Propminer: A Workflow for Interactive Information Extraction and Exploration using Dependency Trees

Alan Akbik, Oresti Konomi and Michail Melnikov
Technische Univeristät Berlin
Databases and Information Systems Group
Einsteinufer 17, 10587 Berlin, Germany
firstname.lastname@tu-berlin.de

Abstract

The use of deep syntactic information such as typed dependencies has been shown to be very effective in Information Extraction. Despite this potential, the process of manually creating rule-based information extractors that operate on dependency trees is not intuitive for persons without an extensive NLP background. In this system demonstration, we present a tool and a workflow designed to enable initiate users to interactively explore the effect and expressivity of creating Information Extraction rules over dependency trees. We introduce the proposed five step workflow for creating information extractors, the graph query based rule language, as well as the core features of the PROPMINER tool.

1 Introduction

Information Extraction (IE) is the task of generating structured information, often in the form of subject-predicate-object relation triples, from unstructured information such as natural language text. Although there are well-established methods for automatically training extractors from annotated data (Mintz et al., 2009), recent years have seen a renewed interest in manually created and maintained rule-based IE systems (Doan et al., 2009; Chiticariu et al., 2010). Advantages of such systems include a better transparency and explainability of extraction rules, and the resulting maintainability and customizability of rule sets.

Another trend in IE is to make increasing use of deep syntactic information in extractors (Bunescu and Mooney, 2005), as dependency parsers become faster and more robust on irregular text (Bohnet, 2010).

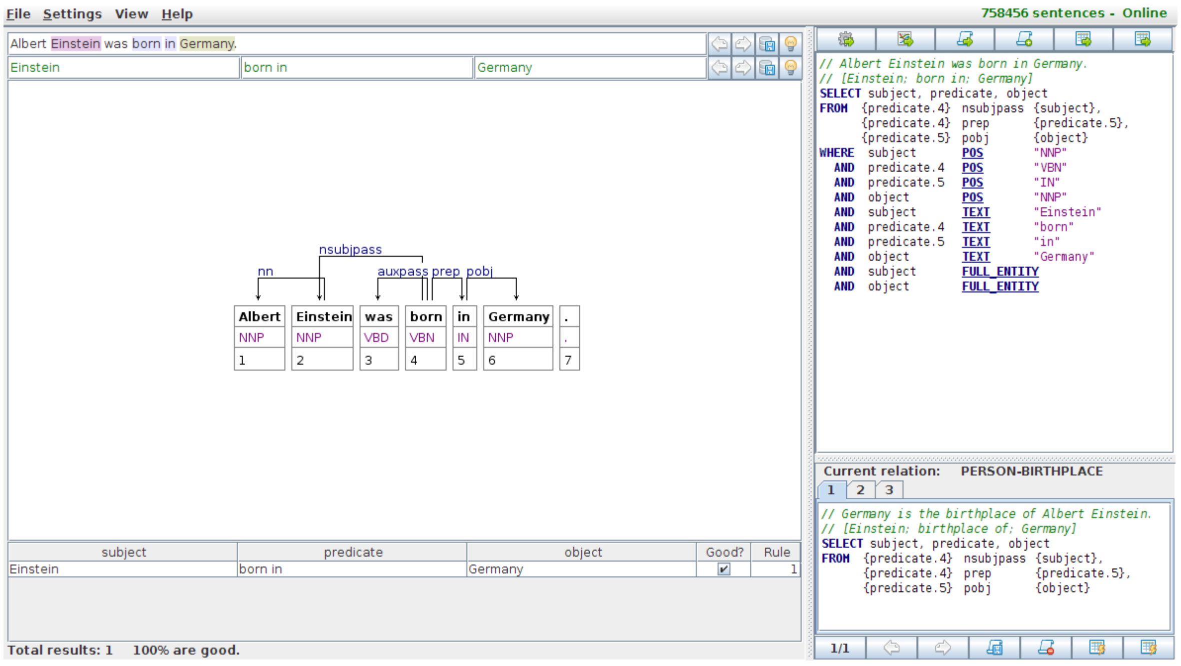
Bringing both trends together are recent works in the field of Open Information Extraction (OIE). The systems KRAKEN (Akbik and Löser, 2012) and CLAUSIE (Del Corro and Gemulla,) use a set of hand crafted rules on dependency trees to outperform previous classification based approaches. The latter system outperforms even OLLIE (Mausam et al., 2012), the machine learning based state-of-the art OIE system on dependency parses. Not only does CLAUSIE report significant precision gains over OLLIE, but also finds 2.5 to 3.5 times more relations.

These results indicate a strong potential for manually creating rule-based Information Extraction systems using dependency trees. The higher level syntactic representation, we argue, may even facilitate rule writing, as - unlike in shallow lexico-syntactic rules - much linguistic variation such as inserted clauses and expressions must not be specifically addressed. This enables the creation of more succinct IE rules, leading to better explainability and easier maintenance.

However, despite these advantages, experience has shown that deep syntactic information is difficult to read and understand for non NLP-experts.

In this system demonstration, we propose a workflow designed to tap into this potential, and present the PROPMINER tool that allows users to execute this workflow. It is specifically designed to help persons familiarize themselves with dependency trees and enable exploration and extraction of relations from parsed document collections. Core features of PROPMINER are:

Rule generation and modification. Initiate users are guided by a workflow in which they first enter and annotate an archetypical sentence with the desired relation. A rule generation process then pre-generates an overspecified rule that users modify along lines suggested by the tool. Our preliminary experiments show that this workflow of generating and modifying rules eases the learning curve for non NLP-experts to concepts such as part-of-speech tags and typed dependencies.



Albert Einstein was born in Germany.

Einstein	born in	Germany

```
// Albert Einstein was born in Germany.
// [Einstein; born in; Germany]
SELECT subject, predicate, object
FROM  {predicate.4} nsubjpass {subject},
      {predicate.4} prep      {predicate.5},
      {predicate.5} pobj      {object}
WHERE  subject       POS       "NNP"
  AND  predicate.4   POS       "VBN"
  AND  predicate.5   POS       "IN"
  AND  object        POS       "NNP"
  AND  subject       TEXT      "Einstein"
  AND  predicate.4   TEXT      "born"
  AND  predicate.5   TEXT      "in"
  AND  object        TEXT      "Germany"
  AND  subject       FULL_ENTITY
  AND  object        FULL_ENTITY
```

subject	predicate	object	Good?	Rule
Einstein	born in	Germany	✓	1

Total results: 1 100% are good.

```
// Germany is the birthplace of Albert Einstein.
// [Einstein; birthplace of; Germany]
SELECT subject, predicate, object
FROM  {predicate.4} nsubjpass {subject},
      {predicate.4} prep      {predicate.5},
      {predicate.5} pobj      {object}
```

Figure 1: Sentence view of PROPMINER, where steps one and two of the workflow are executed. Users enter (or select) a sentence in the top input field and annotate subject, predicate and object for the desired relation. A rule is generated and displayed in the upper right panel. The lower right panel is the repository of already created rules. The parse of the input sentence is displayed in the center panel.

Interactivity and feedback. Each modification of a rule is immediately queried against a large collection of parsed sentences stored in a distributed graph database. The extraction results of the current state of the rule are presented at all times to the user, thereby explaining the rule by showing its effect.

Intuitive query language. Extraction rules are formulated as queries against a graph database. Our query language allows users to formulate subtree queries as *path expressions*, a concept borrowed from the SerQL query language (Broekstra and Kampman, 2003) because of its intuitive properties. We show a visualization of the parse tree of the current sentence next to the generated rule to ease users into understanding the query language (see Figure 1).

Guided workflow. All structured information generated by the user, such as extraction rules, sentence annotations and evaluation results, are stored to build up a repository of structured information. This information is used to suggest appropriate actions to the user.

A preliminary study shows that users without any NLP background are quickly able to use PROPMINER to create Information Extraction rules. We noted that users at first stay true to the workflow and limit manual effort to generalizing rules, but tend to more directly modify extraction rules as they grow more experienced. Furthermore, PROPMINER's interactive nature eases the process of understanding typed dependencies and enables the interactive exploration of parsed document collections.

2 Workflow and Query Language

PROPMINER implements a workflow that consists of five steps (*Annotate*, *Generate*, *Generalize*, *Evaluate* and *Store*). It is designed to allow users that are unfamiliar with syntactic annotation to create rule-based extractors. In the following subsections, we explain the five steps in detail. As a running example, we use the task of creating an extractor for the PERSONBIRTHPLACE relation.

2.1 Annotate

Users begin the process by constructing an *archetypical* sentence for the desired information type. This sentence constitutes an example that expresses the desired relation. For instance, a user interested in the PERSONBIRTHPLACE relation can choose a sentence such as "*Albert Einstein was born in Germany.*".

In this sentence, the user annotates the words belonging to the relation triple, assigning the roles of *subject*, *predicate* and *object*. Subject and object are the entities in the example between which the relation holds. The predicate are the words in the sentence that express the relationship. Accordingly, the user marks "*Albert Einstein*" and "*Germany*" as subject and object, and "*born in*" as predicate in the example sentence.

Figure 1 shows the *sentence view* of PROPMINER, with the example sentence entered and annotated in the top input fields, and the parsed sentence shown in the center panel.

2.2 Generate

PROPMINER generates a rule from the annotated sentence by determining the minimal subtree in the sentence's dependency tree that connects all words labeled as subject, predicate and object. The rule consists of this minimal subtree, as well as constraints in the part-of-speech (POS) tags and lexical values of all involved words.

Rules are formulated as queries against a database in which parsed sentences are stored as graphs: Nodes represent words and edges represent typed dependencies. At each node, the POS tag and the lexical value of the word are stored as attributes.

A PROPMINER rule (or query) consists mainly of three parts: A SELECT clause, a FROM clause and a WHERE clause. The generated rule for the running example is displayed in Figure 1. Its individual parts are discussed in the following subsections.

2.2.1 SELECT and FROM

The SELECT clause determines the fields of tuple to be returned by the query. Typically, this consists of a subject-predicate-object triple, but queries with fewer or more fields are possible.

The FROM clause is a *path expression* that specifies the subgraph in the dependency tree the rule must match, and defines which nodes in the subgraph correspond to the fields in the SELECT clause. A path expression is a set of node-edge-node triples. Each of these triples defines one edge (typed dependency) that must hold between two nodes (words). The nodes are denoted in curly brackets, where the text inside curly brackets assigns a variable name to the node.

Consider the SELECT and FROM clauses for the rule generated for the running example, illustrated in the following:

```
SELECT subject, predicate, object
FROM
    {predicate.3}  nsubjpass  {subject},
    {predicate.3}  prep       {predicate.4},
    {predicate.4}  pobj       {object}
```

Here, the SELECT statement defines the desired result of this query, namely a tuple with a "subject", "object" and a "predicate" field: The path expression in this example is specified in the three lines in the FROM statement. It defines a subtree that consists of four nodes connected by three typed dependencies.

The nodes are assigned the variable names "subject", "object", "predicate.3" and "predicate.4". The node "subject" is defined to be a passive subject (typed dependency "*nsubjpass*") of the node "predicate.3". The node "predicate.3" is also connected via the dependency "*prep*" to the node "predicate.4", which in turn is connected to "object" with the dependency "*pobj*".

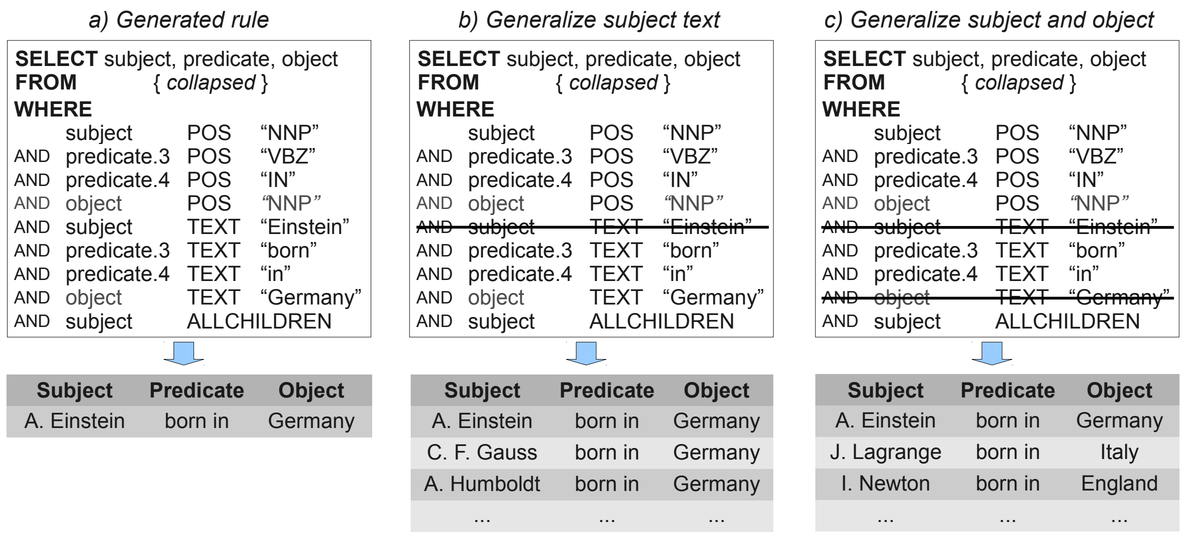
If this rule matches, the lexical values of the matching nodes are returned. Because the predicate in this example consists of two words ("born" and "in"), two nodes are assigned the "predicate" value, subtyped per convention with a dot and a number ("predicate.3" and "predicate.4").

2.2.2 WHERE

In the WHERE-clause, the attributes of words in the subtree can be further restricted. Auto-generated rules are maximally restricted. The rule for the running example is initially restricted as follows:

```
WHERE
 AND subject      POS  "NNP"
 AND predicate.3  POS  "VBN"
 AND predicate.4  POS  "IN"
 AND object       POS  "NNP"
 AND subject      TEXT "Einstein"
 AND predicate.3  TEXT "born"
 AND predicate.4  TEXT "in"
 AND object       TEXT "Germany"
 AND subject      FULL_ENTITY
```

Word attributes are restricted by naming the variable followed either by "POS" or "TEXT" and the restricting value. Here, for instance, the POS tag of the "object" node is restricted to "NNP" (a proper noun), and its lexical value is restricted to "Germany".



a) Generated rule

```
SELECT subject, predicate, object
FROM         { collapsed }
WHERE
     subject      POS      "NNP"
AND  predicate.3  POS      "VBZ"
AND  predicate.4  POS      "IN"
AND  object       POS      "NNP"
AND  subject      TEXT     "Einstein"
AND  predicate.3  TEXT     "born"
AND  predicate.4  TEXT     "in"
AND  object       TEXT     "Germany"
AND  subject      ALLCHILDREN
```

Subject	Predicate	Object
A. Einstein	born in	Germany

b) Generalize subject text

```
SELECT subject, predicate, object
FROM         { collapsed }
WHERE
     subject      POS      "NNP"
AND  predicate.3  POS      "VBZ"
AND  predicate.4  POS      "IN"
AND  object       POS      "NNP"
~~AND  subject      TEXT     "Einstein"~~
AND  predicate.3  TEXT     "born"
AND  predicate.4  TEXT     "in"
AND  object       TEXT     "Germany"
AND  subject      ALLCHILDREN
```

Subject	Predicate	Object
A. Einstein	born in	Germany
C. F. Gauss	born in	Germany
A. Humboldt	born in	Germany
...	...	...

c) Generalize subject and object

```
SELECT subject, predicate, object
FROM         { collapsed }
WHERE
     subject      POS      "NNP"
AND  predicate.3  POS      "VBZ"
AND  predicate.4  POS      "IN"
AND  object       POS      "NNP"
~~AND  subject      TEXT     "Einstein"~~
AND  predicate.3  TEXT     "born"
AND  predicate.4  TEXT     "in"
~~AND  object       TEXT     "Germany"~~
AND  subject      ALLCHILDREN
```

Subject	Predicate	Object
A. Einstein	born in	Germany
J. Lagrange	born in	Italy
I. Newton	born in	England
...	...	...

Figure 2: Conceptual example of rule modification through generalization. Below are example relation triples found for each rule. Rule a) is generated from the annotated sentence in the running example, and finds only one triple. Rule b) is the same rule without the restriction in the subject text. The rule now finds a number of relation triples in the document collection, representing different entities born in Germany. In Rule c) both subject and object text restrictions are removed. This yields a rule that finds different entities born in any entity.

Additionally, a number of subtree gathering mechanisms can be specified in the WHERE clause. For example, the keyword FULL_ENTITY causes the variable binding for the subject to expand to all children nodes expected to be part of a named entity.

2.3 Generalize

The rule generated in step two of the workflow is strongly overspecified to the annotated sentence; all features, including the shallow syntactic and lexical values of all words in the subtree, are constrained. The resulting rule only finds exact instances of the relations as seen in the archetypical sentence. Refer to Figure 2 a) for an example.

In step three of the workflow, the user generalizes the auto-generated rule with the help of suggestions. Common lines of generalizing rules focus on the WHERE clause; here, users can remove or modify constraints on the attributes of words. For example, by removing the restriction on the lexical value of the subject, the rule is generalized to finding *all entities* that were born in "*Germany*", instead of only entities with the lexical value "*Einstein*". This example is illustrated in Figure 2 b).

The rule can then be further generalized by removing the lexical constraint on the object, yielding the (desired) rule that finds all entities that were born in any location with an entity name. Figure 2 c) shows an example of this rule, as well as example results.

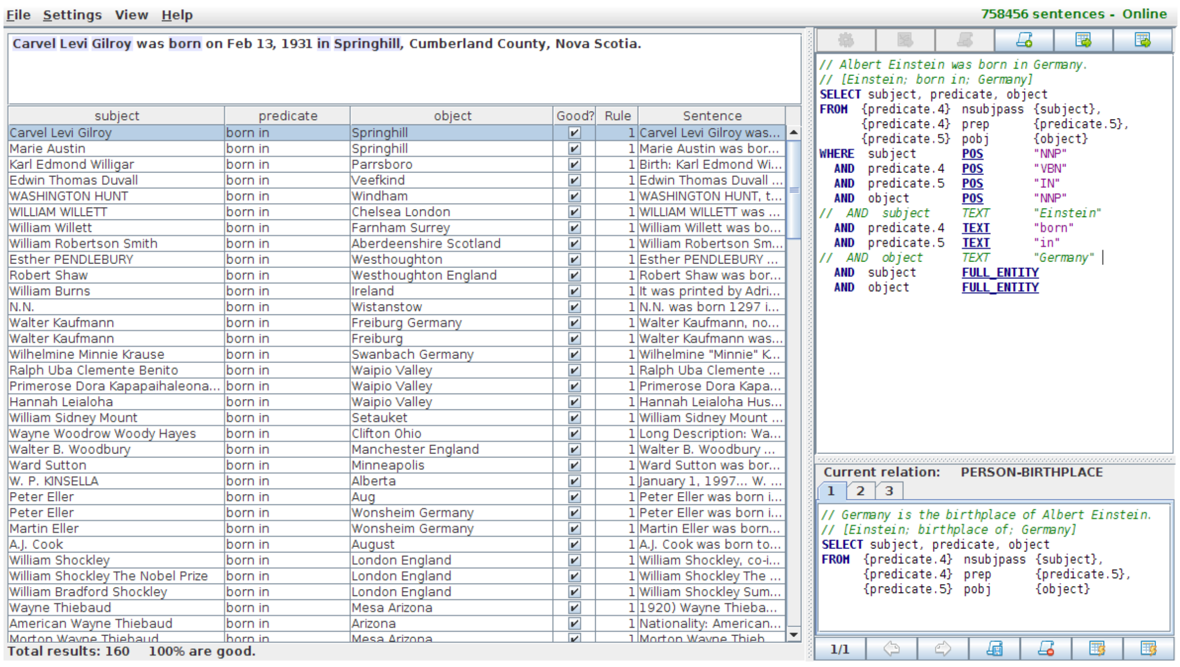
Further options of generalization include removing the lexical constraints in one or both of the predicate words, or modifying the POS tag constraints. At each modification, extraction results for the current state of the rule are displayed to assist the user. When the results match the desired relation, the user can proceed to the next step in the workflow.

2.4 Evaluate

Each rule created by the user is evaluated in the *corpus view* of PROPMINER, displayed in Figure 3. This view shows a sample of extraction results of the rule in a table. The user can scroll through the table and in each row see the extracted information as well as the sentence the information was extracted from. If the extracted information matches the statement in the sentence, the user can mark this fact as correct.

2.5 Store

If the user is satisfied with the extraction rule, he can assign it to a relation and store it in the rule repository. He can repeat the process with another sentence to find more patterns for the desired relation. As the workflow is repeated, the rule repository will build up, along with a repository of evalu-



Carvel Levi Gilroy was born on Feb 13, 1931 in Springhill, Cumberland County, Nova Scotia.

subject	predicate	object	Good?	Rule	Sentence
Carvel Levi Gilroy	born in	Springhill	✓	1	Carvel Levi Gilroy was...
Marie Austin	born in	Springhill	✓	1	Marie Austin was bor...
Karl Edmond Willigar	born in	Parrsboro	✓	1	Birth: Karl Edmond Wi...
Edwin Thomas Duvall	born in	Veefkind	✓	1	Edwin Thomas Duvall ...
WASHINGTON HUNT	born in	Windham	✓	1	WASHINGTON HUNT, t...
WILLIAM WILLETT	born in	Chelsea London	✓	1	WILLIAM WILLETT was ...
William Willett	born in	Farnham Surrey	✓	1	William Willett was bo...
William Robertson Smith	born in	Aberdeenshire Scotland	✓	1	William Robertson Sm...
Esther PENDLEBURY	born in	Westhoughton	✓	1	Esther PENDLEBURY ...
Robert Shaw	born in	Westhoughton England	✓	1	Robert Shaw was bor...
William Burns	born in	Ireland	✓	1	It was printed by Adri...
N.N.	born in	Wistanstow	✓	1	N.N. was born 1297 i...
Walter Kaufmann	born in	Freiburg Germany	✓	1	Walter Kaufmann, no...
Walter Kaufmann	born in	Freiburg	✓	1	Walter Kaufmann was...
Wilhelmine Minnie Krause	born in	Swanbach Germany	✓	1	Wilhelmine "Minnie" K...
Ralph Uba Clemente Benito	born in	Waipio Valley	✓	1	Ralph Uba Clemente ...
Primerose Dora Kapapaihaleona...	born in	Waipio Valley	✓	1	Primerose Dora Kapa...
Hannah Leialoha	born in	Waipio Valley	✓	1	Hannah Leialoha Hus...
William Sidney Mount	born in	Setauket	✓	1	William Sidney Mount ...
Wayne Woodrow Woody Hayes	born in	Clifton Ohio	✓	1	Long Description: Wa...
Walter B. Woodbury	born in	Manchester England	✓	1	Walter B. Woodbury ...
Ward Sutton	born in	Minneapolis	✓	1	Ward Sutton was bor...
W. P. KINSELLA	born in	Alberta	✓	1	January 1, 1997... W. ...
Peter Eller	born in	Aug	✓	1	Peter Eller was born i...
Peter Eller	born in	Wonsheim Germany	✓	1	Peter Eller was born i...
Martin Eller	born in	Wonsheim Germany	✓	1	Martin Eller was born...
A.J. Cook	born in	August	✓	1	A.J. Cook was born to...
William Shockley	born in	London England	✓	1	William Shockley, co-i...
William Shockley The Nobel Prize	born in	London England	✓	1	William Shockley The ...
William Bradford Shockley	born in	London England	✓	1	William Shockley Sum...
Wayne Thiebaud	born in	Mesa Arizona	✓	1	1920) Wayne Thieba...
American Wayne Thiebaud	born in	Arizona	✓	1	Nationality: American...
Morton Wayne Thiebaud	born in	Mesa Arizona	✓	1	Morton Wayne Thieb

Total results: 160   100% are good.

```
// Albert Einstein was born in Germany.
// [Einstein; born in; Germany]
SELECT subject, predicate, object
FROM  {predicate.4} nsubjpass {subject},
      {predicate.4} prep      {predicate.5},
      {predicate.5} pobj      {object}
WHERE  subject       POS        "NNP"
  AND  predicate.4   POS        "VBN"
  AND  predicate.5   POS        "IN"
  AND  object        POS        "NNP"
//  AND  subject     TEXT       "Einstein"
  AND  predicate.4   TEXT       "born"
  AND  predicate.5   TEXT       "in"
//  AND  object      TEXT       "Germany"
  AND  subject       FULL_ENTITY
  AND  object        FULL_ENTITY
```

Current relation: PERSON-BIRTHPLACE

```
// Germany is the birthplace of Albert Einstein.
// [Einstein; birthplace of; Germany]
SELECT subject, predicate, object
FROM  {predicate.4} nsubjpass {subject},
      {predicate.4} prep      {predicate.5},
      {predicate.5} pobj      {object}
```

Figure 3: Corpus view of PROPMINER, where extraction rules are modified and evaluated. The center panel is a table that holds the extraction results for the current rule. Users can inspect each extracted triple by clicking on the row. This will display the sentence in which the triple was found.

ation results. This enables additional functionality in subsequent executions of the workflow:

Sentence suggestions. Evaluation results are used to assist the user in finding new sentences that might be relevant to the relation. For example, a user might mark a triple with the subject "C. F. Gauss" and object "Germany" as a correct instance of the PERSONBIRTHPLACE relation during evaluation. PROPMINER uses this information to retrieve all sentences that contain these two entities from its database. These sentences are treated as probable candidates for containing the PERSONBIRTHPLACE relation, because they contain two entities known to be in this relationship. Accordingly, they are suggested to the user upon request.

Conflict resolution. In order to prevent conflicts with existing rules, the entire rule set in the repository is applied to each sentence the workflow is started with. If any existing information extraction rule can be applied, the results of the extraction are presented to the user as annotations in the sentence. If this extraction result is already complete from the point of view of the user, he can proceed to a new sentence. If not, the user can proceed to generate a new rule, or modify existing ones.

3 Previous Work

Previous work on improving the rule creation process for IE systems has mainly focused on assisting users with machine learning techniques, such as pre-generation of regular expressions (Brauer et al., 2011) or pattern suggestions (Li et al., 2011). To improve usability, (Li et al., 2012) present a tool with a wizard-like environment to guide extractor development. While previous work focuses on shallow patterns, the focus of PROPMINER is to help create rules over dependency trees and aid in the exploration of parsed document collections.

4 Evaluation and Outlook

We conducted a preliminary study in which we asked 5 computer scientists unfamiliar with computational linguistics to use the tool to create extractors for the relations PERSONBIRTHPLACE, PERSONMARRIEDTOPERSON and PERSONWONPRIZE. The participants were given a two hour introduction explaining information extraction and subject-predicate-object triples. We introduced them to the five step workflow using the PERSONBIRTHPLACE example also used as running example in this paper, as well as other, more complex examples. The participants were given one hour for each relation and asked to create a rule set for each relation. After the conclusion we interviewed the participants and asked them to rate the usability both for information extraction, as well as for the exploration of dependency tree information.

In the latter category, participants generally gave positive feedback. Participants stated that the interactive nature of the tool helped understanding extraction rules and facilitated exploring information stated in the document collection. 4 out of 5 participants deviated from the suggested workflow and more directly edited rules as they became more comfortable with the tool. All participants consulted information on POS tags and typed dependencies during the process, in order to better understand the rules and query results. Participants suggested adding an explanation function for individual syntactic elements to the tool.

While all users were generally able to create rule sets for each of the relations, two main problems were cited for the creation of extraction rules. The first is a problem in conflict resolution; in some cases, users were not able to discern why a rule gave imperfect extraction results. We reviewed some rules and found that many of these cases stem from faulty dependency parses, which non NLP-experts cannot recognize. At present, we are searching for ways to address this problem.

A second problem were limitations of the rule language: Participants expressed the need for named entity types such as PERSON and LOCATION, which in the prototype were not included at the time of evaluation. However, because of the design of the query language and the underlying graph database, such additional operators can be incorporated easily.

Consequently, current work focuses on extending the range of user studies to gather more suggestions for the query language and the feature set, and integrating additional operators into the system.

5 Demonstration

In this demonstration we show how PROPMINER can be used for creating extractors or exploring the parsed document collection. The hands-on demonstration allows initiate users to execute the workflow presented in this paper, but also enables persons more familiar with syntactic annotation to more directly query the graph database using our query language and feature set.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments. Alan Akbik received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no ICT-2009-4-1 270137 'Scalable Preservation Environments' (SCAPE).

References

Alan Akbik and Alexander Löser. 2012. Kraken: N-ary facts in open information extraction. In *AKBC-WEKEX*, pages 52–56. Association for Computational Linguistics.

Bernd Bohnet. 2010. Very high accuracy and fast dependency parsing is not a contradiction. In *COLING*, pages 89–97. Association for Computational Linguistics.

Falk Brauer, Robert Rieger, Adrian Mocan, and Wojciech M Barczynski. 2011. Enabling information extraction by inference of regular expressions from sample entities. In *CIKM*, pages 1285–1294. ACM.

Jeen Broekstra and Arjohn Kampman. 2003. Serql: a second generation rdf query language. In *Proc. SWAD-Europe Workshop on Semantic Web Storage and Retrieval*, pages 13–14.

Razvan C Bunescu and Raymond J Mooney. 2005. A shortest path dependency kernel for relation extraction. In *EMNLP*, pages 724–731. Association for Computational Linguistics.

Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick R Reiss, and Shivakumar Vaithyanathan. 2010. Systemt: an algebraic approach to declarative information extraction. In *ACL*, pages 128–137. Association for Computational Linguistics.

Luciano Del Corro and Rainer Gemulla. Clausie: Clause-based open information extraction. In *WWW (to appear in 2013)*.

AnHai Doan, Jeffrey F Naughton, Raghu Ramakrishnan, Akanksha Baid, Xiaoyong Chai, Fei Chen, Ting Chen, Eric Chu, Pedro DeRose, Byron Gao, et al. 2009. Information extraction challenges in managing unstructured data. *ACM SIGMOD Record*, 37(4):14–20.

Yunyao Li, Vivian Chu, Sebastian Blohm, Huaiyu Zhu, and Howard Ho. 2011. Facilitating pattern discovery for relation extraction with semantic-signature-based clustering. In *CIKM*, pages 1415–1424. ACM.

Yunyao Li, Laura Chiticariu, Huahai Yang, Frederick R Reiss, and Arnaldo Carreno-fuentes. 2012. Wizie: a best practices guided development environment for information extraction. In *Proceedings of the ACL 2012 System Demonstrations*, pages 109–114. Association for Computational Linguistics.

Mausam, Michael Schmitz, Stephen Soderland, Robert Bart, and Oren Etzioni. 2012. Open language learning for information extraction. In *EMNLP-CoNLL*, pages 523–534.

Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *ACL/IJCNLP. Volume 2-Volume 2*, pages 1003–1011. Association for Computational Linguistics.