Discovering Semantic Relations from the Web and Organizing them with PATTY

Ndapandula Nakashole, Gerhard Weikum, Fabian Suchanek
Max Planck Institute for Informatics, Saarbruecken, Germany
{nnakasho,weikum,suchanek}@mpi-inf.mpg.de

ABSTRACT

PATTY is a system for automatically distilling relational patterns from the Web, for example, the pattern "X covered Y" between a singer and someone else's song. We have extracted a large collection of such patterns and organized them in a taxonomic manner, similar in style to the WordNet thesaurus but capturing relations (binary predicates) instead of concepts and classes (unary predicates). The patterns are organized by semantic types and synonyms, and they form a hierarchy based on subsumptions. For example, "X covered Y" is subsumed by "X sang Y", which in turn is subsumed by "X performed Y" (where X can be any musician, not just a singer).

In this paper we give an overview of the PATTY system and the resulting collections of relational patterns. We discuss the four main components of PATTY's architecture and a variety of use cases, including the paraphrasing of relations, and semantic search over subject-predicate-object triples. This kind of search can handle entities, relations, semantic types, noun phrases, and relational phrases.

1. INTRODUCTION

Ongoing efforts to extract information from Web data have produced large-scale knowledge bases (KBs) [1, 2, 3, 13]. These KBs store information about real-world entities, such as people, cities, or movies. The KBs mostly use the RDF triple format to store the data. Each triple contains a subject, a predicate, and an object. For example, the fact that Amy Winehouse was born in South Gate would be stored as the triple ⟨*Amy_Winehouse, wasBornIn, South_Gate*⟩. The predicates of such triples are called *relations*. Most KBs contain a limited number of "standard" relations such as *wasBornIn* and *isMarriedTo*. However, there are many more relations that are often missing. For example, in the music domain, one might be interested in relations such as *sang*, *coveredSong* and *hadDuetWith*. Before even populating such relations with triples, one has to find which relations exist. With the PATTY project [10, 11, 12], we embarked on automatically mining new relations from the Web.

Mining relations from the Web is difficult, because relationships between entities are expressed in highly diverse and noisy forms in natural-language text. For example, Web sources may use the verbal phrases ⟨*X's voice in Y*⟩ or ⟨*X's performance of the song Y*⟩ to say that a person sang a song. We call these verbal phrases *patterns*, as opposed to the canonical relation *sang*. So the same relation can be expressed with different patterns. Conversely, the same pattern may denote different relations. For example, ⟨*X covered Y*⟩ could refer to a singer performing someone else's song or to a book covering a historic event (e.g., "War_and_Peace covered Napoleonic_Wars").

Understanding the semantic equivalence of patterns and mapping them to canonical relations is the core challenge in relational information extraction (IE). This problem arises both in seed-based distantly supervised IE with explicitly specified target relations, and in Open IE where the relations themselves are unknown a priori and need to be discovered in an unsupervised manner. Comprehensively gathering and systematically organizing patterns for an *open set of relations* is the problem addressed by the PATTY system.

The approach we take in PATTY is to systematically harvest textual patterns from text corpora. We group synonymous patterns into pattern synsets, so that patterns that express the same relationship are grouped together. We organize these synsets into a subsumption hierarchy, where more general relationships (such as *performed*) subsume more special relationships (such as *sang*). PATTY makes use of a generalized notion of *ontologically typed patterns*. These patterns have a type signature for the entities that they connect, as in ⟨⟨*person*⟩ *sang* ⟨*song*⟩⟩. The type signatures are derived through the use of a dictionary of entity-class pairs, provided by knowledge bases like YAGO[13], Freebase [2], or DBpedia[1].

This paper gives an overview of PATTY based on work reported in [10], [11], and [12]. We first present the design of the main components of PATTY's architecture: the pattern extraction, the SOL pattern model, the pattern generalization, and the subsumption mining. We then present various applications that can make use of the PATTY data.

The PATTY collections of relational phrases are freely available at the URL `http://www.mpi-inf.mpg.de/yago-naga/patty/`.

2. SYSTEM OVERVIEW & DESIGN

PATTY takes a text corpus as input and produces a taxonomy of textual patterns as output. PATTY works in four stages:

- **Pattern extraction.** A pattern is a surface string that occurs between a pair of entities in a sentence, thus the first step is to obtain basic textual patterns from the input corpus. We first apply the Stanford Parser [7] to every sentence of the corpus to obtain dependency paths from which textual patterns are extracted.
- **SOL pattern transformation.** The second step is to transform plain patterns into syntactic-ontological-lexical patterns (SOL) patterns thereby enhancing them with ontological types. A SOL pattern is an abstraction of a textual pattern that connects two entities of interest. It is a sequence of words, POS-tags, wildcards, and ontological types. A POS-tag stands for a word of the part-of-speech class such as a *noun, verb, possessive pronoun, etc.* An ontological type is a semantic class name (such as ⟨*singer*⟩) that stands for an instance of that class. An example of a SOL pattern is: ⟨⟨*person*⟩*'s [adj] voice in * ⟨song⟩*⟩.
- **Pattern generalization.** The third step is to generalize the patterns, both syntactically and semantically. In terms of lexico-syntactic generalization, patterns are generalized into a syntactically more general pattern in several ways: by replacing words by POS-tags, by introducing wildcards, or by generalizing the types in the pattern. For semantic generalization, we compute synonyms and subsumptions, based on the set of entity pairs the patterns occur with — *support sets.*
- **Subsumption and synonym mining.** The last step is to arrange the patterns into groups of synonyms and in a hierarchy based on hypernymy/hyponymy relations between patterns. For semantic generalization, the main difficulty in generating semantic subsumptions is that the support sets may contain spurious pairs or be incomplete, thus destroying crisp set inclusions. To overcome this problem, we designed a notion of a *soft set inclusion*, in which one set S can be a subset of another set B to a certain degree. We thus produce a weighted graph of subsumption relations between the patterns. Patterns with perfectly overlapping support sets are grouped into synonym sets (synsets), where each such synset represents a single relation.

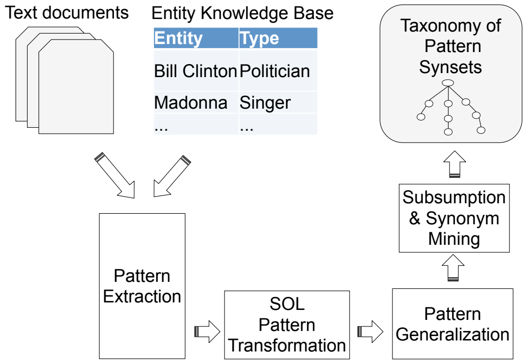


Figure 1: PATTY Architecture

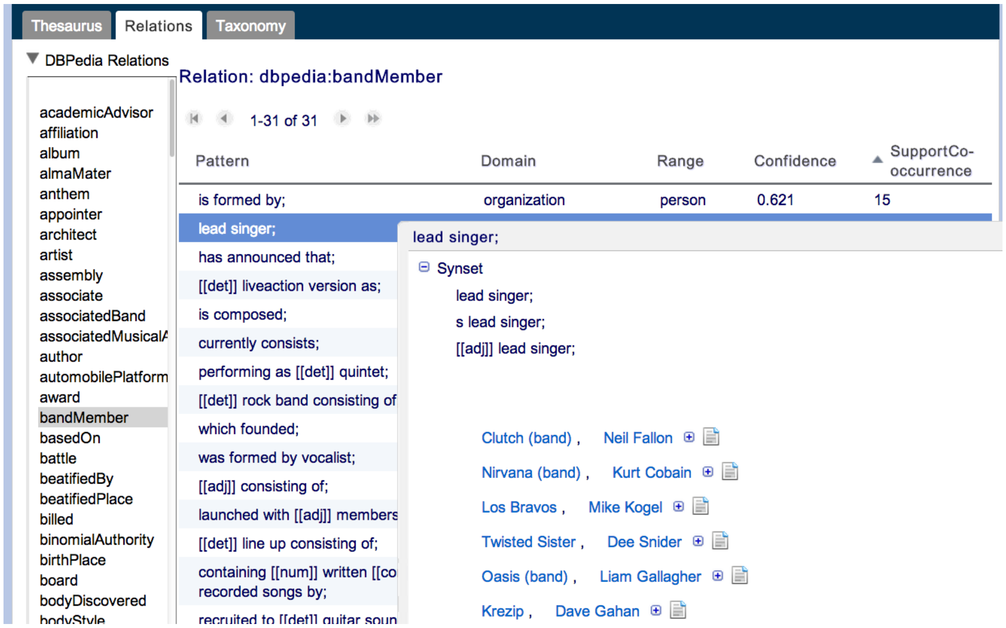
To find entities in the text, and to type them semantically, PATTY requires a pre-defined knowledge base as input. We use either YAGO [13] or Freebase [2]: YAGO has classes derived from Wikipedia categories and integrated with WordNet classes to form a hierarchy of types; Freebase has a handcrafted type system with upper level topical domains as top tier and about entity classes as a second tier. Figure 1 shows the entire PATTY architecture with the role of the knowledge base.

3. IMPLEMENTATION

PATTY is implemented in Java and makes use of the Stanford NLP tool suite for linguistic processing, Hadoop as the platform for large-scale text and data analysis through MapReduce, and MongoDB for storing all resulting data in a key-value representation. The Web-based frontend is running AJAX for asynchronous communication with the server.

Pattern Extraction. The output of pattern extraction are patterns extracted from paths of grammatical dependency graphs, along with the patterns we also output part-of-speech tags of the words from the original sentences. This information is used later for transforming basic patterns into SOL patterns. For distributing pattern extraction with MapReduce, each document is processed independently by the mappers. No coordination is required between concurrent mappers. Thus the input to the mappers are documents from the input corpus. The mapper scans the document, one sentence at a time. If the mapper encounters a sentence with a pair of interesting entities, it emits triples of the form *(*e_1*,* p*,* e_2*)* along with the necessary part-of-speech information. The MapReduce algorithm is outlined in Figure 3.

SOL Pattern Transformation. We take as input the



DBPedia Relations

academicAdvisor
affiliation
album
almaMater
anthem
appointer
architect
artist
assembly
associate
associatedBand
associatedMusicalA
author
automobilePlatform
award
bandMember
basedOn
battle
beatifiedBy
beatifiedPlace
billed
binomialAuthority
birthPlace
board
bodyDiscovered
bodyStyle

Relation: dbpedia:bandMember

Pattern	Domain	Range	Confidence	SupportCo-occurrence
is formed by;	organization	person	0.621	15
lead singer;				
has announced that;				
[[det]] liveaction version as;				
is composed;				
currently consists;				
performing as [[det]] quintet;				
[[det]] rock band consisting of				
which founded;				
was formed by vocalist;				
[[adj]] consisting of;				
launched with [[adj]] members				
[[det]] line up consisting of;				
containing [[num]] written [[co recorded songs by;				
recruited to [[det]] guitar soun				

lead singer;

Synset
lead singer;
s lead singer;
[[adj]] lead singer;

Clutch (band) , Neil Fallon
Nirvana (band) , Kurt Cobain
Los Bravos , Mike Kogel
Twisted Sister , Dee Snider
Oasis (band) , Liam Gallagher
Krezip , Dave Gahan

Figure 2: PATTY paraphrases for the DBPedia relation *bandMember*, the type signature and entities occurring with the relation are also displayed.

```
function map($i$, $d_i$)
  List $S \leftarrow$ all sentences from document $(d_i)$
  for $s \in S$ do
    $NE \leftarrow$ detect named entities in $s$
    if $|NE| > 1$
      $G \leftarrow$ generateDepedencyGraph($s$)
      $P \leftarrow$ dependencyPaths($\forall(e_i, e_j) \in NE$)
      for $p \in P$ do
        emit($e_i, p, e_j, pos$)
```

Figure 3: MapReduce pattern extraction

basic patterns emitted by the pattern extraction module and emit SOL patterns in the form of a sequences of *n-gram* with *type signatures*. To generate SOL patterns from the textual patterns, we decompose the textual patterns into n-grams (n consecutive words) and then generate type signatures for these n-gram patterns.

Frequent N-gram Mining. Only the n-grams that are frequent in the corpus are retained in the SOL patterns, the rest are replaced by wild-cards. The MapReduce algorithm is outlined in Figure 4. Mappers take basic patterns and generate n-grams and emit, for each n-gram, an intermediate key-value pair consisting of the n-gram and a support of 1. The reducers gather support counts for any given n-gram and sum them up to obtain the final support counts. Only those n-grams whose support is above the specified values are emitted. Once we have the frequent n-grams, a second MapReduce algorithm is used to rewrite patterns into a form with frequent n-grams only, disregarding infrequent ones. This way we end up with n-gram patterns. Next, we generate type signatures for the n-gram patterns.

```
function map($i$, $p_i$)
  List $N \leftarrow$ generateNgrams($p_i$)
  for $n_i \in N$ do
    emit($n_i$,1)

function reduce($n_i$, $[v1, v2, v3, ...]$)
  $support \leftarrow 0$
  for $v_i \in [v1, v2, v3, ...]$ do
    $support \leftarrow support + v_i$
  IF $support \geq \gamma$ // where $\gamma$ is minimum support
    emit($n_i$, $support$)
```

Figure 4: MapReduce frequent n-gram mining

Type Signature Generation. For a pattern which is not typed, we can easily compute the occurrence frequencies for each type pair that the pattern occurs with. Based on these initial statistics, we can mine the prevalent type signatures needed to transform type-agnostic patterns into one or more typed patterns.

Given a pattern with type statistics and the entity pairs $(e1, e2)$ in its support set, the key to inferring good type signatures is in the types of entities in a pattern's support set. We take all types that the knowledge base provides for a given entity and use heuristics to eliminate unlikely type signatures. For every $(e1, e2)$, we create two sets, one for all the types of $e1$, T_{e1} and one for all the types of $e2$, T_{e2}. We then compute the cross-product of the two type sets $T(e1)$ and $T(e2)$ with an occurrence frequency of 1. As we iterate over the entity pairs in the support set, we accumulate the occurrence frequencies for every type signature.

This procedure results in a list of possible type signatures for each pattern. The set of candidate signatures is often very large, so we enforce a threshold on the occurrence frequency and drop all signatures below the threshold.

Subsumption & Synonym Mining. Mining subsumptions and synonyms from pattern support sets is not trivial, because a quadratic comparison of each and every pattern support set to every other pattern's support set would be prohibitively slow. Therefore, we developed a Map-Reduce algorithm for this purpose. As input, our algorithm requires a set of patterns and their support sets. As output, we compute a DAG of pattern subsumptions. We first invert the support sets data. Instead of providing, for a pattern, all entity-pairs that occur with it, we provide for an entity pair all the patterns that it occurs with. This can be achieved by a Map-Reduce algorithm that is similar to a standard text indexing Map-Reduce algorithm.

From this data, we have to compute co-occurrence counts of patterns, i.e., the number of entity-pairs that the supports of two patterns have in common. Our Map-Reduce algorithm for this purpose is as follows: The mappers emit pairs of patterns that co-occur for every entity-pair they occur with. The reducers aggregate co-occurrence information to effectively output the sizes of the set intersection of the possible subsumptions. A single machine version of this algorithm is described in [10, 12].

4. RESULTS

We applied PATTY to different corpora to generate relation taxonomies of varying sizes and quality. The version derived from Wikipedia (ca. 3.8 Million articles, version of June 21, 2011) is the richest and cleanest one. It consists of about 350,000 typed-pattern synsets organized in a hierarchy with 8,162 subsumptions.

Precision. Random sampling-based assessment showed that about 85% of the patterns are correct in the sense that they denote meaningful relations with a proper type signature. Furthermore, the subsumptions have a sampling-based accuracy of 83% and 75% for top-ranked and randomly sampled subsumptions respectively. To further evaluate the usefulness of PATTY, we performed a study on relation paraphrasing: given a relation from a knowledge base, identify patterns that can be used to express that relation. We found paraphrasing accuracy to vary from relation to relation: in some cases as low as 53%, and in others as high as 96%, the results are shown in Table 1 with 0.9-confidence Wilson score interval. A random sample of 1000 paraphrases showed an average precision of 0.76 ± 0.03 across all relations.

Recall. Without a reference resource in the form of a comprehensive collection of relations, their synonyms and subsumptions, evaluating recall is not truly possible. We estimated recall by manually compiling an approximate reference resource in the music domain. The reference resource contains all binary relations between entities that appear in Wikipedia articles about musicians. Out of 169 ground-truth relations, PATTY contains 126.

Scalability. In terms of run-times, the most expensive part is pattern extraction, where we identify pattern candidates through dependency parsing and perform entity recognition on the entire corpus. This phase runs about a day for Wikipedia on a Hadoop cluster with ten Dell PowerEdge R720 machines and a10 GBit Ethernet connection. Each machine has 64GB of main memory, eight 2TB SAS 7200 RPM hard disks, and two Intel Xeon E5-2640 6-core CPUs. On the same cluster, all other phases take less than an hour to execute.

5. APPLICATIONS

The data produced by PATTY is a valuable resource for a variety of applications. First, it can boost IE and knowledge base population tasks by its rich and clean repository of paraphrases for the relations. Second, it can improve Open IE by associating type signatures with patterns. Third, it can help to discover "Web witnesses" when assessing the truthfulness of search results or statements in social media [5]. Last, it provides paraphrases for detecting relationships in keyword queries, thus lifting keyword search to the entity-relationship level. This can help to understand questions and text snippets in natural-language QA.

We developed a front-end to the PATTY data for exploring these possibilities in three ways: (1) using PATTY as a thesaurus to find paraphrases for relations, (2) using PATTY as a simple kind of QA system to query the database without having to know the schema, and (3) exploring the relationships between entities, as expressed in the textual sources. The Web-based front-end is running AJAX for asynchronous communication with the server.

5.1 Using PATTY as a Thesaurus

PATTY connects the world of textual surface patterns with the world of predefined RDF relationships. Users who are aware of RDF-based knowledge bases can explore how RDF relations map to their textual representa-

Relation	Paraphrases	Precision
DBPedia/artist [*musical_composition* × *musician*]	83	0.96±0.03
DBPedia/associatedBand [*musician* × *organization*]	386	0.74±0.11
DBPedia/doctoralAdvisor [*person* × *person*]	36	0.558±0.15
DBPedia/recordLabel [*musician* × *organization*]	113	0.86±0.09
DBPedia/riverMouth [*river* × *location*]	31	0.83±0.12
DBPedia/team [*athlete* × *team*]	1,108	0.91±0.07
YAGO/actedIn [*actor* × *movie*]	330	0.88±0.08
YAGO/created [*entity* × *entity*]	466	0.79±0.10
YAGO/isLeaderOf [*person* × *organization*]	40	0.53±0.14
YAGO/holdsPoliticalPosition [*person* × *person*]	72	0.73±0.10

Table 1: Relation Paraphrasing Precision for Sample DBPedia and YAGO Relations

tions. For example, as shown in Figure 2, PATTY knows about 30 ways in which the DBPedia relation *bandMember* can be expressed textually. We hope that this wealth of data can inspire new applications in information extraction, QA, and text understanding.

Users do not need to be familiar with RDF in order to use PATTY. For example, users can find different ways to express the *hasAcademicAdvisor* relation, simply by typing "worked under" into the search box. PATTY also provides the text snippets where the mention was found as a proof of provenance. These text snippets can be explored to understand the context in which a pattern can have a certain meaning. In addition, users can browse the different meanings of patterns, as they occur with different types of entities.

5.2 Schema-Agnostic Search

Internally, PATTY stores all extracted patterns with their support sets. This allows users to search for facts in the database. For this purpose, the PATTY front-end provides a search interface where the user can enter Subject-Predicate-Object triples. Different from existing systems, the user does not have to know the schema of the database (i.e., the relations of the fact triples). It is fully sufficient to enter natural language keywords. For example, to find the co-stars of Brad Pitt, the user can type "costarred with" in place of the relation. PATTY will then search not only for the exact words "costarred with" but also automatically use the paraphrases "appeared with", "cast opposite", and "starred alongside". This way the query needs to be issued only once and the user does not need to enter multiple paraphrases. For each result, PATTY can show the textual sources from which it was derived.

The type signatures of the patterns can be used to narrow down the search results according to different semantic types. For example, when searching for a popular subject like Barack Obama or Albert Einstein, the result may span multiple pages. If the user is interested in only one particular aspect of the entity, then the domain of the subject can be semantically restricted. For example, to see what PATTY knows about Albert Einstein in his role as a scientist, the user can restrict the domain of the relation to *scientist*. Such a query returns Einstein's teaching positions, his co-authors, information about his theories, etc.; but it does not return information about his wives or political activities.

These schema-agnostic queries can be extended to simple join queries. This works by filling out multiple triples and linking them with variables, similar to the way SPARQL operates. Different from SPARQL, our system does not require the user to know the relation name or the entity names. For example, to find visionaries affiliated with MIT, it is sufficient to type: *?x vision ?y, ?x ?z MIT*. This will search for people $?x$ who have a vision $?y$ and who stand in some relationship $?z$ with an entity with name *MIT*. These returns figures like Vannevar Bush (The Endless Frontier vision) and Tim Berners-Lee (Web vision).

5.3 Explaining Relatedness

PATTY can also be used to discover relationships between entities [5]. For example, if the user wishes to know how Tom Cruise and Nicole Kidman are related, it is sufficient to type "Nicole Kidman" into the subject box and "Tom Cruise" into the object box. PATTY will then retrieve all semantic relationships between the two, together with the patterns in which this relationship is expressed. For each result, users can click on the source button discover provenance.

This principle can be extended to full conjunctive queries. For example, to find the entity that links Natalie Portman and Mila Kunis, the user can type: *Natalie Portman ?r ?x, Mila Kunis ?s ?x*. This will find all entities $?x$ that link the two actresses, as well as an explanation of how this entity establishes the link. In the example, PATTY finds the movie "Black Swan" for $?x$, and says that both actresses appeared in this movie. As this example shows, PATTY has created an internal, semantic representation of the input text documents, which allows it to answer semi-structured queries. In addition, to generate semantic patterns, PATTY has implicitly summarized the input text documents. Users can exploit and query these summaries.

5.4 Other Use Cases

Recently, followup work has shown successful usage of PATTY for other tasks. In [9], PATTY 's type signatures are used for semantic typing of out-of-knowledge-base entities. Because the type signatures are fined-grained (e.g., musician, journalist, etc.), the application infers more semantically informative types than standard named entity recognition which works with coarse types such as company, person, etc. In [16], PATTY's relation paraphrases are used for question understanding in the challenging task of question answering.

6. RELATED WORK

Recently, [8] and [17] have addressed the mining of equivalent patterns, in order to discover new relations, based on clustering. These approaches are based on building large matrices or inference on latent models. They differ from PATTY in that the issue of identifying subsumptions between patterns has been disregarded. Among prior works, only ReVerb[4] and NELL[3], have made their patterns publicly available. However, the ReVerb patterns for Open IE are fairly noisy and connect noun phrases rather than entities. NELL is limited to a few hundred pre-specified relations. None of the prior approaches knows the ontological types of patterns, to reveal, e.g., that *covered* holds between a musician and a song.

7. FUTURE WORK

There are several avenues for future research that can build on and improve PATTY. We focused on two types of relatedness: synonymy and hypernymy. However, further types of relatedness between binary relations can be extracted. For example, we can also extract antonyms, where one relation is the opposite of another. Some relations have units; so we could extract the units of relations such as *hasHeight, hasRevenue, hasLength (for songs),* etc. In addition, some relations have value constraints, for example, it is not possible for a person's height to be 5 meters. Another line of future work is extracting n-ary relations for $n > 2$. Such relations might be better suited for explaining complex events and causality.

8. REFERENCES

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z.G. Ives: DBpedia: A Nucleus for a Web of Open Data, ISWC/ASWC, pp. 722-735 2007

[2] K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, J. Taylor: Freebase: a Collaboratively Created Graph Database for Structuring Human Knowledge. SIGMOD, pp. 1247-1250, 2008

[3] A. Carlson, J. Betteridge, R.C. Wang, E.R. Hruschka, T.M. Mitchell: Coupled Semi-supervised Learning for Information Extraction, WSDM, pp. 101-110, 2010

[4] A. Fader, S. Soderland, O. Etzioni: Identifying Relations for Open Information Extraction, EMNLP, pp. 1535 - 1545, 2011

[5] L. Fang, A. Das Sarma, C. Yu, P. Bohannon: REX: Explaining Relationships between Entity Pairs. PVLDB 5(3), pp. 241-252, 2011

[6] G. Limaye, S. Sarawagi, S. Chakrabarti: Annotating and Searching Web Tables Using Entities, Types and Relationships. PVLDB 3(1), pp. 1338-1347, 2010

[7] M.-C. de Marneffe, B. MacCartney and C. D. Manning. Generating Typed Dependency Parses from Phrase Structure Parses. LREC, 2006

[8] T. Mohamed, E.R. Hruschka, T.M. Mitchell: Discovering Relations between Noun Categories, EMNLP, pp. 1447-1455, 2011

[9] N. Nakashole, T. Tylenda, G. Weikum: Fine-grained Semantic Typing of Emerging Entities, ACL, *to appear* 2013.

[10] N. Nakashole, G. Weikum, F. Suchanek: PATTY: A Taxonomy of Relational Patterns with Semantic Types, EMNLP, pp.1135 -1145. 2012

[11] N. Nakashole, G. Weikum, F. Suchanek: Discovering and Exploring Relations on the Web. PVLDB 5(10), pp. 1982–1985, 2012

[12] N. Nakashole: Automatic Extraction of Facts, Relations, and Entities for Web-Scale Knowledge Base Population. *PhD Thesis, Saarland University*, 2012

[13] F.M. Suchanek, G. Kasneci, G. Weikum: Yago: a Core of Semantic Knowledge, WWW, pp. 697-706, 2007

[14] P. Venetis, A. Halevy, J. Madhavan, M. Pasca, W. Shen, F. Wu, G. Miao, C. Wu: Recovering Semantics of Tables on the Web, VLDB, pp. 528-538, 2011

[15] W. Wu, H. Li, H. Wang, K. Zhu: Probase: A Probabilistic Taxonomy for Text Understanding, SIGMOD, pp. 481- 492, 2012

[16] M. Yahya, K.Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, W. Weikum: Natural Language Questions for the Web of Data. EMNLP, pp. 379-390, 2012

[17] L. Yao, A. Haghighi, S. Riedel, A. McCallum: Structured Relation Discovery using Generative Models. EMNLP, pp. 1456 -1466, 2011