 Domain Specific

Domain and from specific IE approaches are the typical cases. More specific information is more precise, more complex and so more useful and interesting. But the extraction method has to be trained to each new domain separately. This usually means indispensable effort.

A good example of domain specific information extraction system is SOBA [Buitelaar *et al.*, 2008]. This complex system is capable to integrate different IE approaches and extract information from heterogeneous data resources, including plain text, tables and image captions but the whole system is concentrated on the single domain of football. Next similarly complex system is ArtEquAKT [Alani *et al.*, 2003], which is entirely concentrated on the domain of art.

1.3.3 Form Specific

Beyond general applicable extraction methods there exist many methods that exploit specific form of the input resource. The linguistic approaches usually process text consisting of natural language sentences. The structure-oriented approaches can be strictly oriented on tables [Pinto *et al.*, 2003] or exploit repetitions of structural patterns on the web page [Zhao *et al.*, 2005] (such algorithm can be only applicable to pages that contain more than one data record), and there are also approaches that use the structure of whole site (e.g. site of single web shop with summary pages with products connected with links to pages with details about single product) [Lerman *et al.*, 2004].

1.4 Information Extraction from Text-based Resources

In this section we will discuss the information extraction from textual resources.

1.4.1 Tasks of Information Extraction

There are classical tasks of text preprocessing and linguistic analysis like

Text Extraction – e.g from HTML, PDF or DOC,

Tokenization – detection of words, spaces, punctuations, etc.,

Segmentation – sentence and paragraph detection,

POS Tagging – part of speech assignment, often including lemmatization and morphological analysis,

Syntactic Analysis (often called linguistic *parsing*) – assignment of the grammatical structure to given sentence with respect to given linguistic formalism (e.g. formal grammar),

Coreference Resolution (or *anaphora resolution*) – resolving what a pronoun, or a noun phrase refers to. These references often cross boundaries of a single sentence.

Besides these classical general applicable tasks, there are further well defined tasks, which are more closely related to the information extraction. These tasks are domain dependent. These tasks were widely developed in the MUC-6 conference 1995 [Grishman and Sundheim, 1996] and considered as semantic evaluation in the first place. These information extraction tasks are:

Named Entity Recognition: This task recognizes and classifies named entities such as persons, locations, date or time expression, or measuring units. More complex patterns may also be recognized as structured entities such as addresses.

Template Element Construction: Populates templates describing entities with extracted roles (or attributes) about one single entity. This task is often performed stepwise sentence by sentence, which results in a huge set of partially filled templates.

Template Relation Construction: As each template describes information about one single entity, this tasks identifies semantic relations between entities.

Template Unification: Merges multiple elementary templates that are filled with information about identical entities.

Scenario Template Production: Fits the results of Template Element Construction and Template Relation Construction into templates describing pre-specified event scenarios (pre-specified "queries on the extracted data").

Appelt and Israel [1999] wrote an excellent tutorial summarizing these traditional IE tasks and systems built on them.

Entity Recognition

Relation Extraction

Event Extraction

1.4.2 Information Extraction Benchmarks

Contrary to the WIE methods based on the web page structure, where we (the authors) do not know about any well established benchmark for these methods[2], the situation in the domain of text based IE is fairly different. There are several conferences and events concentrated on the support of automatic machine processing and understanding of human language in text form. Different research topics as text (or information) retrieval[3], text summarization[4] are involved.

On the filed of information extraction, we have to mention the long tradition of the Message Understand-

[2]It is probably at least partially caused by the vital development of the presentation techniques on the web that is still well in progress.

[3]e.g. Text REtrieval Conference (TREC)
`http://trec.nist.gov/`

[4]e.g. Document Understanding Conferences
`http://duc.nist.gov/`

ing Conference[5] [Grishman and Sundheim, 1996] starting in 1987. In 1999 the event of *Automatic Content Extraction (ACE) Evaluation*[6] started, which is becoming a track in the Text Analysis Conference (TAC)[7] this year (in 2009).

All these events prepare several specialized datasets together with information extraction tasks and play an important role as information extraction benchmarks.

1.5 Our Solutions

1.5.1 Linguistic Information Extraction

Our second approach [Dědek and Vojtáš, 2008b,c,a] for the web information extraction is based on deep linguistic analysis. We have developed a rule-based method for extraction of information from text-based web resources in Czech and now we are working on its adaptation to English. The extraction rules correspond to tree queries on linguistic (syntactic) trees made form particular sentences. We have experimented with several linguistic tools for Czech, namely Tools for machine annotation – PDT 2.0 and the Czech WordNet.

Our present system captures text of web-pages, annotates it linguistically by PDT tools, extracts data and stores the data in an ontology. We have made initial experiments in the domain of reports of traffic accidents. The results showed that this method can e.g. aid summarization of the number of injured people.

To avoid the need of manual design of extraction rules we focused on the data extraction phase and made some promising experiments [Dědek *et al.*, 2008a] with the machine learning procedure of Inductive Logic Programming for automated learning of the extraction rules.

This solution is directed to extraction of information which is closely connected with the meaning of text or meaning of a sentence.

1.6 The Web Semantization Setting

In this section we will discuss possibilities and obstructions connected with the employment of web information extraction systems in the process of web semantization.

One aspect of the realization of the web semantization idea is the problem of integration of all the components and technologies starting with web crawling, going through numerous complex analyses (document preprocessing, document classification, different extraction procedures), output data integration and indexing, and finally implementation of query and presentation interface. This elaborate task is neither easy nor simple but today it is solved in all the extensive projects and systems mentioned above.

The novelty that web semantization brings into account is the cross domain aspect. If we do not want to stay with just general ontologies and general applicable extraction methods then we need a methodology how to deal with different domains. The system has to support extension to a new domain in generic way. So we need a methodology and software to support this action. This can for example mean: to add a new ontology for the new domain, to select and train proper extractors and classifiers for the suitable input pages.

1.6.1 User Initiative and Effort

An interesting point is the question: Whose effort will be used in the process of supporting new domain in the web semantization process? How skilled such user has to be? There are two possibilities (demonstrated on the Fig 1.7). The easier one is that we have to employ very experienced expert who will decide about the new domain and who will also realize the support needed for the new domain. In the Fig 1.7 this situation is labeled as *Provider Initiated* and *Provider Trained* because the expert works on the side of the system that

[5] Briefly summarized in `http://en.wikipedia.org/wiki/Message_Understanding_Conference`.
[6] `http://www.itl.nist.gov/iad/mig/tests/ace/`
[7] `http://www.nist.gov/tac`

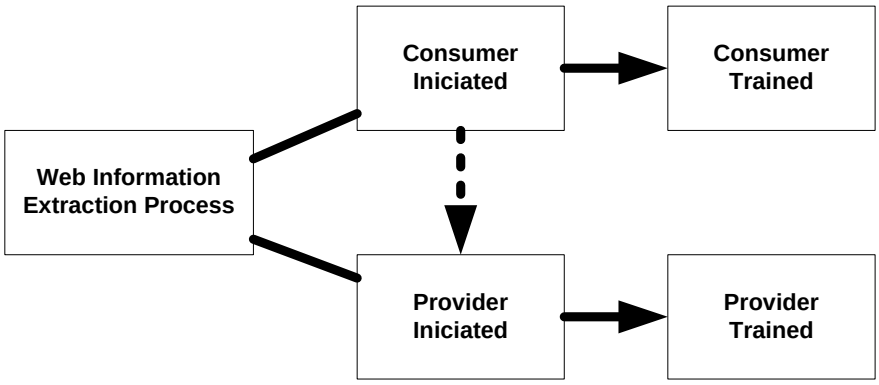


Figure 1.7: User initiative and effort

provides the semantics.

The second possibility is to enable ordinary users form outside to add a new domain to the semantization system. Such user is probably interested in semantic data form the domain so we call such user *Consumer*.

A cooperation of a consumer (possibly a domain expert) and a provider (system expert) on the support of the new domain can be considered as hybrid approach. This is represented with the dashed arrow in the Fig 1.7.

1.7 Conclusion and Future Work

In this chapter we tried to show the complexity of the problem of web semantization in connection with the possibilities of web information extraction systems. Future work goes in several directions:

- Future development of WIE tools and work on their adaptability to new domains.
- Integration of WIE tools to the web semantization system.
- Development of the methodology and software to support the extension of the semantization system to a new domain for a non-expert user.

Chapter 2

Related Work

2.1 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].

2.1.1 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.

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].

2.2 Based on ILP

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.*, 2008] 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.

2.3 Based on Dependency Linguistics

As stated in [Bunescu, 2007], 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. But in our case deep linguistic parsing plays an essential role.

There are other approaches that use deep parsing, but they often use the syntactic structure only for relation extraction and either do not use machine learning at all (extraction rules have to be handcrafted) [Yakushiji *et al.*, 2001], [Fundel *et al.*, 2007], [Buyko *et al.*, 2009] or do some kind of similarity search based on the syntactic structure [Etzioni *et al.*, 2008], [Wang and Neumann, 2007] or the syntactic structure plays only very specific role in the process of feature selection for propositional learners [Bunescu and Mooney, 2007].

2.4 Based on Propositional Machine Learning

2.4.1 GATE Machine Learning

There is also a long row of information extraction approaches that use classical propositional learners like SVM on a set of features manually selected from input text. We do not cite them here. We just refer to [Li *et al.*, 2009] – using machine learning facilities in GATE. This is the software component (Machine Learning PR) to that we have compared our solution.

2.5 Semantic annotation

2.5.1 GATE

Last category of related works goes in the direction of semantics and ontologies. Because we do not develop this topic in this paper, we just refer to the ontology features in GATE [Bontcheva *et al.*, 2004], which can be easily used to populate an ontology with the extracted data. We discus this topic later in Section 6.3.7.

Chapter 3

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.

3.1 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 [Sgall *et al.*, 1986]. 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.

3.1.1 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 3.1. 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 their "targets''). Also some nodes

[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`

without a morphological level counterpart are inserted to the tectogrammatical tree at certain occasions (e.g. a node representing omitted subject – on Figure 3.1 labeled as "#PersPron'').

More over all the layers of PDT sentence representation 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]

3.1.2 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.

3.2 PDT Tools and Resources

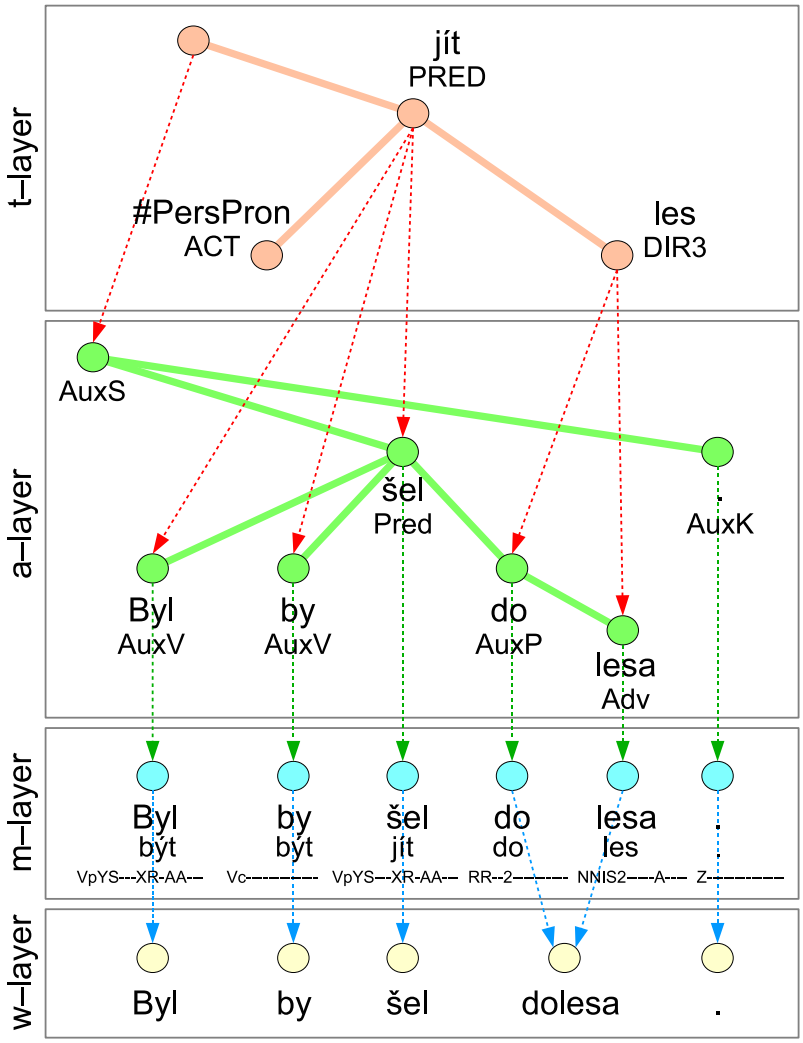
In this section several linguistic tools and resources that are being developed at ÚFAL will be described. These tools and resources are closely connected with the PDT projects and they have been used also in the present work for various purposes.

3.2.1 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.



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

Figure 3.1: Layers of linguistic annotation in PDT

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 3.2.3) and GATE (see Section 3.4). 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 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.

3.2.2 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

[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/`

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/`

3.2.3 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 6.3.1.

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/`

3.2.4 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 5.4.2, it provides additional information about the query language including example Netgraph queries.

Currently Netgraph is replaced by PML Tree Query (see Section 3.2.2) and the Netgraph development is "discontinued". We use Netgraph in the present work because it is written in Java, the language of what we use. This also made it possible to use Netgraph inside GATE as a handy viewer of dependency trees, see Section 6.3.2.

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

3.2.5 Annotation Schemas

Prague Markup Language (PML)

Feature Structure (FS)

3.3 Czech WordNet

The Figure 99 shows, that it would be useful to gather words with similar meanings in our extraction rules. For example, the rule in the Figure 99 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. 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`

3.4 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/`

3.4.1 GATE Annotations

Contrary to PDT, GATE annotations[7] 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*[8] (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 6.3.2 encoding of PDT dependency annotations in GATE will be presented.

3.5 Named Entity Recognition

3.6 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

[7] `http://gate.ac.uk/userguide/sec:corpora:dags`

[8] `http://gate.ac.uk/userguide/sec:corpora:features`

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 in Chapter 8.

3.6.1 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 8.2). 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 8.3) 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 8.5 and Figure 6.5). 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.

3.6.2 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]).

3.6.3 ILP tool

As an ILP tool we have used "A Learning Engine for Proposing Hypotheses" (Aleph v5)[9], which we consider 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[10]) in its background.

Na rozdil od prologu examples we dvou souborech.

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

From our experiments (Section 6.4) can be seen that ILP is capable to find complex and meaningful rules that cover the intended information.

?? large amount of training data ??

As we do not have large amount of training data, there is no problem with excessive time demands during learning and the application of the learned rules is simple and quick.

3.7 Weka

Weka [Hall *et al.*, 2009] is a well known data-mining software.

Weka data mining software[11] [Hall *et al.*, 2009]

Weka Experimenter

experimenter

usage: evaluation of Fuzzy ILP Classifier (Section 8.6.1)

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

[9] `http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/`

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

[11] `http://www.cs.waikato.ac.nz/ml/weka/`

Chapter 4

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, discussions of relevant common aspects of the datasets will be discussed.

4.1 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.

4.1.1 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 4.3.1). 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 4.3.1)
- Czech Fireman Reports Manually Annotated (Section 4.3.2)
- Corporate Acquisition Events (Section 4.3.3)

4.1.2 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 4.3.6)
- Classification Datasets from UCI ML Repository (Section 4.3.7)

These datasets were used for evaluation of the Fuzzy ILP classification method in Section 8.6.1.

4.1.3 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 4.3.4)
- RDF Dataset Based on Corporate Acquisition Events (Section 4.3.5)

These datasets were used for evaluation of the idea of shareable extraction ontologies, see Section 7.5.

4.2 Origin of the Datasets

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

4.2.1 Contributed Datasets

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

- Czech Fireman Reports without Annotations (Section 4.3.1)
- Czech Fireman Reports Manually Annotated (Section 4.3.2)

- RDF Dataset Based on Czech Fireman Reports (Section 4.3.4)
- RDF Dataset Based on Corporate Acquisition Events (Section 4.3.5)
- Classification Dataset Based on Czech Fireman Reports (Section 4.3.6)

All the contributed datasets can be downloaded from the web site of the project.

4.2.2 Third Party Datasets

Datasets that were used without any additional contributions are:

- Corporate Acquisition Events (Section 4.3.3)
- Classification Datasets from UCI ML Repository (Section 4.3.7)

4.3 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.

4.3.1 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 (footnote).

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 55599 (ID) from Olomoucky region has following URL:

```
http://aplikace.mvcr.cz/archiv2008/rs_atlantic/hasici/olomoucky/55599.html
```

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 3.2 for details about the tools and formats). Table 4.1 summarizes some properties of the dataset's reports described bellow. Total sum, average value, standard deviation, minimum, maximum and median of the values is counted 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.

	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 4.1: Dataset 4.3.1 – Czech Fireman Reports without Annotations – statistics

annotation type	**number of annotations**	**used in evaluation**
profesional_unit	78	yes
cars	45	no
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
profesional_units	14	yes
fatalities	11	yes
units	10	no
aqualung	8	no
duration	3	no
fan	3	no
amateur_units	1	no
lather	1	no

Table 4.2: Dataset 4.3.2 – Czech Fireman Reports Manually Annotated – manual annotations

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 5.7.1.

4.3.2 Czech Fireman Reports Manually Annotated

The dataset is made up of 50 articles selected form the dataset described in the previous section. Several kind of information were identified and manually annotated in the articles:

»>Event extraction?

»»»»»>Vlozip popis jednotlivych typu informace«««

Counts of individual annotations are summarized in Table 4.2.

czech_fireman

The fist dataset is called 'czech_fireman'. This dataset was created by ourselves during the development of our IE engine. It is a collection of 50 Czech texts that are reporting on some accidents (car accidents and other actions of fire rescue services). These reports come from the web of Fire rescue service of Czech

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

Table 4.3: Dataset 4.3.3 – Corporate Acquisition Events – manual annotations

Republic[1]. The labeled corpus is publically available on the website of our project[2]. The corpus is structured such that each document represents one event (accident) and several attributes of the accident are marked in text.

4.3.3 Corporate Acquisition Events

Acquisitions v1.1

The second dataset is called "Corporate Acquisition Events" and it is described in [Lewis, 1992]. More precisely we use the *Acquisitions v1.1* version[3] of the corpus. This is a collection of 600 news articles describing acquisition events taken from the Reuters dataset. News articles are tagged to identify fields related to acquisition events. These fields include 'purchaser' , 'acquired', and 'seller' companies along with their abbreviated names ('purchabr', 'acqabr' and 'sellerabr') Some news articles also mention the field 'deal amount'.

4.3.4 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 Chapter 7). The datasets consists of so called document ontologies – RDF representation of source documents (see Section 7.4.1) and extraction ontologies – RDF representation of extraction rules (see Section 7.1.1). 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 7.5.

This dataset is based on the IE dataset Czech Fireman Reports Manually Annotated (Section 4.3.2); 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 7.5.1 provides also some additional information (basic statistics) about the dataset.

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

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

[3] This version of the corpus comes from the Dot.kom (Designing infOrmation extracTion for KnOwledge Management) project's resources: `http://nlp.shef.ac.uk/dot.kom/resources.html`

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
profesional_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 4.4: Accident attributes.

4.3.5 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 4.3.3). It consists of 600 document ontologies, one mapping ontology and currently one extraction ontology, which contains extraction rules for the 'acquired' task.

Section 7.5.1 provides also some additional information (basic statistics) about the dataset.

4.3.6 Classification Dataset Based on Czech Fireman Reports

The main experiment presented in this chapter leads to the seriousness classification of an accident presented in a web report. We use online fire department reports from several regions of the Czech Republic. These reports are written in Czech and can be accessed through the web of the General Directorate of the Fire and Rescue Service of the Czech Republic[4].

For our experiment we have selected a collection of 50 web reports. We have identified several features presented in these reports and manually extracted the corresponding values. This will be described in more detail in Section 4.3.6. 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 whole dataset can be downloaded from our Fuzzy ILP classifier's web page[5].

In this 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. The integration step still lies ahead.

Features of accidents

Table 4.4 summarizes all features (or attributes) that we have obtained from accident reports. Except for the attribute `type` (type of an accident – `fire`, `car_accident` and `other`), all the attributes are numerical and therefore *monotonizable* (see the next sections). There are cases in which the value of an attribute is unknown. We have 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.

[4] `http://www.hzscr.cz`

[5] `http://www.ksi.mff.cuni.cz/~dedek/fuzzyILP/`

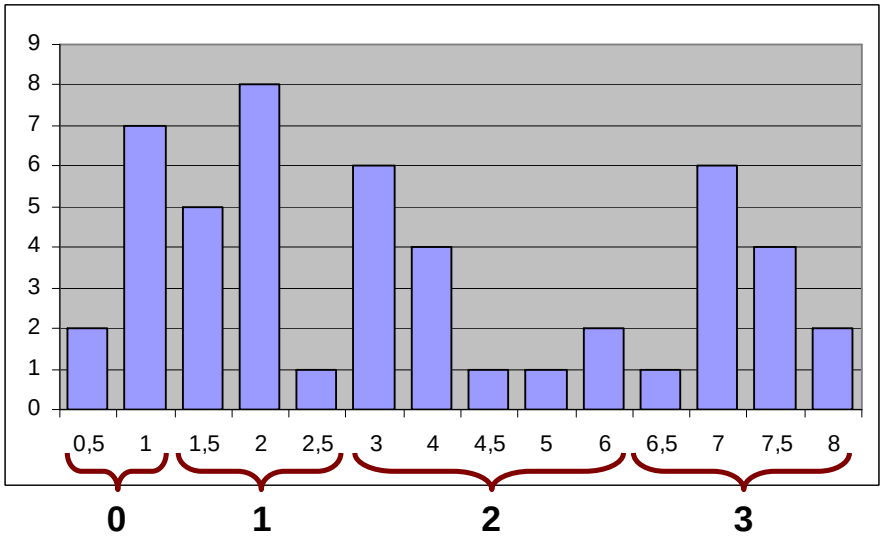


Figure 4.1: Frequencies of the seriousness ranking.

- `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 4.1. We have divided the values into four approximately equipotent groups (see in Figure 4.1) and these groups determine the target class attribute of the classification task.

4.3.7 Classification Datasets from UCI ML Repository

UCI datasets

To answer these questions we performed another experiment. We selected several datasets from the University of California, Irvine, Repository of Machine Learning Databases (UCI) [Frank and Asuncion, 2010] and evaluated all the methods against them.

The list of selected datasets can be found in the legend of Table 8.2. All the datasets are monotonizable (the target attribute can be naturally ordered), so the fuzzy classifier could take advantage of that.

Chapter 5

Extraction Method Based on Manually Created Rules

5.1 Introduction

5.1.1 Presented Extraction Methods

The main motivation for creating both these methods was an attempt to use deep linguistic analysis of natural language texts. 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 a 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 (not changing the factual meaning of a sentence) of the word order.

5.1.2 Manually Created Rules

In this chapter 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 several techniques and 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. At the end of the section several experiments will be presented to explain the usefulness of the method.

5.2 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 and destination of the extraction process is the extracted information stored in the form of a semantic web ontology data or more precisely semantic web ontology instance. 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 user; Section 5.5 provides details about that. A description of the individual steps follows.

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.

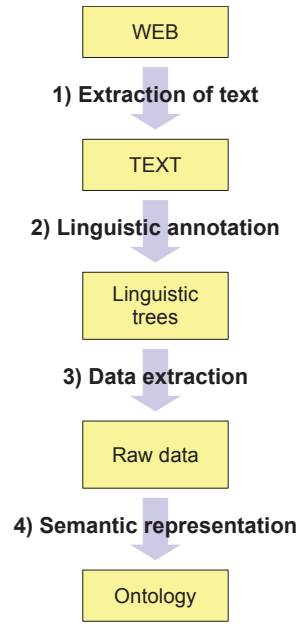


Figure 5.1: Schema of the extraction process.

Figure 5.2: One web page with an accident report.

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 3.2.1), but in this chapter only PDT based tools were used and the 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 structured data form the text. The extraction method used in this phase is the main topic of the present chapter and many details about the method are discuses in next sections.

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 and the output has to be converted to the format of an appropriate ontology 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 thing 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 Chapter 7) because this task seems to be strongly dependent on manual work of human designers and its potential for meaningful scientific investigation seems to be rather small. Details about this step are discussed in Section 5.6.

5.3 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 is listed in Figure 5.3. It clearly demonstrates all drawbacks of the procedural rule design: 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 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 Chapter 6) 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 Chapter 7).

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

my @injure_verbs = ("zranit", "usmrtit", "zemřít", "zahynout", "přežít");

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 injurance
                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} . "\">";

                    #patients 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 5.3: 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 5.4: *XML* structured output of the query written in *Btred*.

5.4 Implementation

5.4.1 Procedural Extraction Rules

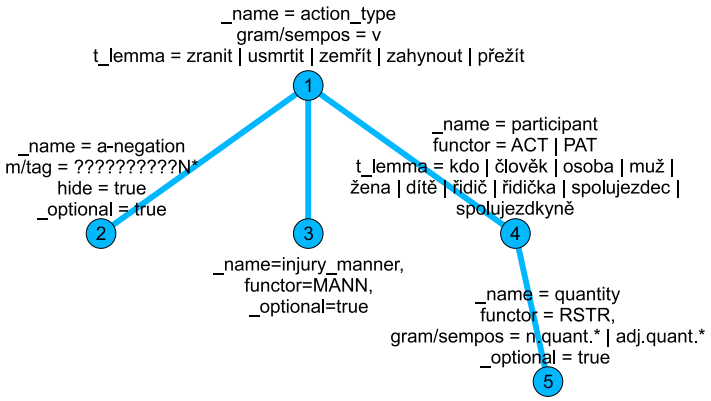
Our first implementation was based on Btred API for processing linguistic trees. The extraction rules were implemented as Perl Btred procedures that were evaluated on all of the trees of an arbitrary corpus by Btred. An example of such extraction rule is in Figure 5.3 and corresponding extraction output on Figure 5.4.

5.4.2 Netgraph Based Extraction Rules

The second implementation was based on Netgraph – a linguistic tool used for searching through a syntactically annotated corpus (see Section 3.2.4 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 in this text we will use the graphical representation only. Examples of a Netgraph queries (in the graphical representation) can be found in Figures 5.5 and 5.6. They clearly show the necessary tree structure that has to be present in any matching tree and attribute restrictions are printed beside the corresponding nodes.

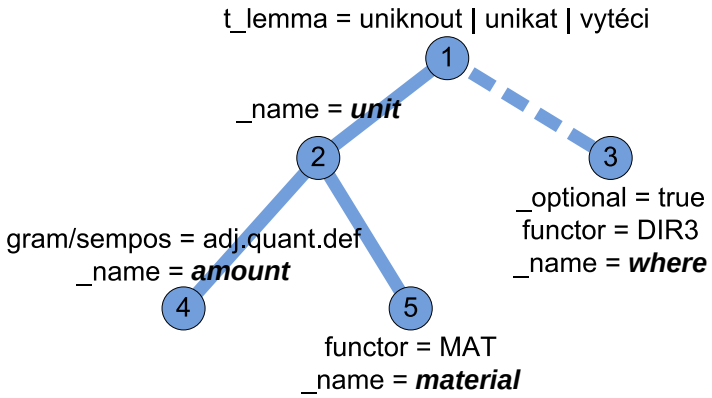
The Netgraph based extraction procedure works with extraction rules in a pseudo SQL SELECT form:



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		

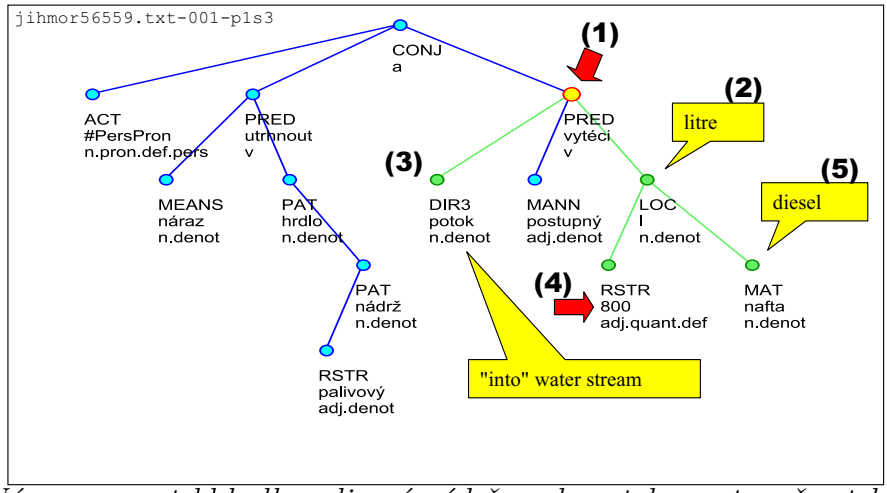
Figure 5.5: A manually created extraction rule investigating numbers of injuries and fatalities.



Transcript:

uniknout, unikat to leak out	vytéci to flow out

Figure 5.6: 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 5.7: A tree matching with the corresponding extraction rule in Figure 5.6.

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

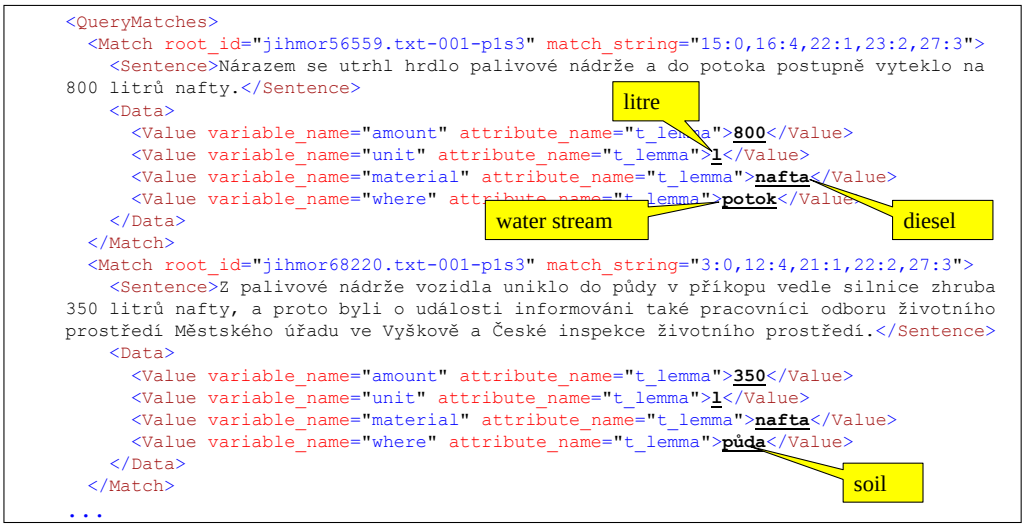


Figure 5.9: *XML* structured output of the SQL select like query corresponding with the extraction rule in Figure 5.6 and matching tree in Figure 5.7.

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

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.

5.4.3 Illustration Examples

Let us explain it in more detail by using the example of extraction rule from the Figure 5.5, 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.

Evaluation of the extraction rule from Figure 5.6 is illustrated on Figure 5.7. Figure 5.7 shows a linguistic tree matching with the extraction rule. Matching nodes are decorated and labeled by the numbers of corresponding query nodes.

5.4.4 Extraction Output

Small pieces of extraction outputs are shown in Figure 5.8 (for the extraction rule in Figure 5.5) and in Figure 5.9 (for the extraction rule in Figure 5.6).

The former example (Figure 5.8) contains three matches of the extraction rule in three different articles. Each query match is closed in the `<Match>` element and each contains values of some linguistic attributes closed inside the `<Value>` elements. Each value comes from some of the nodes of the extraction rule. Name of corresponding query node is saved in the `variable_name` attribute of the `<Value>` element.

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).

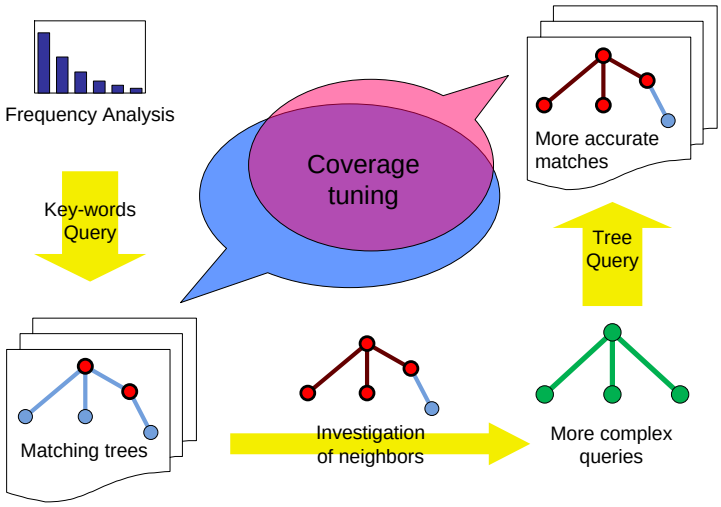


Figure 5.10: Gradual refinement of an extraction rule.

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).

5.5 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 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 5.10 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 5.5 — each node keeps different kind of information.

5.6 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.

So far in the description of the implementation of the extraction method the output of the extraction was in the form of although structured but still proprietary XML format. This corresponds to the penultimate stage (raw data) of our data flow schema (Section 5.2). In this section we will describe details about the last step of the data flow – semantic representation of extracted data.

First interesting thing is the difference between the output formats of the procedural extraction rules (Section 5.4.1, Figure 5.4) and the Netgraph based extraction rules (Section 5.4.2, Figure 5.8). Apparently 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 extraction process than on the data. The difference is clearly connected with the difference of the design processes of the 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. Contrary, during coding of such procedural rule the programmer has great freedom in the design of that rule and he or she can adapt the rule to precisely fit with the data. A designer of a Netgraph based rule has the only freedom in the construction of the Netgraph query and in selection particular query nodes and linguistic attributes that will be printed on the output.

The goal of the semantic extraction and annotation is to output the extracted information in the form of a semantic web ontology. This is not difficult in the case of procedural rules. If the target ontology is selected then 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). This 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]).

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 they will be rendered on the output. A graphical representation of such extraction query can look like in Figure 5.11. It shows 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 probably the most interesting part of such extraction queries. The linguistic information on one side has to be converted the ontological information on the other side. Several translation types have to be supported e.g. in Figure5.11 a translation of numerals to numbers, lexical translation from a source language (Czech), and detection of negation present in a query node are used.

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

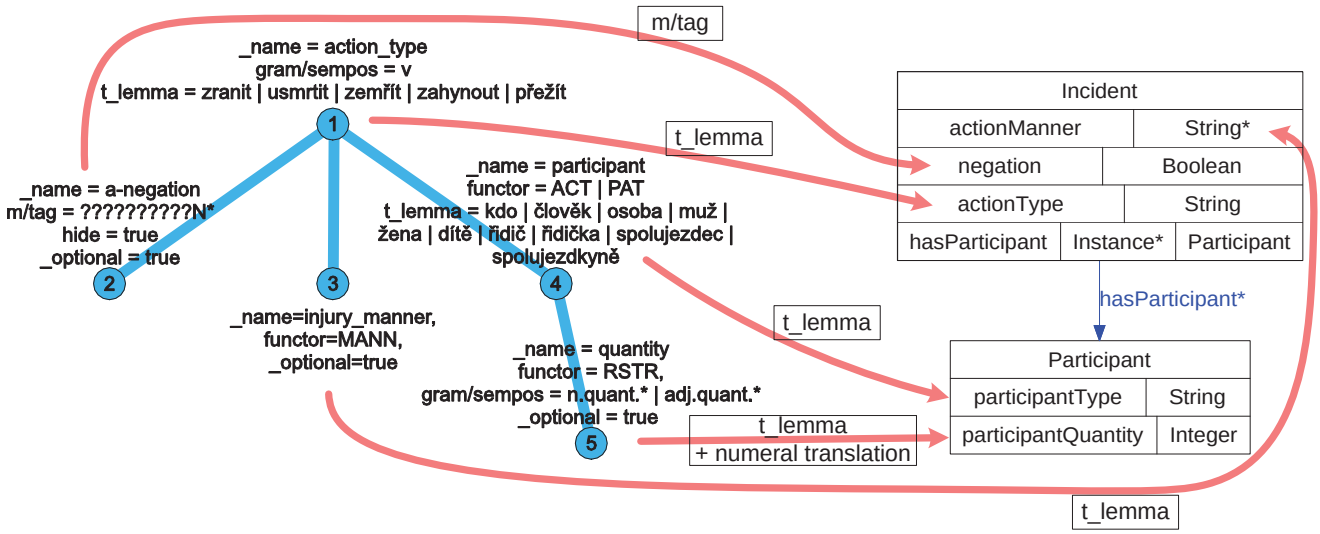


Figure 5.11: Semantic interpretation of the extraction rule.

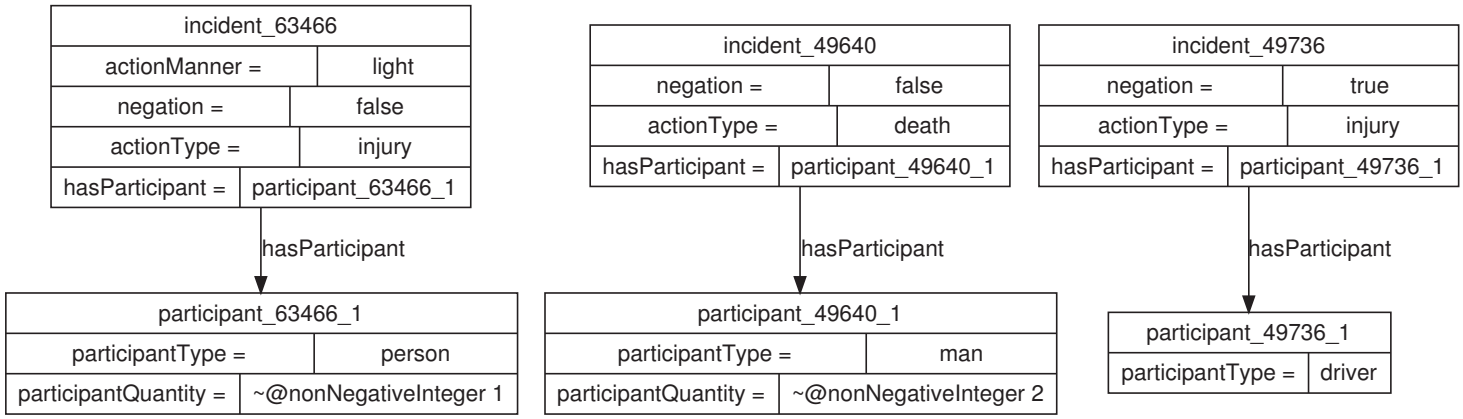


Figure 5.12: Extracted instances of the target ontology.

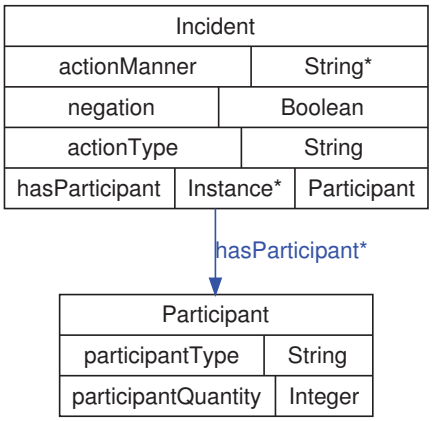


Figure 5.13: Schema of the target ontology.

```
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 5.14: *SPARQL* query that summarizes fatalities of particular incidents.

For better illustration Figure 5.12 shows how the extraction output would look like in the semantic case. It is based on the same data as in the Figure 5.8. The presented ontology was designed only for the illustration. Schema of the ontology can be seen in the Figure 5.13. 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 5.14) 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 data.

5.7 Experiments and Evaluation

In this section two experiments with manually created extraction rules will be presented. 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.

5.7.1 First Experiment – Quantitative Evaluation

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 5.3 and the Netgraph based rules correspond with the rule in Figure 5.5. 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 5.1 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

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

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.

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.

	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 5.2: Evaluation of the manually created rule form Figure 5.5 on the manually annotated dataset.

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.)
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 99. 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).

5.7.2 Second Experiment – Qualitative Evaluation

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 5.2 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.

	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 5.3: 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 5.4: Cross method comparison of found instances.

Manual Design of Rules Using Training Data Set

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 (see Chapter 6).

The results are summarized in Table 5.3. 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 5.4 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 different datasets, extraction tasks and human designers.)

5.8 Conclusion

Note that in this case we did not concern the annotation aspect of information extraction. Although it is possible to infer an annotation based variety of the presented method, here 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 next section the emphasis will be 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.

Chapter 6

Extraction Method Based on ILP Machine Learning

6.1 Introduction

Automated semantic annotation (SA) is considered to be one of the most important elements in the evolution of the Semantic Web. Besides that, SA can provide great help in the process of data and information integration and it could also be a basis for intelligent search and navigation.

In this chapter we present a method for classical and semantic information extraction and annotation of texts, which is based on a deep linguistic analysis and Inductive Logic Programming (ILP). 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.

A description, implementation and initial evaluation of the method are the main contributions of the present work.

6.2 Data Flow ?or Schema of the Extraction Process?

Figure 6.1

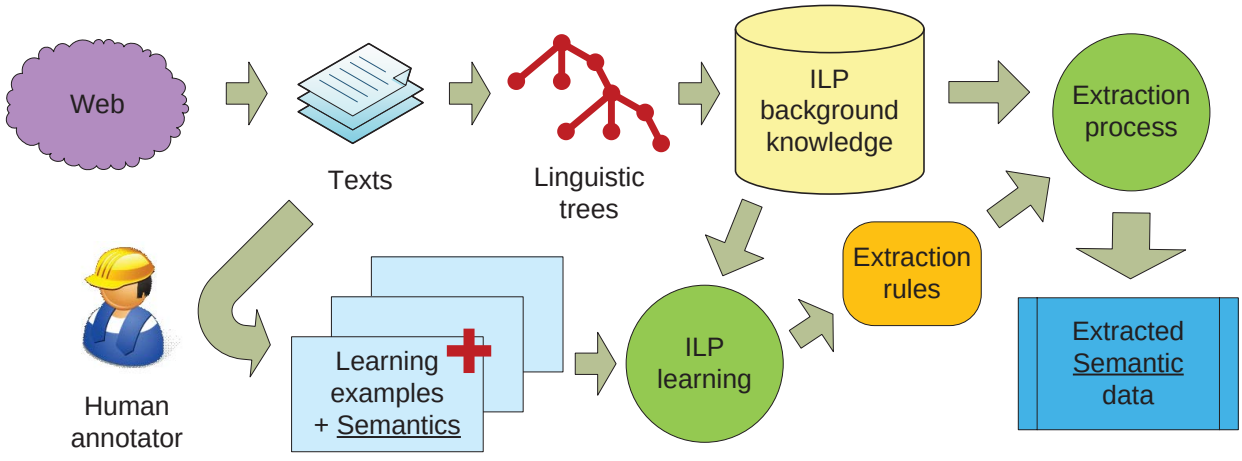


Figure 6.1: ILP data flow.

6.3 Implementation

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

6.3.1 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[1]. 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 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[2]).

6.3.2 PDT Annotations in GATE

Although GATE annotations are just highlighted pieces of text (see also Section 3.4.1) 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[3].

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 (dependencies) are implemented by the TectoMT wrapper: analytical dependencies, tectogrammatical dependencies, aux.rf (auxiliary reference) and lex.rf (main lexical reference). The last two

[1] `http://gate.ac.uk/userguide/sec:ml:batch-learning-pr`
[2] `http://www.xmlrpc.com/`
[3] `http://gate.ac.uk/userguide/sec:parsers:stanford`

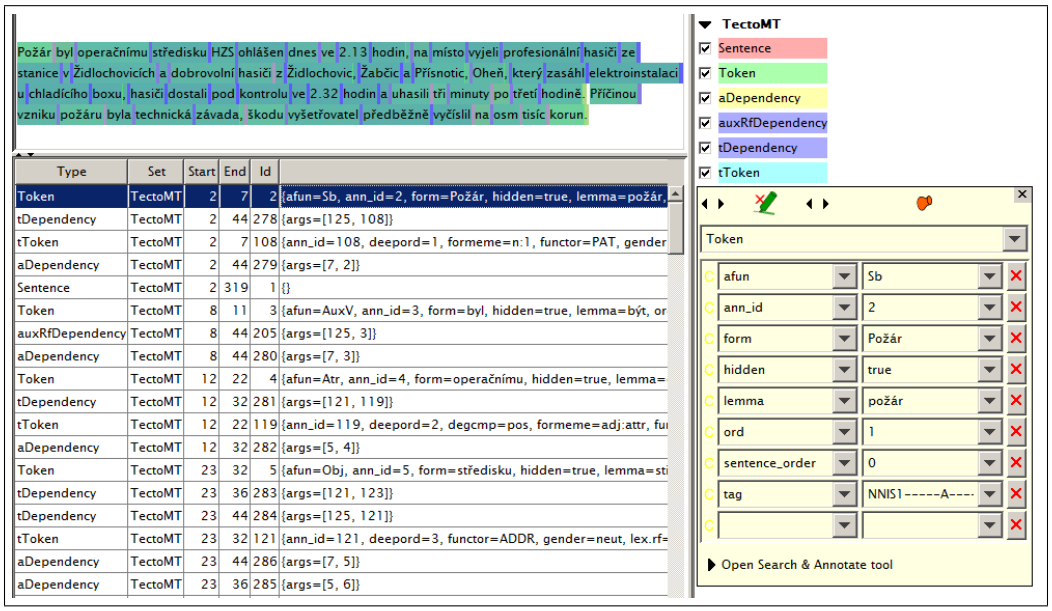


Figure 6.2: PDT annotations in GATE (screenshot).

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 GATE annotation feature can only have one value per 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 the wrapper for the Stanford Parser (using the single list feature "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 6.2) that partly illustrates that.

Netgraph Tree Viewer

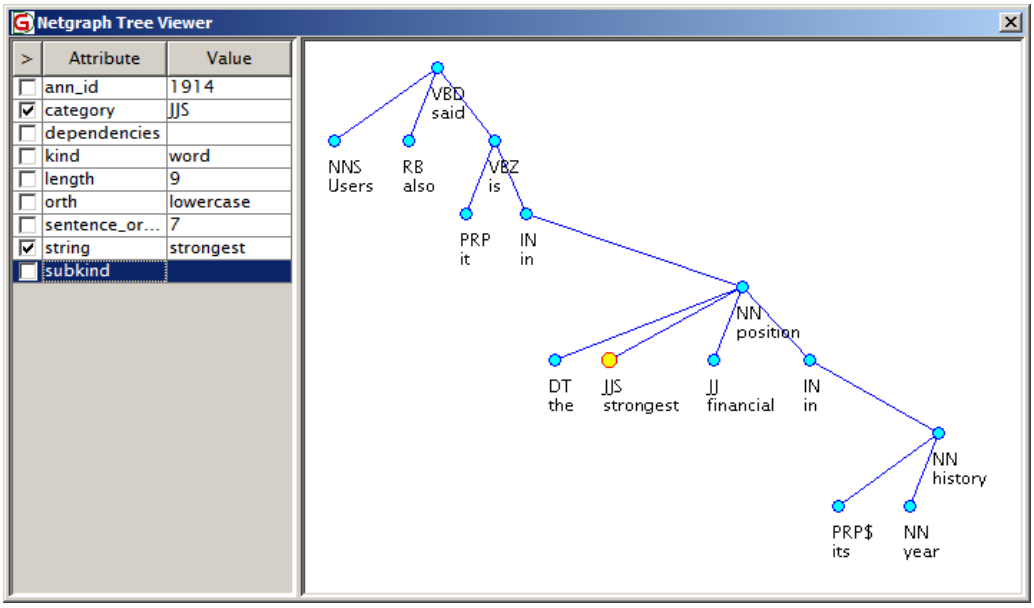
Figure 6.3

6.3.3 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.

In both steps the linguistic analysis has to be done before and in both steps 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 6.3.4.



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

Figure 6.3: Netgraph Tree Viewer in GATE (for Stanford Dependencies, screenshot).

After the ILP serialization is done, 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 (and we did so in the present solution), but it can be optionally changed to any other textual unit, for example only numerals or tectogrammatical nodes (words with lexical meaning) can be selected. This can be done easily with the help of *Machine Learning PR* (LM PR) from GATE[4].

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.

6.3.4 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 6.3.2: 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 (sentence order and node deep order are used outside of GATE, see PML→RDF transformation in Section 7.4.2.) These node IDs correspond to simple atoms and they represent tree nodes in the fact database. A node type (used by the ILP learning algorithm) is assigned to a node ID by predicates

[4] *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.

`Token(NodeID)` for analytical tree nodes and `tToken(NodeID)` for tectogrammatical tree nodes.

Tree nodes are connected by edges using binary predicates of the 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 minority of them is usually used in extraction rules.

Example of a serialized tectogrammatical tree is in Figure 6.4 it is the same tree as in Figure 1.4.

6.3.5 ?Connecting Linear GATE Annotations with Tree Nodes? Intersection with Tree Nodes?

How to compare correctness of IE?

ILP identifies relevant tree nodes but manual gold standard annotations are put directly on the surface text.

A mapping of tree nodes and text surface has to be established.

6.3.6 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 application 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* 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 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*. We have successfully used this technique for the 'damage' task of our evaluation corpus (See Section 6.4 for details.)

6.3.7 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

```
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 6.4: ILP serialization example

annotation and saves it as a label of a new instance or as a value of some property of a shared 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*[5]:

> "...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, 2008a].

6.3.8 How to Download

The project website[6] 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.

6.4 Evaluation

6.4.1 Dataset

We have evaluated our state of the art solution on a small dataset that we use for development. It is a collection of 50 Czech texts that are reporting on some accidents (car accidents and other actions of fire rescue services). These reports come from the web of Fire rescue service of Czech Republic[7]. The labeled corpus is publically available on the web of our project[8]. The corpus is structured such that each document represents one event (accident) and several attributes of the accident are marked in text. For the evaluation we selected two attributes of different kind. The first one is 'damage' – an amount (in CZK - Czech Crowns) of summarized damage arisen during a reported accident. The second one is 'injuries', it marks mentions of people injured during an accident. These two attributes differ. Injuries annotations always cover only a single token, while damage annotations usually consist of two or three tokens – one or two numerals express the amount and one extra token is for currency.

These two attributes differ in two directions:

1. Injuries annotations always cover only a single token while damage usually consists of two or three tokens - one or two numerals express the amount and one extra token is for currency.

2. The complexity of the marked information (and the difficulty of the corresponding extraction task) differs slightly. While labeling of all money amounts in the corpus will result in 75% accuracy for damage annotations, in the case of injured persons mentions there are much more possibilities and indications are more spread in context.

6.4.2 Comparison with Paum classifier

To compare our solution with other alternatives we took the Paum 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

[5] http://www.ebusiness-unibw.org/wiki/Modeling_Product_Models#Recipe:_.22Incremental_Enrichment.22
[6] http://czsem.berlios.de
[7] http://www.hzscr.cz/hasicien/
[8] http://czsem.berlios.de/

task/method	matching	missing	excessive	overlap	prec.%	recall%	F1.0%
damage/ILP	14	0	7	6	51.85	70.00	59.57
damage/ILP – lenient measures					74.07	100.00	85.11
dam./ILP-roots	16	4	2	0	88.89	80.00	84.21
damage/Paum	20	0	6	0	76.92	100.00	86.96
injuries/ILP	15	18	11	0	57.69	45.45	50.85
injuries/Paum	25	8	54	0	31.65	75.76	44.64
inj./Paum-afun	24	9	38	0	38.71	72.73	50.53

Table 6.1: Evaluation results

lemmas and morphological tags. The precision was further improved by adding the feature of *analytical function* from the syntactic parser (see the last row of Table 6.1).

Because we did not want to invest much time to this and the feature setting of the Paum learner was quite simple (a window of two preceding and following token lemmas and morphological tags). We admit that looking for better features could further improve the results of the Paum learner.

6.4.3 Cross validation

We used the 10-fold cross validation in the evaluation. Thanks to this technique the evaluation is simple. After processing all the folds each document is processed with some of the ten learned models such that the particular document was not used in learning of that model, so all documents are unseen by the model applied on them. At the end we just compare the gold standard annotations with the learned ones in all documents.

6.4.4 Results

Results of a 10-fold cross validation are summarized in Table 6.1. We used standard information retrieval performance measures: precision, recall and F_1 measure and also theirs lenient variants (overlapping annotations are added to the correctly matching ones, the measures are the same if no overlapping annotations are present).

In the first task ('damage') the methods obtained much higher scores then in the second ('injuries') because the second task is more difficult. In the first task also the root/subtree preprocessing/postprocessing improved results of ILP such that afterwards, annotation borders were all placed precisely. The ILP method had better precision and worse recall than the Paum learner but the F_1 score was very similar in both cases.

Statistical Significance

The term statistical significance 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[9] is used. Test significance is 0.05 in all cases.

6.4.5 Examples of learned rules

In Figure 6.5 we present some examples of the rules learned from the whole dataset. The rules demonstrate a connection of a target token with other parts of a sentence through linguistic syntax structures. For example the first rule connects a root numeral (*n.quant.def*) of 'damage' with a mention of 'investigator' that stated the mount. In the last rule only a positive occurrence of the verb 'injure' is allowed.

Experience with human-designed rules.

[9] `http://www.cs.waikato.ac.nz/ml/weka/`

```
%[Rule 1] [Pos cover = 14 Neg cover = 0]
damage_root(A) :- lex_rf(B,A), has_sempos(B,'n.quant.def'), tDependency(C,B),
   tDependency(C,D), has_t_lemma(D,'investigator').

%[Rule 2] [Pos cover = 13 Neg cover = 0]
damage_root(A) :- lex_rf(B,A), has_functor(B,'TOWH'), tDependency(C,B),
   tDependency(C,D), has_t_lemma(D,'damage').


%[Rule 1] [Pos cover = 7 Neg cover = 0]
injuries(A) :- lex_rf(B,A), has_functor(B,'PAT'), has_gender(B,anim),
   tDependency(B,C), has_t_lemma(C,'injured').

%[Rule 8] [Pos cover = 6 Neg cover = 0]
injuries(A) :- lex_rf(B,A), has_gender(B,anim), tDependency(C,B),
   has_t_lemma(C,'injure'), has_negation(C,neg0).
```

TODO: `damage_root -> mention_root`

Figure 6.5: Examples of learned rules, Czech words are translated.

6.5 Conclusion and Future Work

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 because Czech is a language with free word order and we would expect much better results of the dependency 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 should be evaluated on bigger datasets (e.g. MUC, ACE, TAC, CoNLL) and other languages. So far we also do not provide a method for classical relation extraction (like e.g. in [Bunescu and Mooney, 2007]). In the present solution we deal with relations implicitly. The method has to be adapted for explicit learning of relations in the form of "subject predicate object".

Our method can also provide a comparison of linguistic formalisms and tools because on the same data we could run our method using different linguistic analyzers and compare the results.

Chapter 7

Shareable Extraction Ontologies

7.1 Introduction

Information extraction (IE) and automated semantic annotation of text are usually done by complex tools and all these tools 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 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.)

In the beginning, a model is either created by a human user or it is learned from a 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 tool. These models are usually stored in proprietary formats and they are accessible only by the corresponding tool.

In the environment of the Semantic Web it is essential that information is shareable and some ontology based IE tools 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 tool and cannot be used separately. The strong dependency of an extraction ontology on the corresponding tool makes it very difficult to share. When an extraction ontology cannot be used outside the tool there is also no need to keep the ontology in a standard ontology format (RDF or OWL). The only way how to use such extraction ontology is within the corresponding extraction tool. It is not necessary to have the ontology in a "owl or rdf file". In a sense such extraction ontology is just a configuration file. For example in [Labský *et al.*, 2009] (and also in [Embley *et al.*, 2002]) the so called extraction ontologies are kept in XML files with a proprietary structure and it is absolutely sufficient, there is no need to treat them differently.

7.1.1 Shareable Extraction Ontologies

In this chapter we present an extension of the idea of extraction ontologies. 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.

In this chapter 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 7.4) 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.

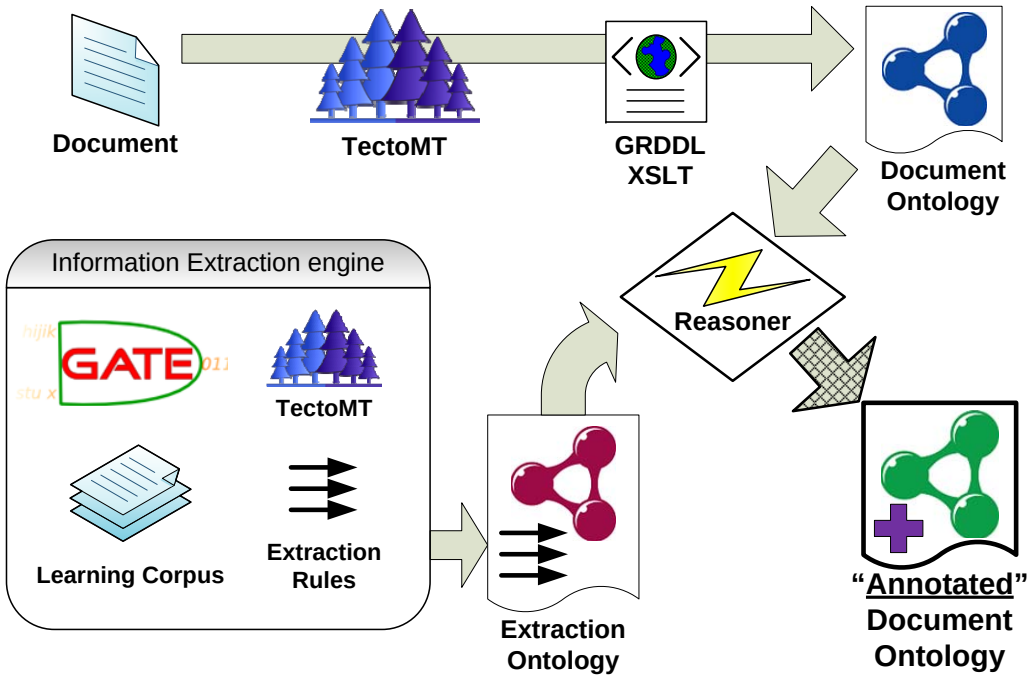


Figure 7.1: Semantic annotation driven by an extraction ontology and a reasoner – schema of the process.

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 Fig 7.1 and it will be described in detail in Section 7.4.2.

Section 7.2 presents several closely related works. The main idea of the chapter will be described in Section 7.3, its implementation in Section 7.4 and in Section 7.5 an experiment with several OWL reasoners and IE datasets will be presented. In Section 7.6 related issues are discussed and Section 7.7 concludes the chapter.

7.2 Related Work

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.*, 2006b] is a closely related discipline. Many extraction and annotation tools can be found in the above mentioned surveys ([Wimalasuriya and Dou, 2010; Chang *et al.*, 2006b]), 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[1] 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

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

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 extraction 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[2] 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 Sect. 7.4.1).

7.3 Semantic Annotation Semantically

The problem of extraction ontologies that are not shareable was pointed out in the introduction (Section 7.1). 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. In this chapter we present a proof of concept that 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. And this is where the title of the chapter and the present section comes from. 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. In the presented solution the approaching support for Semantic Web Rule Language (SWRL) [Parsia *et al.*, 2005] is exploited. Although SWRL is not yet approved by W3C it is already widely

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

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. The case study in Section 7.4 demonstrates the translation of the native extraction rules to SWRL rules that form the core of the extraction ontology.

7.4 The Main Idea Illustrated – a Case Study

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

7.4.1 Document Ontologies

The main idea of this chapter 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.

7.4.2 Implementation

In this section we will present details about the case study. We have used our IE engine [Dědek, 2010] based on deep linguistic parsing and Inductive Logic Programming. 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 described above.

Linguistic Analysis

Our 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 construction and application of extraction rules.

We usually use TectoMT system to do the linguistic preprocessing. TectoMT is a Czech project that contains many linguistic analyzers for different languages including Czech and English. We are using a majority of applicable tools from TectoMT: a tokeniser, a sentence splitter, morphological analyzers (including POS tagger), a syntactic parser and the deep syntactic (tectogrammatical) parser. All the tools are based on the dependency based linguistic theory and formalism of PDT.

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

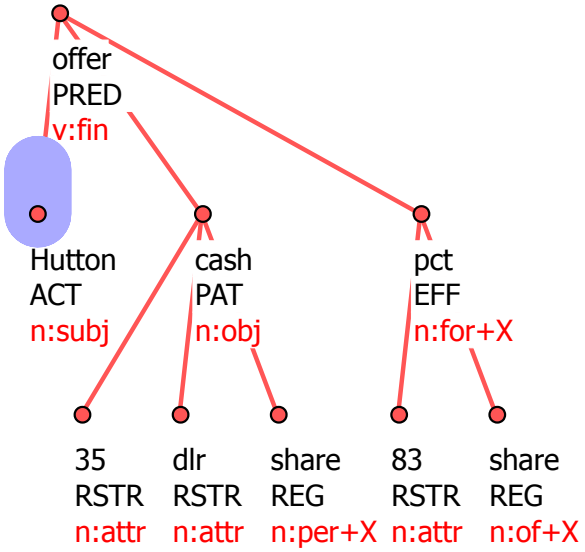


Figure 7.2: Tectogrammatical tree of the sentence: "Hutton is offering 35 dlrs cash per share for 83 pct of the shares." Nodes roughly correspond with words of a sentence, edges represent linguistic dependencies between nodes and some linguistic features (tectogrammatical lemma, semantic functor and semantic part of speech) are printed under each node. The node 'Hutton' is decorated as a named entity.

(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. An example of such (tectogrammatical) tree structure is in Figure 7.2.

PML→RDF Transformation

In this case study PML files made from source documents by TectoMT are transformed to RDF document ontology by quite simple GRDDL/XSLT transformation. Such document ontology contains the whole variety of PML in RDF format.

Rule Transformations

Extraction rules produced by the IE engine are natively kept in a Prolog format; examples can be seen in Figure 7.5. The engine is capable to export them to the OWL/XML[3]syntax for rules in OWL 2 [Glimm *et al.*, 2009] (see in Figure 7.7).Such rules can be parsed by OWL API[4] 3.1 and exported to RDF/SWRL, which is very widely supported and hopefully becoming a W3C recommendation. Figure 7.6 shows the example rules in Protégé[5] 4 – Rules View's format. The last rule example can be seen in Figure 7.8, it shows a rule in the Jena rules format[6]. Conversion to Jena rules was necessary because it is the only format that Jena can parse, see details about our use of Jena in Section 7.5. The Jena rules were obtained using following transformation process: OWL/XML → RDF/SWRL conversion using OWL API and RDF/SWRL → Jena rules conversion using SweetRules[7].

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.

[3] http://www.w3.org/TR/owl-xmlsyntax/
[4] http://owlapi.sourceforge.net/
[5] http://protege.stanford.edu/
[6] http://jena.sourceforge.net/inference/#RULEsyntax
[7] http://sweetrules.semwebcentral.org/

```
@prefix node: <http://czsem.berlios.de/ontologies/.../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 7.3: RDF serialization example

```
@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 .
##############################################################################################
```

TODO: extraction rules jsou anglicky !!! vysledek (Annotated document ontology) cesky !!!!!!!

Figure 7.4: Annotated document ontology example

```
%[Rule 1] [Pos cover = 23 Neg cover = 6]
mention_root(acquired,A) :-
        'lex.rf'(B,A), t_lemma(B,'Inc'),
        tDependency(C,B), tDependency(C,D),
        formeme(D,'n:in+X'), tDependency(E,C).
%[Rule 11] [Pos cover = 25 Neg cover = 6]
mention_root(acquired,A) :-
        'lex.rf'(B,A), t_lemma(B,'Inc'),
        tDependency(C,B), formeme(C,'n:obj'),
        tDependency(C,D), functor(D,'APP').
%[Rule 75] [Pos cover = 14 Neg cover = 1]
mention_root(acquired,A) :-
        'lex.rf'(B,A), t_lemma(B,'Inc'),
        functor(B,'APP'), tDependency(C,B),
        number(C,pl).
```

Figure 7.5: Examples of extraction rules in the native Prolog format.

```
#[Rule 1]
lex.rf(?b, ?a), t_lemma(?b, "Inc"),
tDependency(?c, ?b), tDependency(?c, ?d),
formeme(?d, "n:in+X"), tDependency(?c, ?e)
        -> mention_root(?a, "acquired")
#[Rule 11]
lex.rf(?b, ?a), t_lemma(?b, "Inc"),
tDependency(?c, ?b), formeme(?c, "n:obj"),
tDependency(?c, ?d), functor(?d, "APP")
        -> mention_root(?a, "acquired")
#[Rule 75]
lex.rf(?b, ?a), t_lemma(?b, "Inc"),
functor(?b, "APP"), tDependency(?c, ?b),
number(?c, "pl")
        -> mention_root(?a, "acquired")
```

Figure 7.6: Examples of extraction rules in Protégé 4 – Rules View's 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/onto ... 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;number" />
                <Variable IRI="urn:swrl#c" />
                <Literal>pl</Literal>
            </DataPropertyAtom>
        </Body>
        <Head>
            <DataPropertyAtom>
                <DataProperty IRI="&pml;mention_root" />
                <Literal>acquired</Literal>
                <Variable IRI="urn:swrl#a" />
            </DataPropertyAtom>
        </Head>
    </DLSafeRule>
</Ontology>
```

Figure 7.7: Rule 75 in the OWL/XML syntax for Rules in OWL 2 [Glimm *et al.*, 2009].

```
@prefix pml: <http://ufal.mff.cuni.cz/pdt/pml/>.
[rule-75:
        ( ?b pml:lex.rf ?a )
        ( ?c pml:tDependency ?b )
        ( ?b pml:functor 'APP' )
        ( ?c pml:number 'pl' )
        ( ?b pml:t_lemma 'Inc' )
     ->
        ( ?a pml:mention_root 'acquired' )
]
```

Figure 7.8: Rule 75 in the Jena rules syntax.

Schema of the Case Study

A schema of the case study was presented in Figure 7.1. 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 Section 7.4.1.

7.5 Experiment

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 7.5.1). 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) 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.

How to Download

All the resources (including source codes of the case study and the experiment, datasets and ontologies) mentioned in this chapter are publically available on the project's web site[8] and detailed information can be found there.

[8] http://czsem.berlios.de/

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 7.1: Description of datasets that were used.

7.5.1 Datasets

In the experiment we used two slightly different datasets from two different domains and languages. Table 7.5.1 summarizes some basic information about them.

Czech Fireman

4.3.2 resp. 4.3.4

The fist dataset is called 'czech_fireman'. This dataset was created by ourselves during the development of our IE engine. It is a collection of 50 Czech texts that are reporting on some accidents (car accidents and other actions of fire rescue services). These reports come from the web of Fire rescue service of Czech Republic. The corpus is structured such that each document represents one event (accident) and several attributes of the accident are marked in text. For the experiment we selected the 'damage' task – to find an amount (in CZK - Czech Crowns) of summarized damage arisen during a reported accident.

Acquisitions v1.1

4.3.3 resp. 4.3.5

The second dataset is called "Corporate Acquisition Events" and it is described in [Lewis, 1992]. More precisely we used the *Acquisitions v1.1* version[9] of the corpus. This is a collection of 600 news articles describing acquisition events taken from the Reuters dataset. News articles are tagged to identify fields related to acquisition events. These fields include 'purchaser' , 'acquired', and 'seller' companies along with their abbreviated names ('purchabr', 'acqabr' and 'sellerabr'). Some news articles also mention the field 'deal amount'.

For the experiment we selected only the 'acquired' task.

7.5.2 Reasoners

Four OWL reasoners were used in the experiment(namely Jena[10] ,HermiT[11] ,Pellet[12] and FaCT++[13]) 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[14] 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

[9]This version of the corpus comes from the Dot.kom (Designing infOrmation extracTion for KnOwledge Management) project's resources: `http://nlp.shef.ac.uk/dot.kom/resources.html`

[10]`http://jena.sourceforge.net`

[11]`http://hermit-reasoner.com`

[12]`http://clarkparsia.com/pellet`

[13]`http://code.google.com/p/factplusplus`

[14]`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 7.2: Time performance of tested reasoners on both datasets.

to let it continue.

7.5.3 Evaluation Results of the Experiment

Table 7.2 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 bespoke 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.

7.6 Discussion

In this chapter (Section 7.4.1) we have described a 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`[15] from the Protégé project and `makeTemp(?x)` or `makeInstance(?x, ?p, ?v)`[16] from the Jena project. And these solutions can be used in the future work.

7.6.1 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 other way round.

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

7.6.2 Contributions for Information Extraction

The chapter 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. In this chapter we do not present a solution that would improve the performance of IE tools. We also do not provide a proposal of a universal extraction 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.

7.7 Conclusion

In the beginning of the chapter 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 7.4 we presented a case study that shows that the idea of shareable extraction ontologies is realizable. We presented implementation of an 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 7.5 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.

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

Chapter 8

Usage of Annotations – Fuzzy ILP Document Classification

8.1 Introduction

In this chapter 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 Chapters 5 and 6) and fuzzy inductive logic programming (fuzzy ILP).

The chapter 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 chapter we do not provide any details about the information extraction part of the solution (it was already described in Chapters 5 and 6). We concentrate on the classification part and present a detailed study of the fuzzy ILP classification method (so-called 'Fuzzy ILP Classifier').

In this application we are facing the 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 chapter) 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 8.6.1) 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 main *contributions* presented in this chapter are *formal models*, prototype *implementation* and experimental *evaluation* of the Fuzzy ILP Classifier.

The chapter is organized as follows: Design of the experimental system is presented in Section 8.2. Section 8.3 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 8.4, followed by a description of implementation of the models in the system. In Section 8.6 we present the main results of the work, and then we evaluate and compare the methods with other well-known classifiers. Section 8.7 concludes the chapter.

See also Section 2.1, where some closely related works are introduced.

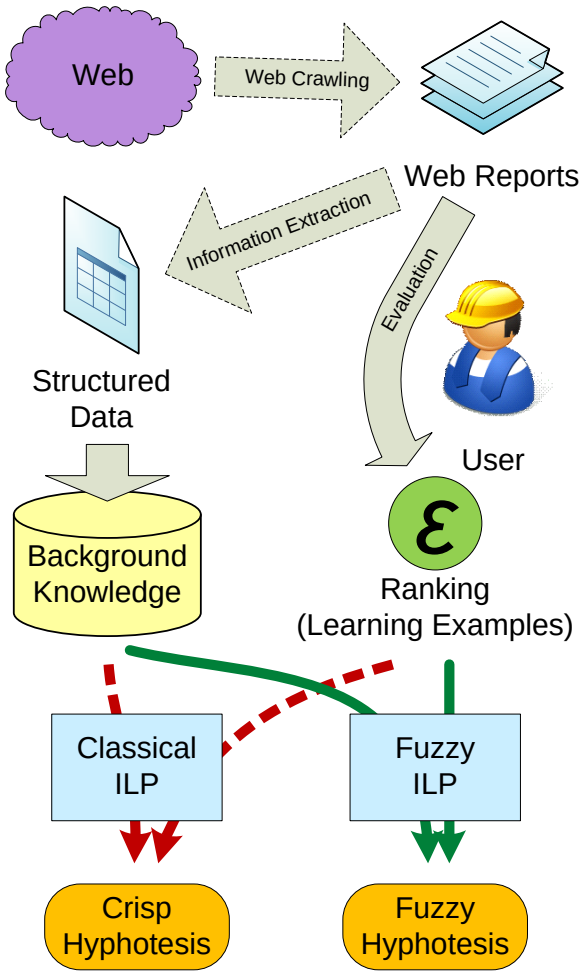


Figure 8.1: Schema of the experimental system.

8.2 Design of the System

A general schema of the experimental system is shown in Figure 8.1. In the schema, previously developed information extraction tools based on third party linguistic analyzers are used (the upper two dashed arrows; see Chapters 5 and 6). The information extraction tools are supposed to extract structured information from the reports and the extracted information is then translated to an ILP knowledge base 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 chapter), see in the next sections.

8.3 The Case Study – Accident Seriousness Classification

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

For the experiment a classification dataset (Section 4.3.6) 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 4.3.6.

In this 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.

The integration step still lies ahead.

8.4 Theoretical Background

Formal definitions of classical and fuzzy ILP tasks were presented in Section 3.6. In this section, additional formal constructions will be presented, because they are the basis for the implementation.

8.4.1 Translation of Fuzzy ILP Task to Several Classical ILP Tasks

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 in [Krajci *et al.*, 2004] it was shown 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:

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

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

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

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

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

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

Note that all x_i and y are variables, t_i and t are constants and $x_{n+1}, \ldots, x_m$ 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

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

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 8.5 (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 $t = 3$:

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

$u(x_1, \ldots, x_m) \geq 3$ 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].

Crisp learning examples

```
serious_2(id_47443). %positive

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

Monotonized learning examples

```
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 8.2: 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).
profesional_units(id_47443, 1).
pipes(id_47443, unknown).
lather(id_47443, 0).
aqualung(id_47443, 0).
fan(id_47443, 0).
```

Figure 8.3: 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 8.4: Monotonization of attributes (damage_atl ← damage).

8.5 Implementation

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 3.6.3) 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 8.6 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 8.2.

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 8.3).

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 8.4. The first rule deals with `unknown` values (Section 4.3.6 deals with unknown values in the dataset) and the second does the monotonization.

Negations used in Figure 8.4 and Figure 8.6 are the standard Prolog *negations as failure*.

Once we have learning examples and background knowledge, we can run the ILP inductive procedure and

```
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 8.5: 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 8.6: Conversion rules for monotonized hypotheses (`serious_t` ← `serious_atl_t`).

obtain learned rules (a learned hypothesis). According to the kind of the ILP task (crisp or monotonized), we obtain the corresponding kind (crisp or monotonized) of rules (see e.g. in Figure 8.5). 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 8.6. 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.

8.6 Results

Figure 8.5 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.

8.6.1 Evaluation

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 3.7). This interface makes it possible to use the ILP methods as an ordinary Weka classifier for any[1] classification task inside the Weka software. This also makes the presented experiments easily repeatable (see Secrion 9.2 for details.)

For our experiment we used the Weka Experimenter (see in Section 3.7) and performed an experiment in which the Crisp and Fuzzy ILP classifiers were compared with five additional classifiers:

- 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 8.7 and in Table 8.1 (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.

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

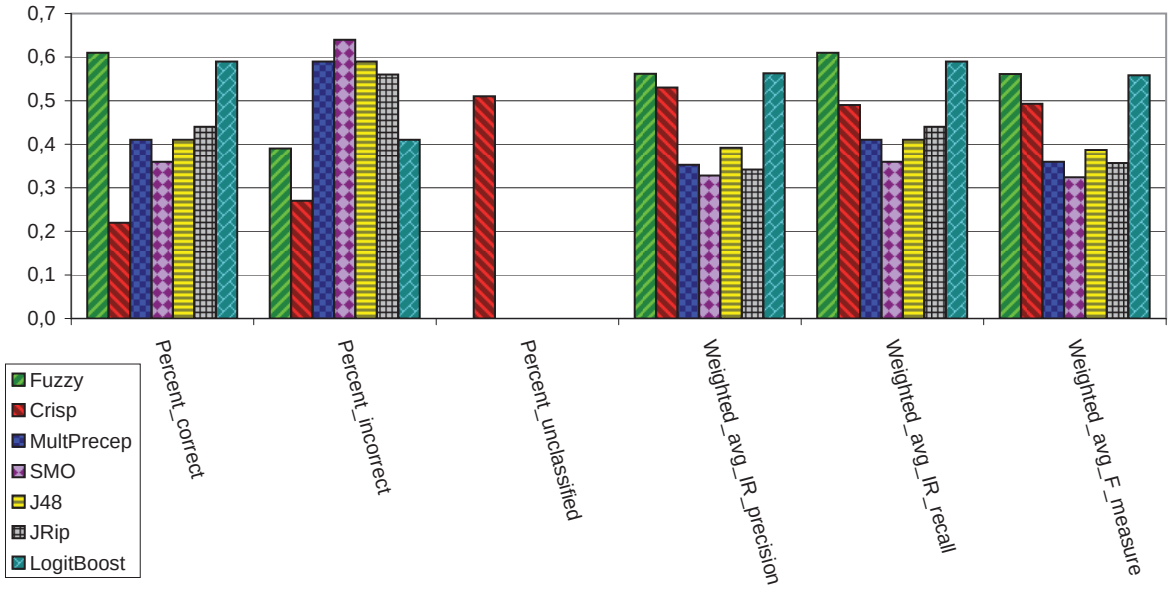


Figure 8.7: Evaluation of the methods – average values.

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

	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

car .....http://archive.ics.uci.edu/ml/datasets/Car+Evaluation
wine .....red wine dataset http://archive.ics.uci.edu/ml/datasets/Wine+Quality
cmc .....http://archive.ics.uci.edu/ml/datasets/Contraceptive+Method+Choice
tae .....http://archive.ics.uci.edu/ml/datasets/Teaching+Assistant+Evaluation
pop .....http://archive.ics.uci.edu/ml/datasets/Post-Operative+Patient
nurs .....http://archive.ics.uci.edu/ml/datasets/Nursery

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

UCI datasets

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 4.3.7) and evaluated all the methods against them.

The list of selected datasets can be found in the legend of Table 8.2. All the 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 8.1) 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 8.2).

Table 8.2 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 8.8 (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 8.9 demonstrates time complexity of the classifiers in the same experiment as in Figure 8.8. 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.

8.7 Conclusion

In this chapter we have presented a fuzzy system, which provides a fuzzy classification of textual reports. Our approach is based on usage of third party linguistic analyzers, our previous work on information extraction, and fuzzy inductive logic programming.

The main contributions are formal models, prototype implementation of the presented methods and an evaluation experiment. Our data and our implementation are publicly available on the Web. The first experiment evaluated performance of the presented methods and compared them with other machine learning

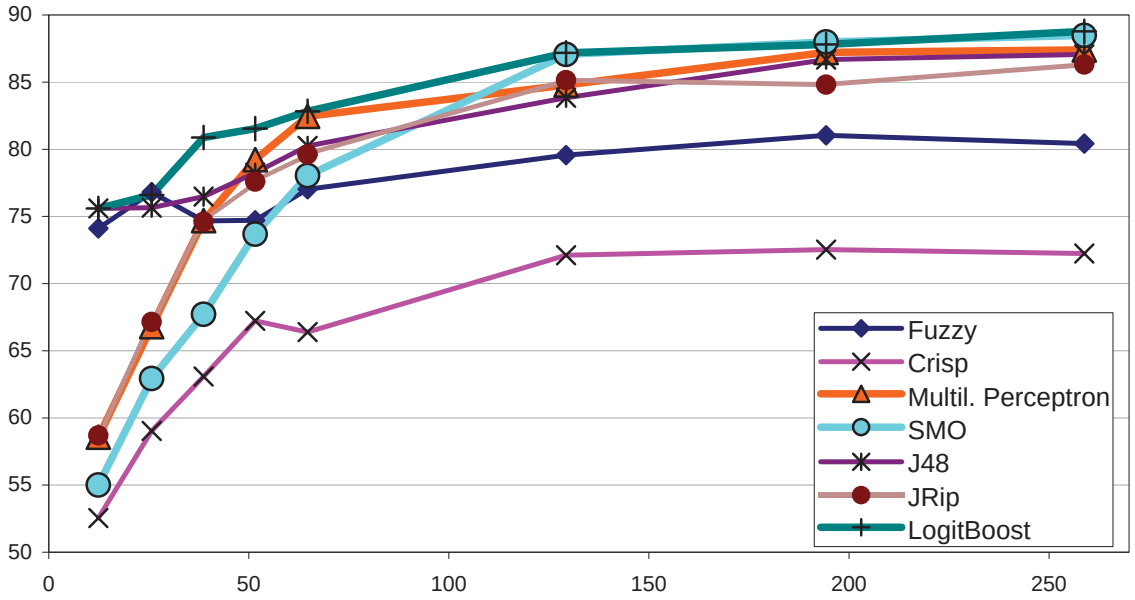


Figure 8.8: x-axis: number of training instances, y-axis: percent of correctly classified instances, average values from 10 repetitions, 'nursery' dataset.

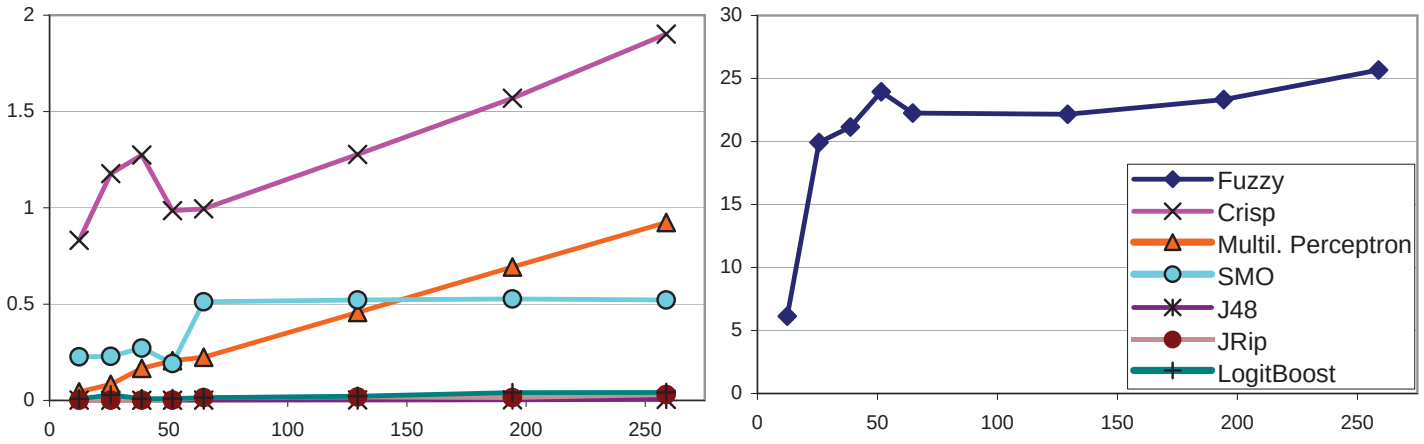


Figure 8.9: x-axis: number of training instances, y-axis: training time in seconds, average values from 10 repetitions, 'nursery' dataset.

procedures used in data mining on our data. 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 has shown 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 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.

Chapter 9

Conclusion

9.1 Extraction Methods or Classification Methods?

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

9.2 Repeatability of Experiments

Our implementation is publicly available. The data, source codes and a platform-independent installer of the Crisp and Fuzzy ILP classifiers for Weka can be downloaded from our Fuzzy ILP classifier's web page[1]. This makes our experiment repeatable according to the SIGMOD Experimental Repeatability Requirements [Manolescu *et al.*, 2008].

[1] `http://www.ksi.mff.cuni.cz/~dedek/fuzzyILP/`

Nomenclature

API	Application Programming Interface
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
FS	Feature Structure, see Section 3.2.5
GAP	Generalized Annotated Programs, see Section 8.4.1
GATE	General Architecture for Text Engineering, see Section 3.4
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
IE	Information Extraction
ILP	Inductive Logic Programming
JAPE	Java Annotation Patterns Engine http://gate.ac.uk/userguide/chap:jape
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 3.1
PEDT	Prague English Dependency Treebank
PML	Prague Markup Language, see Section 3.2.5
POS	Part of Speech
PR	Processing Resource (in GATE) http://gate.ac.uk/userguide/chap:creole-model

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
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/

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.

Harith Alani, Sanghee Kim, David E. Millard, Mark J. Weal, Wendy Hall, Paul H. Lewis, and Nigel R. Shadbolt (2003), Automatic Ontology-based Knowledge Extraction from Web Documents, *IEEE Intelligent Systems*, 18:14–21.

Douglas E. Appelt and David J. Israel (1999), Introduction to Information Extraction Technology, A tutorial prepared for IJCAI-99, Stockholm, Schweden, URL http://www.ai.sri.com/~appelt/ie-tutorial/IJCAI99.pdf.

Tim Berners-Lee (2008), The Web of Things, *ERCIM News - Special: The Future Web*, 72:3, URL http://ercim-news.ercim.org/content/view/343/533/.

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.

J.C. Bioch and V. 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.

K. Bontcheva, V. Tablan, D. Maynard, and H. 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/.

Paul Buitelaar, Philipp Cimiano, Anette Frank, Matthias Hartung, and Stefania Racioppa (2008), Ontology-based information extraction and integration from heterogeneous data sources, *Int. J. Hum.-Comput. Stud.*, 66(11):759–788, ISSN 1071-5819, doi:http://dx.doi.org/10.1016/j.ijhcs.2008.07.007.

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.

Chia-Hui Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan (2006a), A Survey of Web Information Extraction Systems, *Knowledge and Data Engineering, IEEE Transactions on*, 18(10):1411–1428, URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1683775.

Chia-Hui Chang, Mohammed Kayed, Moheb R. Girgis, and Khaled F. Shaalan (2006b), A Survey of Web Information Extraction Systems, *IEEE Trans. Knowl. Data Eng.*, 18(10):1411–1428.

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

Philipp Cimiano, Siegfried Handschuh, and Steffen Staab (2004), Towards the self-annotating web, in *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pp. 462–471, ACM, New York, NY, USA, ISBN 1-58113-844-X, doi:http://doi.acm.org/10.1145/988672.988735.

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.

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.

H. Cunningham, D. Maynard, K. Bontcheva, and V. 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.

Jan Dědek (2010), 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*, pp. 297–304, Springer-Verlag Berlin Heidelberg, Shanghai / China, ISBN 978-3-642-17748-4, URL http://iswc2010.semanticweb.org/accepted-papers/219.

Jan Dědek, Alan Eckhardt, and Peter Vojtáš (2008a), Experiments with Czech Linguistic Data and ILP, in Filip Železný and Nada Lavrač, editors, *ILP 2008 - Inductive Logic Programming (Late Breaking Papers)*, pp. 20–25, Action M, Prague, Czech Republic, ISBN 978-80-86742-26-7, URL http://ida.felk.cvut.cz/ilp2008/ILP08_Late_Breaking_Papers.pdf.

Jan Dědek, Alan Eckhardt, Peter Vojtáš, and Leo Galamboš (2008b), Sémantický Web, in Václav Řepa and Oleg Svatoš, editors, *DATAKON 2008*, pp. 12–30, Brno, ISBN 978-80-7355-081-3, URL http://www.datakon.cz/datakon08/zvane.html#5.

Jan Dědek and Peter Vojtáš (2008a), 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.

Jan Dědek and Peter Vojtáš (2008b), Exploitation of linguistic tools in semantic extraction - a design, in Mieczysław Kłopotek, Adam Przepiórkowski, Sławomir Wierzchoń, and Krzysztof Trojanowski, editors, *Intelligent Information Systems XVI*, pp. 239–247, Academic Publishing House EXIT, Zakopane, Poland, ISBN 978-83-60434-44-4, URL http://iis.ipipan.waw.pl/2008/proceedings/iis08-23.pdf.

Jan Dědek and Peter Vojtáš (2008c), 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*, pp. 85–94, Springer-Verlag, Catania, Italy, ISBN 978-3-540-85256-8, ISSN 1860-949X, URL http://www.springerlink.com/content/w7213j007t416132.

Stephen Dill, Nadav Eiron, David Gibson, Daniel Gruhl, R. Guha, Anant Jhingran, Tapas Kanungo, Sridhar Rajagopalan, Andrew Tomkins, John A. Tomlin, and Jason Y. Zien (2003), SemTag and seeker: bootstrapping the semantic web via automated semantic annotation, in *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pp. 178–186, ACM, New York, NY, USA, ISBN 1-58113-680-3, doi:http://doi.acm.org/10.1145/775152.775178.

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.

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

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.

Ralph Grishman and Beth Sundheim (1996), Message Understanding Conference-6: a brief history, in *Proceedings of the 16th conference on Computational linguistics*, pp. 466–471, Association for Computational Linguistics, Morristown, NJ, USA, doi:http://dx.doi.org/10.3115/992628.992709.

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.

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.

Ian Horrocks (2008), Ontologies and the semantic web, *Commun. ACM*, 51(12):58–67, ISSN 0001-0782, doi:http://doi.acm.org/10.1145/1409360.1409377.

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.

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 *Pro-*

ceedings 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.

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.

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.

Kristina Lerman, Lise Getoor, Steven Minton, and Craig Knoblock (2004), Using the structure of Web sites for automatic segmentation of tables, in *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pp. 119–130, ACM, New York, NY, USA, ISBN 1-58113-859-8, doi:http://doi.acm.org/10.1145/1007568.1007584.

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

Y. Li, K. Bontcheva, and H. 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.

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.

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.

David Pinto, Andrew Mccallum, Xing Wei, and Bruce W. Croft (2003), Table extraction using conditional random fields, in *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pp. 235–242, ACM Press, New York, NY, USA, ISBN 1581136463, doi:10.1145/860435.860479, URL http://dx.doi.org/10.1145/860435.860479.

Borislav Popov, Atanas Kiryakov, Damyan Ognyanoff, Dimitar Manov, and Angel Kirilov (2004), KIM – a semantic platform for information extraction and retrieval, *Nat. Lang. Eng.*, 10(3-4):375–392, ISSN 1351-3249, doi:http://dx.doi.org/10.1017/S135132490400347X.

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 (2008), Us-

ing ILP to construct features for information extraction from semi-structured text, in *ILP'07: Proceedings of the 17th International Conference on Inductive Logic Programming*, pp. 211–224, Springer-Verlag, Berlin, Heidelberg, ISBN 3-540-78468-3, 978-3-540-78468-5.

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.

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.

Hongkun Zhao, Weiyi Meng, Zonghuan Wu, Vijay Raghavan, and Clement Yu (2005), Fully Automatic Wrapper Generation For Search Engines, in *WWW Conference*, pp. 66–75.