

A PRODUCTION SYSTEM FOR INFORMATION EXTRACTION BASED ON COMPLETE SYNTACTIC-SEMANTIC ANALYSIS

Starostin A. S. (astarostin@abbyy.com),
Smurov I. M. (ismurov@abbyy.com),
Stepanova M. E. (mstepanova@abbyy.com)

ABBYY, Moscow, Russia

The article presents a mechanism for information extraction from unstructured natural language data. The key feature of this mechanism is that it relies on deep syntactic and semantic analysis of the text. The system takes a collection of syntactic-semantic dependency trees as input and, after processing them, outputs an RDF graph consistent with certain domain ontology.

The mechanism was implemented within a deployable information extraction system, which is a part of ABBYY Compreno technology—a powerful tool for a broad range of NLP-tasks that include machine translation, semantic search and text categorization. The description of the extraction algorithm and the results of the system performance evaluation are given.

Evaluation tests were conducted on the MUC-6 corpus. The overall F-measure we achieved using Compreno technology was 0.83, which is lower than the best results claimed by the researchers using machine learning approaches. Our system is still under development at the moment and we hope to improve its performance in the future. One of the advantages of Compreno technology is that, unlike many statistical approaches, it does not show an abrupt performance drop if the test corpus is changed. Thus Compreno demonstrates little dependence on the exact textual data it receives and therefore might be seen as a more universal and less domain-dependent solution. Our tests on the CoNLL corpus yielded an F-measure of 0.75 with no prior adjustments made.

Key words: information extraction, named entity recognition, syntactic analysis, anaphora and coreference resolution, production rule systems

Introduction

The article describes an information extraction method which is the core of the data mining system that has been in development by ABBYY over the last three years. This system is an integral part of a more universal text analysis technology known as ABBYY Compreno. Its key feature is the ability to perform complete syntactic-semantic analysis of the input text.

At the first stage a given input is analyzed by the Compréno parser [1]. The result is a collection of syntactic-semantic dependency-based parse trees (one tree per sentence). Nodes and edges of each tree are augmented with diverse grammatical and semantic information. The parse tree forest is then used as input for a production system of information extraction rules. The application of the rules results in the formation of an RDF graph consistent with a domain ontology.

In the first section of the article we provide a description of the information extraction mechanism. We briefly describe the input data, the method used to store extracted information, the structure of the extraction rules and the algorithm of their implementation.

The approach we propose demonstrates two significant advantages. Firstly, the availability of syntactic and semantic structure allows us to extract facts as well as entities. Fact extraction rules that rely on the structure of syntactic-semantic trees tend to be laconic yet highly efficient, easily covering most natural language expressions. Secondly, the system shows little dependence on a particular language. Since our parse trees contain language-independent data (like semantic roles or universal semantic classes), many extraction rules are universal and can be used for different languages.

Despite the fact that we use declarative rules in our system, our approach to information extraction cannot be described as a rule-based one, because the syntactic and semantic analysis that precedes the extraction is not based on a set of rules. The sort of analysis performed by the Compréno parser can be defined as model-based: it rests upon a multilevel model of natural language created by linguists and then corpus-trained. Thus it is possible to consider our method hybrid, it being model-based at the first (preparatory) stage and rule-based at the second.

In the second part of the article we provide the results of the tests we conducted to evaluate our system's performance. We used the MUC-6 corpus to run the tests and chose a standard set of information objects (Person, Organization, Location and Time) for evaluation.

Information Extraction Mechanism

The input accepted by the information extraction mechanism is a sequence of syntactic-semantic trees (one tree per sentence). These trees are generated by the Compréno parser during the analysis. Each tree is projective and its nodes in most cases correspond to the words of the respective sentence, although there are some null-nodes with no surface realization. Nodes and edges of a tree are augmented with grammatical and semantic information. More details on the input and the Compréno parser can be found in the full version of the paper available on the conference website, or in [10].

The output of the extraction mechanism is an RDF graph. The idea of RDF (Resource Definition Framework, [9]) is to assign each individual information object a unique identifier and store the information about it in the form of SPO triples. S stands for subject and contains the identifier of an object, P stands for predicate and identifies some property of an object, O stands for object and stores the value

of that property. This value can be either a primitive data type (string, number, Boolean value) or an identifier of another object.

All the RDF data is consistent with an OWL-DL¹ ontology [7] which is predefined and static. Information about situations and events is modelled in a way that is ideologically similar to that proposed by the W3C consortium for defining N-ary relations [2]. The consistency of the extracted information with the domain model is a built-in feature of the system. It is secured, firstly, by the extraction rules syntax and, secondly, by validation procedures that prevent generation of ontologically inconsistent data.

In addition to RDF graph, extraction mechanism generates annotations, i.e. the information that links extracted entities to the respective parts of the original text. The combination of an RDF graph and annotation links will hereinafter be called an annotated RDF graph.

An annotated RDF graph is generated at the very final stage of the information extraction process. Until that we use a more complex structure to store information during the process. This structure can be described as a set of noncontradictory statements about information objects and their properties. Further on we will often call that a “bag of statements”. Running a few steps forward, we have to note that all the statements are generated during the process of information extraction rules’ application.

A bag of statements has several important properties:

1. **Cumulativity.** Statements can be added to but not removed from a bag.
2. **Consistency.** All the statements in a bag are non-contradictory to each other.
3. **Consistency with ontology.** A bag of statement can anytime be converted into an annotated RDF graph consistent with certain ontology.
4. **Transactionality.** Statements are added in groups, and if any statement of a group contradicts other statements from the bag, the addition of the whole group is cancelled.

The final annotated RDF graph can also be viewed as a bag of statements, if each SPO triple and each link from an object to a segment of text is considered a statement about that object. Therefore it is important to point out the difference between our temporary information storage structure (the inner structure) and the final output in the form of an RDF graph.

The main distinction is that the statements from the inner structure can be used to create functional dependencies, i.e. some statements may depend on the presence of others. For instance, we can state that a set of values of a certain object’s property should always contain a set of values of some other property of a different object. If the set of values of the second object is changed, the first object’s property changes as well. We will hereinafter refer to such statements (which use functional dependencies) as *dynamic* statements. Another difference of the inner structure is that it may contain some auxiliary statements that do not comply with the final annotated RDF graph structure and are used only during the extraction process.

¹ The OWL DL language subset that we use is similar to OWL Lite, but we also exploit DisjointWith axiom.

Here is the list of the possible statement types:

1. **Existence statement.** A statement that proclaims the existence of an information object and creates unique identifiers for them.
2. **Class membership statement.** A statement that attributes an object to a certain class in the ontology.
3. **Property statement.** A statement that defines some property of an object.
4. **Annotation statement.** A statement that connects information objects to parts of the original input text.
5. **Anchor statement.** A statement that links information objects to parse tree nodes, which enables us to access these objects again during the extraction process.
6. **Identification statement.** A statement that merges objects which refer to a single real-life entity.
7. **Functional restriction.** A function, returning a Boolean value, which makes it possible to impose additional restrictions on certain groups of objects. After a function has been added to a bag of statements no statement that would make the function false can enter the bag.

Figure 1 contains schematic diagrams of all statements types available in our system. One can see that only statements of four types may be dynamic. Identification, anchor and existence statements may not depend on other statements.

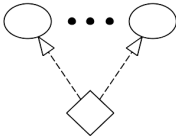
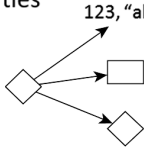
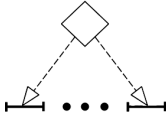
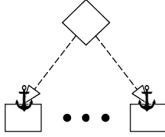
Static statements	Dynamic statements	
create \diamond	classes 	properties 
TheSame(\diamond , \diamond)	annotations 	
anchor 	constraints $f(\diamond \dots \diamond , \square \dots \square) \rightarrow \{0,1\}$	

Fig. 1. Types of statements used in the information extraction process. Diamonds represent information objects (individuals), ellipses represent classes (or concepts) and rectangular boxes represent parse tree nodes

Let us describe anchor statements more thoroughly because they are a very important part of information extraction mechanism. Anchor statements link information objects to parse tree nodes, which enables us to access these objects continuously during the extraction process. The term ‘anchor’ was coined when the system was

in development so that the links between objects and tree nodes could be easily referred to. One object can be anchored to a set of nodes via a number of anchor statements.

The interpreter of the information extraction rules handles these anchors in a special way: the left-hand side (or condition side) of a rule in our system can contain so-called object conditions. These conditions require object(s) with certain properties to be anchored to a node before the rule can be executed. And if the object was found and the production executed, this object can be accessed in the right-hand side of the rule.

Object conditions are most widely used in the rules that extract facts, but they are quite useful with named entities as well, since they make it possible to break the extraction process down to several simple stages. For instance, one rule might only create an unspecified Person entity, while the following ones add properties like first name, surname, middle name and alike. It has also become quite common to create auxiliary objects which serve as dynamic labels of parse tree nodes. First some rules create these auxiliary objects and attach them to certain nodes, and then other rules check for these objects with help of object conditions in their left-hand sides.

Detailed information about other types of statements can be found in the full version of the article, available on the website of the conference.

Information Extraction Rules

Information extraction process is controlled by the production rule system. There are two types of rules in the system: parse subtree interpretation rules (or simply interpretation rules) and identification rules. Since interpretation rules are much more frequent, whenever we do not specify the exact type of rule the reader should assume that it is an interpretation one. Information on both types of rules can be found in the full version of the article, available on the conference website.

During the development of the extraction mechanism several goals were pursued. In the first place, our intention was to exploit such advantages of the production rule systems as modularity [8] and separation of knowledge from the procedure. We particularly wanted to relieve the developers from the necessity to order the rules². Secondly, we intended to implement an efficient deterministic output model. Speaking in terms of traditional production systems [3] we can define parse tree forest and a bag of statement as our knowledge base, while the extraction process itself can be described as a forward chaining inference process.

² One particular example of quasi-production language that does not comply with this requirement is Jape [5]. Jape requires setting the order in which groups of production rules (or phrases) are executed explicitly. During their execution rules within a group do not have the access to each other's results. In the process of development of such rules it often occurs that the rules which create an object of a certain type are executed after the rules which accept such an object as their input. Moreover, it is not possible to reorder the rules because rules from the first group might also use some objects created by the second group. The only solution to this problem is to launch the same groups of rules several times. However, this solution is far from ultimate since it artificially limits the number of recursion steps.

Information Extraction Algorithm

While describing the information extraction algorithm we use the generic term ‘matching’. By this term we mean both matching of a tree template in an interpretation rule with a subtree of an actual parse tree and matching of an identification rule with a certain pair of objects. Formal definition of matching can be found in the full version of the article, available on the conference website. Here we will just point out that finding a matching is a sufficient condition for the right-hand side of the rule to be converted into a set of statements.

The information extraction algorithm has the following steps:

1. Analyze the input text with the Comprono parser to get a forest of syntactic-semantic parse trees.
2. Find all the matchings for the interpretation rules that do not have object conditions.
3. Add the matching to the sorted matching queue
4. If the queue is empty, terminate the process.
5. Get the highest priority matching from the queue
6. Convert the right-hand side of the production a group of statements.
7. Try to add the statements to the bag.
8. If failed, declare the group of statements invalid and go to step 4.
9. Else if succeeded, look for new matchings.
10. If found, add new matches to the queue Go to step 4.

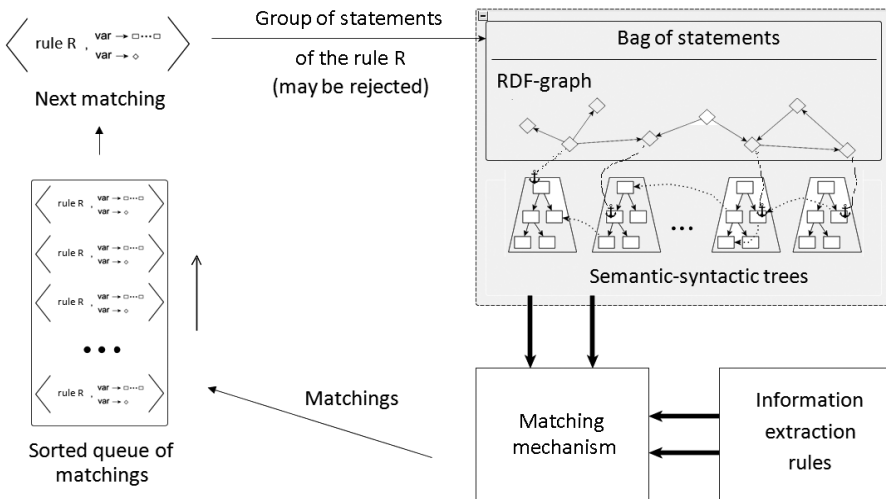


Fig. 2. Schematic representation of the information extraction process

Some parts of the above algorithm need to be described more thoroughly. Steps 2 and 9 are performed with the help of a special matching mechanism. This mechanism can retrieve all the matches for the rules without object conditions. It also constantly monitors the contents of the bag of statements. Every time step 7 is performed successfully and new statements get into the bag, the mechanism takes them into account and, if necessary, generates new matches for the rules that do contain object conditions. These new matchings can be created both for the rules that have already been matched before and for those which remained unmatched until that moment. The former occurs when an object condition of a certain rule is matched by more than one object. In this case each object is matched in a separate matching.

The implementation of the matching mechanism is relatively complex. For one, it has a built-in bytecode interpreter for the compiled rules, a system of indexes for the syntactic-semantic trees, a module for tracking changes in the bag of statements and several other features. Full-length description of this mechanism is beyond the scope of the paper.

It is also important to explain the way the queue of matchings is sorted at the third step. In some cases developers can set the order of rules, i.e. there is partial order over the whole set of rules. Of any two rules one can be given priority over the other. It means that if both rules are ready to be executed, the rule with the higher priority should go first. For convenience reasons we also support group ordering of rules. If group A was given priority over group B, then each rule belonging to group A has higher priority than one belonging to group B. Partial order relation is transitive. Correctness of partial order is checked every time a system of rules is compiled. If loops are detected, compilation fails and the user receives an error message. The order of matching in the queue is always consistent with the partial order set within a system of rules. This approach differs significantly from those with consecutive execution of rules, since partial order only determines the priority of rules and does not prevent repeated execution.

It is easy to see that the described algorithm does not consider alternatives. If some matching is inconsistent with the bag of statements in its current state, it is simply dismissed. We can afford to use this ‘greedy’ principle because our parser performs word-sense disambiguation, so we rarely ever have to hypothesize about a node. There are some exceptions like words unknown to the parser, but for such cases we have special methods of dealing with these words and incorporating them in our model.

Evaluation

We tested our system on the texts that were manually annotated with name entities for the 6th Message Understanding Conference (MUC-6) held in November 1995 [4]. Today the MUC-6 data set is considered one of the main evaluation benchmarks for named entity recognition. You can find the detailed description of the evaluation process in the full version of the article. In the paper version we limit ourselves only to results, which are shown in the table below:

Table 1. Evaluation results

Type of entity	Precision	Recall	F-measure
All entities	0.853	0.813	0.832
Money	0.947	0.933	0.940
Person	0.700	0.887	0.783
Location	0.936	0.806	0.866
Organization	0.767	0.639	0.697
Date	0.941	0.880	0.910
Time	0.674	0.573	0.620

These results are lower than the numbers shown by machine-learning systems on MUC-6 original test corpus of 30 texts (the F-measures of many systems that participated in the contest were higher than 90% and the winner reached 96,42% in F-measure). However it is worth noting that our system was not specifically trained on MUC-6 corpus texts or any other WSJ articles. We also did not make any deliberate changes in our model (apart from the technical ones, see the description in the full version of the article) that could artificially improve performance on this particular set of texts. It would be correct to assume that our system was put in position of a statistical entity extractor trained on a completely different corpus.

Error analysis demonstrated that approximately 60% of the errors were the errors of the Compreno parser, 20% occurred due to flaws in the extraction rules and the MUC-6 corpus inconsistencies accounted for the remaining 20%. These results show that the system has a significant potential for further development, especially since there are hopes to improve the quality of the syntactic-semantic parser.

After testing our system on MUC-6 corpus we also conducted additional tests on CoNLL corpus [6]. During these tests no settings were modified and no changes were made whatsoever. The resulting F-measure was 0,75. This allows us to make a preliminary conclusion that our system is more resistant to the replacement of one corpus with another than systems based on machine-learning approaches. In the near future we intend to conduct a more extensive performance evaluation on several other corpora.

We do realize that the tests we conducted are insufficient to provide complete evaluation of the system performance (and give the reader full insight of the system), especially since the spectrum of its applications is much wider than named entity recognition.

Conclusion

In this paper we described an information extraction mechanism based on a production rule system. The rules are applied to the results of full syntactic-semantic analysis performed by the Compreno parser. The output of the extraction mechanism is an RDF graph consistent with domain ontology and augmented with information about annotations of extracted individuals.

We also presented the idea of storing extracted information as a set of dynamic logical statements. We mentioned two types of declarative extraction rules: interpretation rules that interpret subtrees of syntactic-semantic trees and identification rules

that merge information objects. We gave schematic description of the information extraction algorithm.

A considerable advantage of the system we have created is that a developer of rules does not have to set the order of their execution. Rules are executed in arbitrary order if there is data that matches their left-hand sides. However, if the necessity appears, the developer can set partial rule order.

Finally, we present the results of the evaluation tests we conducted on the MUC-6 manually annotated corpus. Our system demonstrated relatively good performance with no prior adjustments made. Additional tests on the CoNLL corpus allow us to make a preliminary conclusion that our system is not dependent on a particular corpus (like statistical ones often are) and remains efficient after the corpus is changed. To confirm this conclusion further tests are required and we plan to conduct them in the nearest future. After these tests are performed we intend to publish a new article focusing on the task of fact extraction.

References

1. *Anisimovich K. V., Druzhkin K. Ju., Minlos F. R., Petrova M. A., Selegey V. P., Zuev K. A.* (2012), Syntactic and semantic parser based on ABBYY Compreno linguistic technologies, Computational Linguistics and Intellectual Technologies: Proceedings of the International Conference “Dialog” [Komp’iuternaia Lingvistika i Intellektual’nye Tehnologii: Trudy Mezhdunarodnoj Konferentsii “Dialog”], Bekasovo, pp. 90–103.
2. Defining N-ary Relations on the Semantic Web, available at <http://www.w3.org/TR/swbp-n-aryRelations>
3. *Gavrilova T. A., Khoroshevskij V. F.* (2000) Knowledge Bases of Intellectual Systems [Bazy znaniy intellektual’nyh system], Piter, St. Petersburg, Russia
4. *Grishman R., Sundheim B.* (1996), Message Understanding Conference—6: A Brief History, available at: <http://acl.ldc.upenn.edu/C/C96/C96-1079.pdf>.
5. *Karasev V., Khoroshevsky V., Shafirin A.* (2004), New Flexible KRL JAPE+: Development & Implementation, Knowledge-Based Software Engineering. Proceedings of the Sixth Joint Conference on Knowledge-Based Software Engineering, Amsterdam. Language-Independent Named Entity Recognition, available at <http://www.cnts.ua.ac.be/conll2003/ner/>
6. OWL Web Ontology Language Overview, available at <http://www.w3.org/TR/2004/REC-owl-features-20040210>
7. *Pospelov D. A.* (1989) Modelling Reasoning. Experience in the Analysis of Mental Acts [Modelirovanije Rassuzhdenij. Opyt Analiza Myslitel’nyh Aktov]. Radio i Svayz, Moscow, Russia.
8. Resource Description Framework, available at <http://www.w3.org/RDF/>
9. *Zuev K. A., Indenbom M. E., Judina M. V.* (2013), Statistical machine translation with linguistic language model, Computational Linguistics and Intellectual Technologies: Proceedings of the International Conference “Dialog” [Komp’iuternaia Lingvistika i Intellektual’nye Tehnologii: Trudy Mezhdunarodnoj Konferentsii “Dialog”], Bekasovo, vol. 2, pp. 164–172.