SemDupl: Semantic-based Duplicate Identification

Sven Hartrumpf, Hermann Helbig,
Tim vor der Brück, Christian Eichhorn

Arbeitsgruppe Intelligente
Informations- und Kommunikationssysteme
FernUniversität in Hagen
58084 Hagen – Germany

{sven.hartrumpf,hermann.helbig,tim.vorderbrueck}@FernUni-Hagen.de

christian.eichhorn@outtalimits.de

July 2011

FernUniversität in Hagen — Informatik-Bericht

Contents

1 Overview, Introduction and Motivation; Accomplishment of Tasks 5

2 Comparison with other Systems 6

3 Architecture of SemDupl 6

4 The SemDupl Corpus 7

5 The Shallow Duplicate Detectors SemDupl-shallow and SemDupl-quick 9
- 5.1 Features in SemDupl-shallow 9
- 5.2 Combination of Features 10
- 5.3 Training Phase 10
- 5.4 Application Phase 10
- 5.5 Aggregation of Feature Values 11
- 5.6 SemDupl-quick (SQ) 11
- 5.7 Comparison of Shallow Approaches 12

6 Linguistic Phenomena Relevant for Semantic Duplicates 12
- 6.1 Types of Paraphrases for Semantic Duplicates 12
- 6.2 Restrictive Contexts and Other Precision Problems for Semantic Duplicates 15

7 Knowledge Acquisition for Deep Duplicate Detectors 16
- 7.1 Hypernyms and Meronyms 16
- 7.2 Deep vs. Shallow Patterns 19
- 7.3 Validation of Relation Hypotheses 21
- 7.4 System Architecture: Relation Extraction 23

8 Annotation GUI 24

9 Extraction of Entailments 26
- 9.1 Extracting Entailments Employing a Search Engine 26
- 9.2 Extracting Entailments from SNs Basically Following Ravichandran and Hovy 26
- 9.3 Extracting Entailments Basically Following Lin and Pantel 27

10 The Deep Duplicate Detector SemDupl-deep 27

11 Combination 28

12 Evaluation 30

13 Evaluation Interpretation and Conclusions 32

A Extracted Knowledge 34

B Program Documentation and Manuals **44**
B.1 SemDupl-deep . . . 44
B.2 SemDupl-shallow . . . 45
B.3 SemDupl-quick . . . 48
B.4 Combiner . . . 50
B.5 Knowledge Acquisition . . . 50
B.5.1 Hyponyms, Meronyms and Synonyms . . . 50
B.5.2 Entailments . . . 56
B.6 The GUI . . . 60
B.6.1 Text Collection Maintenance . . . 60
B.6.2 Text Collection Analysis . . . 62
B.6.3 Text Duplicate Search . . . 62
B.7 Class Diagram SemDupl-shallow . . . 62
B.7.1 Class *Text* . . . 62
B.7.2 Class *Toolbox* . . . 63
B.7.3 Class Comparer . . . 64
B.7.4 Class *LAUT* . . . 64
B.7.5 Class CErkenner . . . 64

List of Figures

1 Architecture of the duplicate detector SemDupl. . . . 8
2 Data model of SemDupl-quick. . . . 12
3 Shallow pattern for hyponymy extraction where the premise is given as a regular expression. . . . 17
4 Deep pattern for hyponymy extraction where the premise is given as an SN. . . . 18
5 Matching a deep pattern to a semantic network. . . . 18
6 Deep pattern for hyponymy extraction where the premise is given as an SN. . . . 19
7 Application of a deep pattern to an SN representing the sentence *He sells all common string instruments except celli.* . . . 20
8 System Architecture of SemQuire. . . . 23
9 Screenshot of the annotation tool. . . . 24
10 Graphical user interface of SemChecker for displaying the result of the logical validation, which marks the entry SUB(*land.1.1*, *provinz.1.1*) as potentially defective. . . . 25
11 System architecture of the Combiner. . . . 29
12 Precision of hyponymy hypotheses depending on score. . . . 32
13 Screenshot of the GUI for SemDupl. . . . 61
14 Class diagram of SemDupl-shallow. . . . 62

List of Tables

1 Assumed equivalent formulas with given confidence score. . . . 27
2 Confusion matrix for WCopyFind . . . 30
3 Confusion matrices for shallow approaches . . . 31

4 Confusion matrices for deep approaches 31
5 F-Measure, precision, recall, and accuracy for shallow approaches. . . 31
6 F-Measure, precision, recall, and accuracy for deep approaches. . . . 31
7 Extracted semantic and lexical relation hypotheses. 32
8 Lines of source code of the SemDupl components. 32
9 Hypernymy relations with high confidence score. 35
9 Hypernymy relations with high confidence score. 36
10 Meronymy hypotheses with high confidence score. 37
10 Meronymy hypotheses with high confidence score. 38
10 Meronymy hypotheses with high confidence score. 39
11 Synonymy hypotheses with high confidence score. 40
11 Synonymy hypotheses with high confidence score. 41
12 Selection of entailments. 42
12 Selection of entailments (ctd.). 43

1 Overview, Introduction and Motivation; Accomplishment of Tasks

What has been the motivation for the project SemDupl (Semantic Duplicate Detection)? – Duplicate detection becomes more and more important in the information society as the number of available text documents grows rapidly; see for example the growth of the web. At a similar speed, the number of duplicates increases.

Duplicates are relevant in many different areas. In search engines, question answering, and other information access applications, duplicates should not be counted as different documents for scoring purposes and should be excluded from result pages presented to users. Copyright owners want to find cases of copyright violations, even if the violators apply intelligent techniques to hide the origin of their plagiarisms. Desktop administration and backup tools need ways to suggest which files could be unneeded duplicates. Depending on the application, near-duplicates are subsumed under the term duplicates in this paper.

In prior work, duplicate detection worked with shallow checkers; they can be called shallow because they work only on surface-oriented factors or features and do not step into the semantics of words, sentences, paragraphs, or even whole texts. Surface-oriented features of a given text are derived from n-grams, rare words, spelling errors, etc. But two texts can be semantic duplicates, i.e., expressing the same content, without sharing many words or word sequences and hence without having similar values of shallow features. Shallow checkers can be easily tricked by experienced users that employ advanced paraphrase techniques. Therefore, a deep (semantic) approach that compares full semantic representations of two given texts has been designed, implemented, and evaluated in the SemDupl (Semantic Duplicate Detection) project.

To achieve this goal, the following tasks had to be accomplished according to the project proposal:

(A1) Conceptional work In the preparation phase, existing shallow duplicate recognizers had to be compared, with the aim to find one recognizer which can be used as a baseline (see Section 2) and as a starting point for combining shallow and deep methods for duplicate detection (see Section 3).

(A2) Knowledge acquisition This task turned out to be the most ambitious part of the project, since the knowledge acquisition had to be done automatically, and the results are the precondition for a successful work of the deep duplicate recognizer. To this end, thousands of basic relations (subordination relations, meronymy relations and semantic entailments, so-called meaning postulates) had to be automatically found by means of statistical learning methods and to be validated with logical methods (see Section 7 and Section 9).

(A3) Creation of convenient test beds As a basis for checking whether the duplicate recognizer works well, sufficiently large corpora of pairs of duplicate texts and non-duplicates are needed. These corpora had partially been handcrafted and partially gathered from existing collections (see Section 4). Several corpora form the SemDupl corpus, which can be seen as a valuable resource produced in the SemDupl project.

(A4) Textual entailment During the work on the project, it turned out that the theorem prover which had been developed in the LogAnswer project[1] is also well suited for textual entailment. Thus the main work consisted in the generation of a large number of textual entailment patterns, deep as well as shallow ones (see Section 7.2).

(A5) Technical realization of the duplicate recognizer The programs embodying the recognizer have been developed according to the architectures shown in Figure 1 and Figure 11. Their technical handling is described in Appendix B.

(A6) Evaluation The results of the evaluation of the recognizer developed in the SemDupl project are finally described in Section 12.

2 Comparison with other Systems

As stated, detecting duplicates is of high interest for holders of rights and tutors. Therefore, many tools exist to detect plagiarisms in given corpora or the web. Below are some of the best ranked systems according to the 2008 test of the University of Applied Sciences Berlin (HTW) (Weber-Wulff, 2009).

Copyscape [2], a plagiarism checker of Indigo Stream Technologies Ltd. Given a text, it searches the Internet for possible duplicates of this text using the document's words in the given order.

Plagiarism Detector [3] by SkyLine, Inc. uses non-overlapping n-grams with a configurable spacing between them to find online-plagiarisms of a given text in various possible input formats.

Urkund [4] by PrioInfo AB targets to check papers written by students for possible plagiarism and searches the Internet (with known paper mills), an own corpus of scientific publications and papers checked for plagiarism before.

WCopyfind [5] is an n-gram based plagiarism checker of the University of Virginia, Charlottesville (Balaguer, 2009). It targets student's papers, searching a corpus which has to be compiled by the user. Since it is open source software this tool was used as a comparison for our SemDupl system.

3 Architecture of SemDupl

SemDupl is a complex software system, whose top-level architecture is illustrated in Figure 1 (Hartrumpf et al., 2010b,a). (For simplicity, only one of the two shallow duplicate detectors is included in the graph.) The user can access SemDupl via a graphical user interface (GUI) or via a command line interface (CLI); for other application programs, an application program interface (API) is provided.

[1] This project had been funded by DFG under contract number HE 2847/10-1.

[2] `http://www.copyscape.com/`, first and third place (premium and free version)

[3] `http://plagiarism-detector.com/`, scored second place

[4] `http://www.urkund.de/`, scored fourth place

[5] `http://plagiarism.phys.virginia.edu/Wsoftware.html`, marked as "good"

The GUI is designed with two different user groups in mind: (1) developers and testers that use SemDupl on a daily basis, (2) normal users that want to use SemDupl to investigate their own corpus for duplicates. It was implemented in Hop, a Scheme-based language for the Web 2.0, because the main parts of SemDupl (parser, deep detector) are implemented in Scheme, too, and because Hop allows concise and elegant programs that run efficiently across the Internet. The details of the GUI are explained in Appendix B.6.

The main functionality (or mode) is described in Figure 1: testing a single text (the candidate text) against a corpus of texts (the base corpus) in order to decide whether it is a duplicate (and if so, from what texts and in which parts exactly). The candidate text is distributed by the central controller to three duplicate detectors: two shallow ones (SemDupl-shallow, SemDupl-quick), to quickly find the easy cases; and a deep one (SemDupl-deep), to tackle the difficult cases. The results of the three detectors are combined to give a single result with one confidence score for the reported decision.

The shallow detector SemDupl-shallow and the Combiner need corpus statistics derived from an annotated corpus of plagiarism. In addition, lexical-semantic relations are used to allow to follow semantic paraphrasing at least on the word level.

SemDupl-deep is much more complex than the shallow detector SemDupl-shallow: it starts with the syntactico-semantic parser WOCADI[6] (Hartrumpf, 2003), which delivers semantic networks of the MultiNet[7] formalism (Helbig, 2006) and syntactic dependency trees. Only the semantic networks are further processed in SemDupl-deep. The base corpus is completely parsed during a preprocessing step, which is comparable to an indexing step in simpler systems. The semantic representations of the candidate texts are expanded (query expansion) and textual entailments are precalculated by a large rule component. The resulting representations are the input for a semantic matcher, which can only run efficiently over the base corpus due to a specialized index for semantic networks.

The knowledge aspect of SemDupl-deep is depicted in the lower right corner of Figure 1. Automatic knowledge acquisition methods deliver semantic facts and rules from parsed corpora like the German Wikipedia. In addition, knowledge engineers can validate and extend the automatically generated resources.

4 The SemDupl Corpus

The corpus used in the learning process of the Combiner, SemDupl-shallow (see Section 5) and for evaluation purposes is composed of five manually annotated corpora:

RSS news (semdupl-rss) Automatically collected news feed articles of different German media consisting of 99 texts fully annotated with 298 duplicate pairs.

Prose (semdupl-prose) Short stories by Edgar Allen Poe translated to German by different translators (split in 136 parts of about 600 words each) with 200 duplicate pairs.

[6] WOCADI is the abbreviation for **wo**rd **cla**ss based **di**sambiguating parser.

[7] MultiNet is the abbreviation of **Multi**layered Extended Semantic **Net**works

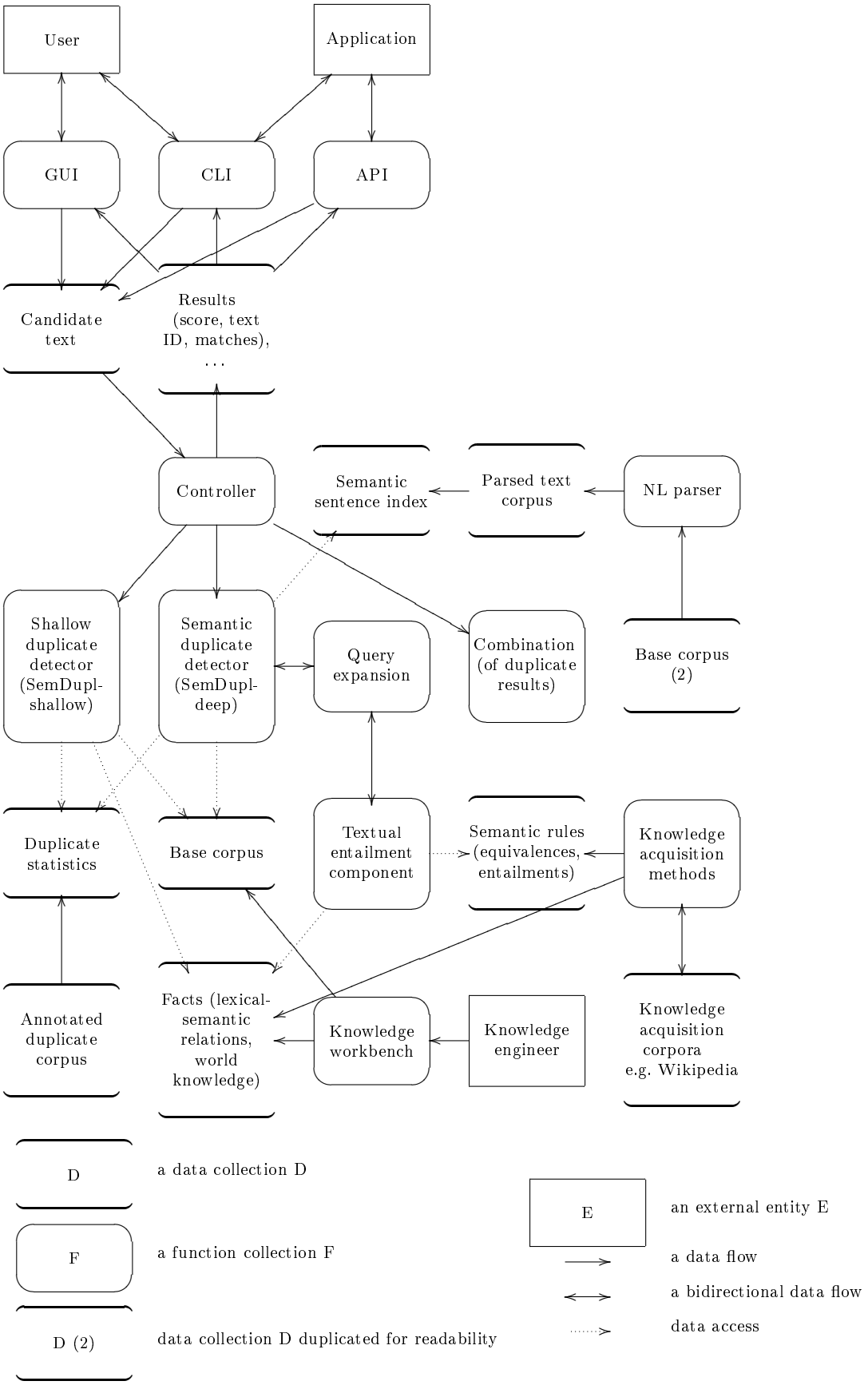

Figure 1: Architecture of the duplicate detector SemDupl .

Internet (semdupl-google) 100 texts collected from Google (the 10 top texts of the 10 fastest growing search terms in 2008), containing 224 duplicate pairs.

Minimal test units (semdupl-units) A collection of (mostly) minimal text pairs. These were written to test for single paraphrase phenomena; the phenomena will be discussed in Section 6. This subcorpus contains 134 duplicate pairs and 5642 non-duplicate pairs.

Plagiarism (htw) Weber-Wulff's collection of plagiarisms, annotated as 77 texts with 141 duplicate pairs and 5788 non-duplicate pairs.

5 The Shallow Duplicate Detectors SemDupl-shallow and SemDupl-quick

SemDupl-shallow is an approach to detect duplicates in a corpus of text documents using shallow techniques only (Eichhorn, 2009). It employs the decision tree learning algorithm ID3 (Quinlan, 1986) to learn to distinguish between duplicate pairs and non-duplicate pairs of texts.

5.1 Features in SemDupl-shallow

Features in SemDupl-shallow use only the surface structure of the analyzed texts. These features are the following:

Checksum To test quickly whether two texts are identical a checksum of each text is calculated with the MD5 algorithm (Rivest, 1992).

Word sets The words of the compared texts as set of words. The sets calculated are: all words of the text, the words of the text without stop words, the words of the text in stemmed form using the Porter stemming algorithm (Porter, 1997), and the words of the text united with the sets of their synonyms.

Typos A good strategy for humans searching for plagiarism is to compare the spelling mistakes made by different authors. If a text is plagiarized, its typos are often copied, too, as shown by Weber-Wulff (2002). Typos are calculated using the spell checker GNU aspell.

Length of words and sentences Weber-Wulff (2002) shows that texts which are plagiarized often share the same style of writing. Since the average length of words and sentences (per paragraph) is one characteristic of the writer's style, these two feature values are calculated and compared.

N-grams Word n-grams are sequences of $n \in \mathbb{N}$ words from the texts. In SemDupl-shallow, different types of n-grams are used to compare a pair of texts:

Standard n-grams n-grams with a length of 3 and 7.

Alliterations These are special n-grams with a length of 3 or 7 where all words start with the same letter.

Phonetic alliterations These are alliterations with the words sharing the same initial phoneme; initial phonemes are determined as described by Brügge and Mohs (2003).

K-skip-n-grams Special n-grams where up to $k \in \mathbb{N}$ words are skipped between the elements of the n-grams, with a skip-length of $k = 2$ and a length of $n = 3$.

5.2 Combination of Features

In order to combine the different features, which may be calculated on a given text pair, the system uses a trained decision tree (Quinlan, 1986). Starting at the root of the tree, the value of the feature given in the node is calculated, the decision tree path appropriate to the calculated value is chosen. This is repeated until a leaf is reached and the pair falls in a class of duplicate or non-duplicate. This is a bit unlike the usual use of decision trees, where all values are known prior to the use of the tree, but in this way only some of the features have to be calculated, which reduces processing time. Since there are 39 features which may be calculated and the learned tree has a minimal depth of 3, a maximal depth of 14 and an average depth of 9.6. roughly only a quarter of all features have to be calculated in order to determine whether the pair is a duplicate pair or a non-duplicate pair.

5.3 Training Phase

Training is done in two steps. In the first step, each feature for each annotated pair of corpus texts is calculated and stored in order to rerun the actual training without time-consuming recalculation of feature values.

In the second step, a decision tree is calculated using the ID3 algorithm (Quinlan, 1986). Since the values are from the interval $[0, 1]$, an extended version of the algorithm is used which is capable of dealing with intervals instead of discrete values only (Petersohn, 2005). Both the original and the extended algorithm are part of the learning environment RapidMiner (Mierswa et al., 2006), which is used as the machine learning component in this process.

5.4 Application Phase

During the application phase, the learned decision tree is traversed. The 111 leafs of the tree consist of:

- 30 leafs containing only duplicate pairs
- 49 leafs containing only non-duplicate pairs
- 32 containing pairs of both classes.

If the tree is used for detecting duplicates, the leafs can be interpreted in different ways:

- The **standard interpretation** (which shows the best overall detection rate) is to take the class of the majority of the examples to determine the class of the leaf.

- If false detection of duplicates has to be reduced, another interpretation called **presumption of innocence** can be used. In this case, a leaf falls into the duplicate class only if there is no evidence that it may be a non-duplicate and therefore only pure duplicate leafs belong to this class, while the others are seen as non-duplicate leafs.

- If, in contrast, false negatives are to be avoided, the interpretation of **presumption of guilt** can be used: only leafs without any duplicates are seen as non-duplicates, while all other leafs are seen as members of the duplicate class.

- Sometimes, SemDupl-shallow may be used as preprocessor for another automated method of detection. Then it is crucial to eliminate all pairs of texts from the analyzed text set that can be reliably classified as duplicate or non-duplicate in order to save machine time which would be used to classify these examples again. Also, the pairs detected to be non-duplicates should contain as few duplicates as possible because if eliminated during preprocessing there is no chance of detecting them with the latter method. This preprocessing or filtering can be done by using the **three class interpretation**, allowing SemDupl-shallow to classify pairs as unknown. A pair is classified as unknown if its leaf contains examples of both classes.

5.5 Aggregation of Feature Values

In order to develop an aggregated numerical value for duplicity, different measures have been tested. Among these methods, the arithmetic mean of the feature values has shown to be the most useful. Calculated on the values of the annotated corpus, the aggregated value of duplicates falls into the interval of $[0,021; 1]$, the aggregated value of non-duplicate falls into $[0,016; 0,701]$. If the aggregated value is to be determined, all features have to be calculated, which will need a lot more time than just determining the pair's class.

5.6 SemDupl-quick (SQ)

Tests indicated that the shallow approach of SemDupl-shallow achieves good results regarding precision and accuracy, but due to its time complexity it is rather unsuited for large corpora. So another shallow approach was devised, using only features that can be calculated efficiently.

In the preprocessing phase, SemDupl-quick (SQ) searches the given texts for misspelled words and words with a frequency class above a given threshold and compiles all n-grams with lengths from 3 to 7. These values are used as indices whereas the text's id (e.g. filename) is used as value. This generates a database with a list of text ids for each value (with a table for each feature).

In the detection phase all rows r containing a given text are searched inside the tables. For each *other* affected text found inside the rows the ratio between the total number of rows r and the number of rows in r containing the affected text id is calculated for each table (and therefore feature). These scores are combined linearly and normalized, resulting in an combined score for each text pair. A text is regarded as a duplicate if the score is greater than a given threshold.

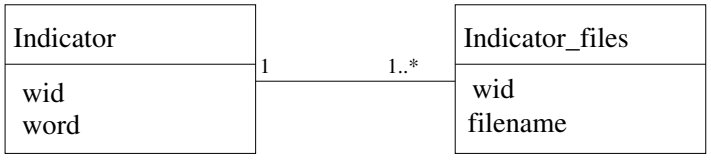

Figure 2: Data model of SemDupl-quick.

Figure 2 shows the data model of SemDupl-quick. In the initialization phase, for each feature the tables <feature>and <feature_files> are filled. An example entry of table *trigrams* is:
Table trigrams: **wid**: 3, **word**: *aber dann dachte*
An example entry of table trigramme_files is:
Table trigrams_files: **wid**: 3, **filename**: brief1_01.txt
For determining the overlap score of a file with all other files only the table <feature_files> is used. The score is determined by calculate the number of common entries, i.e., the relative amount of wid's which occur together with both filenames.

5.7 Comparison of Shallow Approaches

SemDupl-shallow is ready to instantly check an arbitrary pair of texts without *any* preprocessing steps as an "out-of-the-box" duplicate detector. Its capability to learn the "definition" of duplicates on an annotated corpus leads to a detection which has a lower chance of failing because of bad user-set thresholds. Its downside is it has to inspect every possible text pair in order to detect all duplicates in a given corpus, resulting in quadratic time complexity, so it should be used on small or filtered corpora.

SemDupl-quick, on the other hand, uses preprocessing resulting in a lower time complexity while detecting, but only some of the possible features can be used as index-values and the thresholds, which are defined by the user, may, if not set well, become a source of errors.

6 Linguistic Phenomena Relevant for Semantic Duplicates

6.1 Types of Paraphrases for Semantic Duplicates

The following problems must be solved by advanced duplicate detectors. Most of them are out of reach for standard surface-oriented methods; only items 1, 2, 4, 5, 13, 14, and 15 can probably be detected by shallow approaches, at least partially.

For each problem, an illustrative example is given and a solution in our semantic approach is described. Most of these examples are already covered by our deep detector SemDupl-deep (if not indicated otherwise).

1. different word forms (e.g. '*alle Barockkirchen*'/'*all baroque churches*' vs. '*jede Barockkirche*'/'*every baroque church*'): solved by the morphology component, more specifically by the operation called lemmatization. Differences in syntactic case are irrelevant in paraphrases, but differences in number must be investigated more closely. In the above example, the difference is irrelevant

because together with the quantifiers both noun forms mean almost the same. Such semantic equivalences of syntactically different noun phrases can be determined (in part) by the parser.

2. different orthography (e.g. '*Fluss*'/'*river*' vs. '*Fluß*'): solved by the morphology component, by way of spelling normalization (new and old orthography in German; also spelling variants).

3. semantically light or empty expressions: '*Der Außenminister will den Grund der Diskussion* ja dann doch *wissen.*'/'*The secretary of foreign affairs now wants to know the reason for the discussion.*' vs. '*Der Außenminister will den Grund der Diskussion wissen.*'/'*The secretary of foreign affairs wants to know the reason for the discussion.*'

4. abbreviations/acronyms and expanded forms ('*die EU*'/'*the EU*' vs. '*die europäische Union*'/'*the European Union*'): solved by lexical resources available to the parser.

5. different hyphenation of compounds (e.g. '*Barock-Kirche*'/'*baroque church*' vs. '*Barockkirche*'): solved by the compound analysis component.

6. different word order, e.g. '*Der Schüler freute sich über das Buch.*'/'*The student was happy about the book.*' vs. '*Über das Buch freute sich der Schüler.*'/'*literally: About the book the student was happy.*': solved by the parser. In German, word order is much freer than in English.

7. discontinuous word forms (e.g. German verbs with separable prefix: '*... als der Minister den Namen* aufschrieb.'/'*... as the secretary noted the name down.*' vs. '*Der Minister* schrieb *den Namen* auf.'/'*The secretary noted the name down.*'): solved by the parser, which combines a separated verb prefix with the corresponding base verb.

8. different voices (active or passive in German): e.g. '*Der Lehrling reparierte das Auto.*'/'*The apprentice repaired the car.*' vs. '*Das Auto wurde vom Lehrling repariert.*'/'*The car was repaired by the apprentice.*': solved by the parser because it generates identical semantic networks.

9. arguments in different clauses, e.g. '*Die Frau lief 10 Kilometer.*'/'*The woman ran 10 kilometers.*' vs. '*Es gelang der Frau, 10 Kilometer zu laufen.*'/'*The woman achieved to run 10 kilometers.*'. In this example, the lexical syntax and semantics of the control verb '*gelingen*'/'*to achieve*' allows to identify similar content.

10. nominalization of situations, e.g. '*der Kauf der kleinen Firma durch den Weltkonzern*'/'*the acquisition of the small company by the world concern*' vs. '*Der Weltkonzern kauft die kleine Firma.*'/'*The global company buys the small company*'. Solution: The verb and the corresponding nominalization are linked in the lexicon and lead to isomorphic semantic networks that can be easily matched to find similar sentences.

11. information distribution across sentences, e.g. '*Der Räuber verließ die Bank und flüchtete zum Bahnhof.*'/'*The bandit left the bank and escaped to the train station.*' vs. '*Der Räuber verließ die Bank. Er flüchtete zum Bahnhof.*'/'*The bandit left the bank. He escaped to the train station.*' As the parser works sentence-oriented, the representations on the sentence level are different. But if one allows to split a conjunction (of situations) when searching, the information can be found also in the document version that uses several sentences. (Note that the case where the information in the candidate document is split in several sentences matches a document with a combining sentence is already covered without any extensions.)

12. names vs. definite description, e.g. '*Paris*' vs. '*die französische Hauptstadt*'/'*the French capital*'. This effect can be compensated by so-called question decomposition in SemDupl-deep. As the term question decomposition indicates, this kind of decomposition was introduced for question answering, but it can also applied in semantic search for similar sentences. Currently, only one direction (the replacement of a name) can be detected, mainly for efficiency reasons.

13. synonyms: partially solved by HaGenLex (Hartrumpf et al., 2003) plus GermaNet (Hamp and Feldweg, 1997) (relation SYNO); for compounds, many synonyms can be inferred from synonyms of parts, e.g. '*Großstadt-Atmosphäre*'/'*the big city atmosphere*' (literally) vs. '*Metropolen-Atmosphäre*'/'*the metropolis atmosphere*'.

14. hyponyms (or viewed reversely: hypernyms; in the case of verbs, these are often called troponyms), e.g. '*Geschäftsbrief*'/'*business letter*' vs. '*Brief*'/'*letter*': solved by lexico-semantic relations (SUB and similar relations) stored in HaGenLex and derived from other sources like GermaNet.

15. antonyms (plus negation), e.g. '*nicht richtig*'/'*not right*' vs. '*falsch*'/'*false*'): solved by lexico-semantic relations (ANTO and its subrelations) stored in HaGenLex and derived from other sources like GermaNet. In some cases, the negation of an antonym is not completely equivalent to the original concept. Corpus analyses and statistics can help to detect such cases.

16. compounds vs. analytical expressions like complex NPs and clauses: compounds with regular semantics can be paraphrased by a complex constituent, e.g. '*Finanzierungslücke*'/'*finance gap*' vs. '*Lücke bei der Finanzierung*'/'*gap in financing*' and '*die Großstadt-Atmosphäre*'/'*the metropolis atmosphere*' vs. '*die großstädtische Atmosphäre*'/'*the metropolitan atmosphere*'. Solution: transformation of compound semantics by the rule component of SemDupl.

17. idioms, e.g. '*einen Entwurf in die Tonne werfen.*'/'*to throw a draft into the bin*' (literally) vs. '*einen Entwurf verwerfen.*'/'*to discard a draft*': These are probably not solvable in all cases in the near future. But cases like '*ins Gras beißen*'/'*kick the bucket*' vs. '*sterben*'/'*to die*' and the example given above can be entered in an idiom lexicon. Currently, an idiom lexicon of around 250 idioms based on verbs is employed.

18. support verb constructions (SVCs), e.g. '*einen Widerspruch äußern*' vs. '*widersprechen*'. Solution: an SVC lexicon that allows to normalize an SVC to a form

without the SVC. In SemDupl, this achieved by MultiNet rules applied during query expansion.

19. coreferences (different expressions referring to the same entity): solved by the coreference module. For example intrasentential coreferences: '*Syrakus, das im Jahr 734 v. Chr. gegründet wurde ...*'/'*Syracuse, which was founded in the year 734 BC ...*' vs. '*Syrakus wurde im Jahr 734 v. Chr. gegründet.*'/'*Syracuse was founded in the year 734 BC.*' (the semantic representation of the first sentence contains a correct resolution of the relative pronoun '*das*'); intersentential coreferences: '*Adenauer*', '*Konrad Adenauer*', '*er*' is a possible sequence of noun phrases (forming a so-called coreference chain), as is '*Konrad Adenauer*', '*er*', '*Adenauer*', which leads to completely different surface sentences, but almost identical semantics.

20. presuppositions: existential presuppositions, e.g. '*Der König von Frankreich reiste 1730 ...*'/'*The king of France traveled in 1730.*' implies (presupposes) '*Es gab einen König von Frankreich 1730.*'/'*There was a king of France in 1730.*'

21. entailments (especially entailments of verbs), e.g. '*Die Firma verkauft ein Patent an den Konkurrenten.*'/'*The company sells a patent to the business rival.*' vs. '*Der Konkurrent kauft ein Patent von der Firma.*'/'*The business rival bought a patent from the company.*'): covered in part by entailments from HaGenLex and entailments derived from knowledge bases like GermaNet and manual translations of XWordNet.

22. metaphors like '*am Anfang des Lebensweges*'/'*at the beginning of the path of life*' vs. '*nach der Geburt*'/'*after birth*' (metaphor: life-is-journey or life-is-way). Metaphors span a whole range of phenomena. For example, lexicalized metaphors can be easily handled by the lexicon, but more general cases would require a special metaphor interpretation module to be used by the parser.

This enumeration could be extended further and further. In the end, complete text understanding plus complete paraphrasing and inferencing is needed; this is the ultimate goal of natural language understanding (NLU); this goal can only be approximated step by step, phenomenon by phenomenon, module by module, as indicated in this section.

6.2 Restrictive Contexts and Other Precision Problems for Semantic Duplicates

For each problem, an illustrative example and a solution is given.

1. incorrectly collapsed discourse entities, e.g. '*IBM kaufte 2000 andere Firmen.*'/'*literally: IBM bought 2000 other companies*': '*2000*' is a year not a quantifier for '*andere Firmen*'. Partially solved by sentence segmentation and parsing of sentences.

2. incorrect phrases: solved by parsing sentences

3. incorrectly selected reading (wrong reading of ambiguous word or constituent): e.g. '*Die Birne schmeckt gut.*'/'*The pear tastes well.*' would be irrelevant if the query is about '*Birne*' in the sense of '*Glühbirne*'/'*light bulb*'. Solved by the lexicon and the parser.

4. negation; constituent negation (compatibility test for the FACT layer feature in MultiNet suffices); sentence negation, similarly.

5. other modalities, e.g. '*Frankreich erzielt vielleicht höhere Einnahmen.*'/'*Maybe France achieves higher incomes.*' would (probably) be an incorrect duplicate match if the other text reads '*Frankreich erzielt höhere Einnahmen.*'/'*France achieves higher incomes*'. Incompatible modalities are tested in the semantic network representations. Similarly, hypothetical situations must be excluded from answering factual questions, e.g. '*Wenn London in Frankreich liegt, ist Schnee weiß.*'/'*If London is located inside France, then snow is white.*' A sentence that could incorrectly match as a (partial) duplicate of the main clause is: '*London liegt in Frankreich.*'/'*London is located inside France.*' Other examples of modality come from epistemic modals like '*glauben*'/'*to believe*', e.g. '*Er glaubte, dass der Minister vom Präsidenten entlassen wird.*'/'*He believed that the secretary was dismissed by the president.*' vs. '*Der Minister wird vom Präsidenten entlassen.*'/'*The secretary was dismissed by the president.*'

7 Knowledge Acquisition for Deep Duplicate Detectors

The deep duplicate detector SemDupl-deep can only be as good as the underlying knowledge bases. Therefore, the SemDupl project tries

- to consolidate our existing knowledge sources,
- to automatically (or semi-automatically) derive new knowledge bases, and
- to validate these new knowledges bases.

7.1 Hypernyms and Meronyms

A type of textual entailment that is both quite easy to create and to detect is a sentence pair where the two sentences are almost identical, with the exception that certain words (or concepts on a semantic level) of the original sentence are replaced by synonyms, hypernyms or holonyms (the inverse of meronyms). This is a modification which can be done quite easily to obfuscate a plagiarism. For example, '*His father went to Bavaria.*' implies '*His father went to Germany*'. In the second sentence, '*Bavaria*' is replaced by one of its holonyms, '*Germany*'. Thus, a large collection of synonyms, hypernyms and holonyms is quite vital for entailment recognition.

Since the Wikipedia is often a source for creating plagiarisms or duplicates, synonyms, hypernyms and holonyms are extracted from Wikipedia using a pattern-based approach (vor der Brück, 2009; vor der Brück, 2010a,c,b, 2011; vor der Brück and Stenzhorn, 2010; Tim vor der Brück, 2010; vor der Brück and Helbig, 2010). For

$$(a1 \text{ SUB0 } a2) \leftarrow \quad a1([word\ \text{","}]\ [cat\ (\text{art})]^{?}\ a1)^{*}$$
$$[word\ \text{"und (and)"}]\ [cat\ (\text{art})]^{?}$$
$$[lemma\ \text{"ander (other)"}]\ [cat\ (\text{a})]^{?}\ a2$$

Figure 3: Shallow pattern for hyponymy extraction where the premise is given as a regular expression.

this it is differentiated between shallow and deep patterns. Both types of patterns consist of a conclusion part of the form
$(a1 \text{ SYNO } a2)$, $(a1 \text{ SUB0 } a2)$ or $(a1 \text{ MERO } a2)$
which specifies that, if the premise holds, a synonymy / hyponymy / meronymy relationship between the constants which are assigned to the variables *a1* and *a2*, holds. The assignments for both variables are determined by matching the premise part to a linguistic structure which is created by analyzing the associated sentence.

In the following, we only describe the extraction of hyponyms. The extraction of synonyms and meronyms works quite similar. The premise of a shallow pattern is given as a regular expression and is matched to the token information given by the parser. This token information contains the following information:

- word: word form
- category: grammatical category
- lemmas: a list of possible lemmas
- readings: a list of possible readings
- reading: the reading determined by the word sense disambiguation of the parser
- lemma: lemma determined by the word sense disambiguation of the parser

If a match is successful, then the variables of the conclusion are replaced with the constants matched to the same variables in the premise. Note that a variable of the premise can be matched several times, either if it appears several times in the premise or if that variable is in the scope of a Kleene-star operator (both is true in the pattern given in Figure 3). In this case, the relations stated by the Cartesian product of all possible variable instantiations are extracted.

The premise of a deep pattern is given as a semantic network graph, whereas the conclusion is the semantic relation to be deduced. The deep pattern is applied by a graph pattern matcher (or an automatic theorem prover if axioms are to be employed). An example pattern is given in Equation 1 and Figure 4.

$$\begin{aligned}(a1 \text{ SUB0} a2) \leftarrow\ & follows_{*_{\text{ITMS}}}(c,d) \wedge (d \text{ PRED } a2) \wedge \\ & (d \text{ PROP } ander.1.1(other.1.1)) \wedge \\ & (c \text{ SUB} a1)\end{aligned} \tag{1}$$

$follows_{*_{\text{ITMS}}}(c,d)$ denotes the fact that c precedes d in the argument list of the function *ITMS.

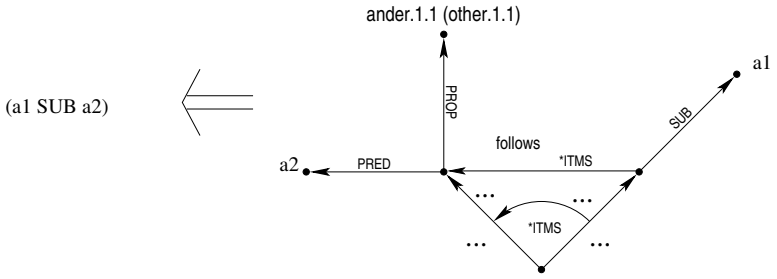

Figure 4: Deep pattern for hyponymy extraction where the premise is given as an SN.

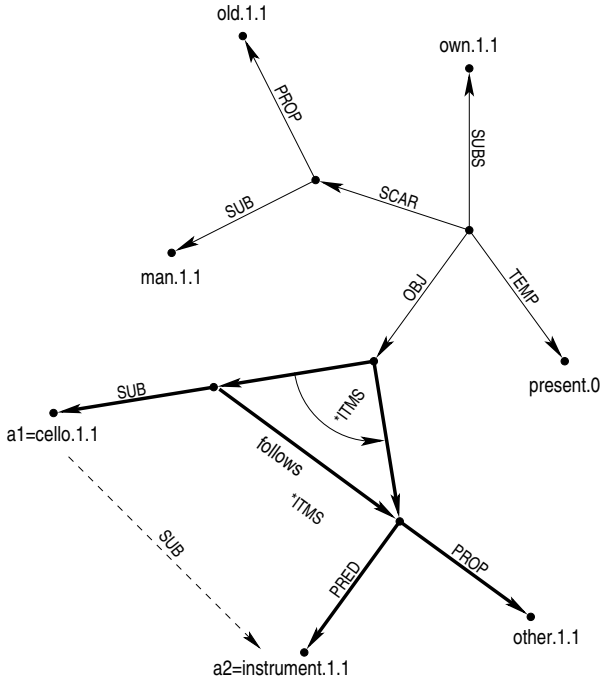

Figure 5: Matching a deep pattern to a semantic network.

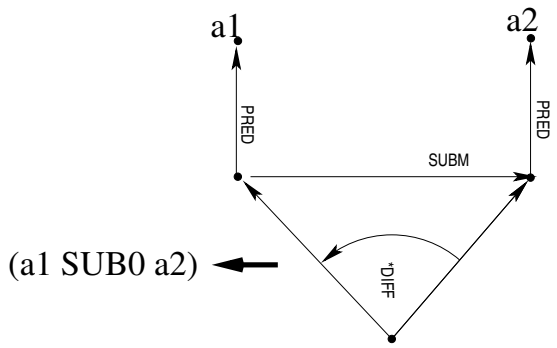

Figure 6: Deep pattern for hyponymy extraction where the premise is given as an SN.

Figure 5 illustrates the application of the deep pattern that is displayed in Figure 4 to a semantic network representing the sentence: '*Der alte Mann besitzt ein Cello und andere Instrumente.*'/'*The old man owns a cello and other instruments*'. The semantic network arcs which are matched with the literals of the pattern are printed in bold. The dashed edge represents the inferred fact: (*cello.1.1* SUB *instrument.1.1*) with the variable instantiations *a1* = *cello.1.1*, *a2* = *instrument.1.1*.

Note that we consider *instance of* relations as a special kind of hyponymy as well and such relations were also extracted by our algorithm. *Instance of* relations involving named entities are represented in MultiNet using attribute-value constructions, e.g. '*Bonn ist eine Stadt.*'/'*Bonn is a city.*' is represented by

$$(a \text{ ATTR } b) \wedge (b \text{ SUB } name.1.1) \wedge (b \text{ VAL } bonn.0) \wedge (a \text{ SUB } stadt.1.1) \qquad (2)$$

7.2 Deep vs. Shallow Patterns

On the one hand, a shallow pattern has the advantage that it is also applicable if the parse fails. It only relies on the fact that the tokenization is successful. On the other hand, deep patterns are still applicable if there are additional constituents or subclauses between hyponyms and hypernyms which usually cannot be covered by shallow patterns. The following sentences (the first two originating from the Wikipedia corpus) are typical examples where the hyponymy relationship could only be extracted using deep patterns (hyponym and hypernym are set in bold face).

'*Die Stadt besitzt eine romanische* ***Kathedrale*** *aus dem 12. Jahrhundert, viele andere romanische* ***Kirchen*** *und einige Museen.*' '*The city contains a Romanesque* ***cathedral*** *from the 12th century, a lot of other Romanesque* ***churches*** *and some museums.*'

Another advantage of deep patterns is illustrated by the following sentence: '*Auf jeden Fall sind nicht alle Vorfälle aus dem* ***Bermudadreieck*** *oder aus anderen* ***Weltgegenden*** *vollständig geklärt.*'/'*In any case, not all incidents from the* ***Bermuda Triangle*** *or from other* ***world areas*** *are fully explained.*'

From this sentence, a hyponymy pair cannot be extracted by the Hearst pattern '*X or other Y*' (Hearst, 1992). The application of this pattern fails due to the word '*aus*'/'*from*' which cannot be matched. To extract this relation by means of shallow patterns an additional pattern would have to be introduced. This could also be the case if syntactic patterns were used instead since the coordination of

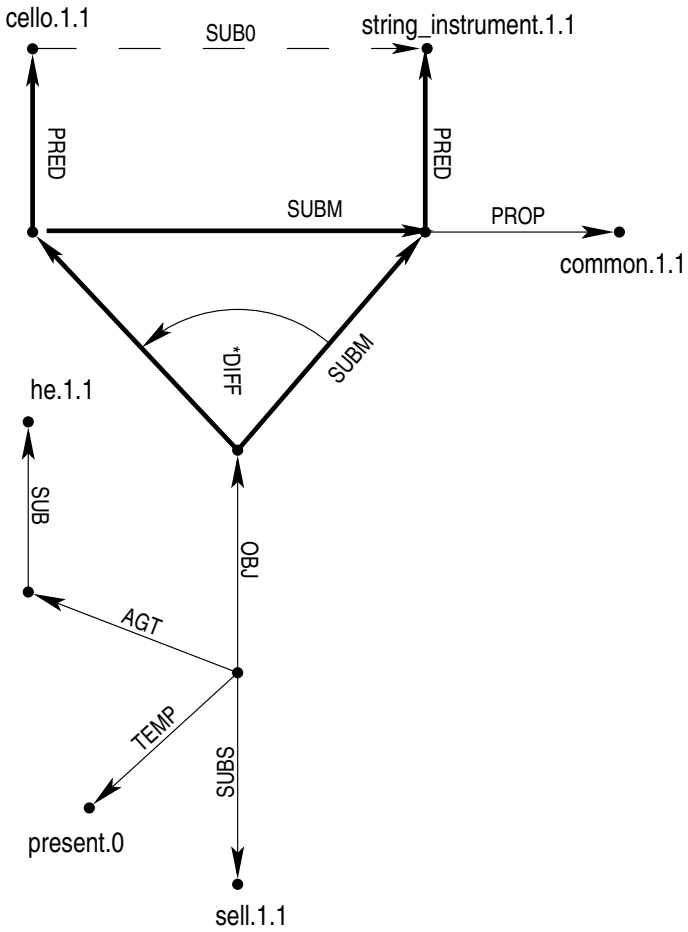

Figure 7: Application of a deep pattern to an SN representing the sentence *He sells all common string instruments except celli.* Arcs matched with the pattern premise are printed in bold. The dashed arc is inferred by application of the pattern.

'*Bermudadreieck*'/'*Bermuda Triangle*' and '*Weltgegenden*'/'*world areas*' is not represented in the syntactic constituency tree but only on a semantic level[8]. Thus, the same deep pattern can be used for the hyponymy extraction in this sentence as for extracting the hyponymy relationship from the phrase: '*das Bermuda-Dreieck oder andere Weltgegenden*'/'*the Bermuda Triangle or other world areas*'.

Furthermore, several syntactic or surface representations are frequently mapped to the same semantic network like in the sentences:

1. '*Er besitzt ein Cello, eine Geige* ***und*** *andere Instrumente.*'/'*He owns a cello, a violin* ***and*** *other instruments.*'

2. '*Er besitzt eine Geige, ein Cello* ***sowie*** *andere Instrumente.*'/'*He owns a violin, a cello* ***as well as*** *other instruments.*'

Thus, the hyponymy relationship that cello and violins are instruments can be extracted by the application of the same deep pattern. However, to extract the same information by the application of shallow or even syntactic patterns, two different patterns have to be defined.

Another example:

- '*Er verkauft alle gebräuchlichen Streichinstrumente außer Celli.*'/'*He sells all common string instruments except celli.*'
- '*Er verkauft alle gebräuchlichen Streichinstrumente bis auf Celli.*'/'*He sells all common string instruments aside from celli.*'
- '*Er verkauft alle gebräuchlichen Streichinstrumente ausgenommen Celli.*'/'*He sells all common string instruments excluding celli.*'

All three sentences have different dependency trees (tested by applying the Stanford Dependency Parser (de Marneffe and Manning, 2008) on the English translations). However, the SN representations of all three sentences are identical (depicted in Figure 7), i.e., the pattern given in Figure 6 can be applied to extract the hyponymy relation (*cello.1.1* SUB0 *string_instrument.1.1*) from all three sentences, while three different syntactic patterns would have to be designed if the same relation was to be extracted from the dependency parse. The same fact holds if surface representations are used. Thus, the use of a deep semantic representation reduces the amount of required patterns in comparison to a surface or dependency based representation. A further advantage of the deep semantic approach consists in the fact that person names are already identified by the parser which simplifies the extraction of hyponyms (instance-of relations) relating to persons. Finally, the deep approach allows the usage of logical axioms, which can make the patterns more generally applicable.

7.3 Validation of Relation Hypotheses

Naturally, not all hypotheses extracted by the patterns are correct. Therefore, a validation component is required. A two-step validation is followed here. In the first step, ontological sorts and features (Helbig, 2006) of the two concepts involved in a relation hypothesis are compared. Only if the sort/feature combination is admissible, the hypothesis is stored in the database. In the second step, several

[8]Note that some dependency parsers do a semantic-oriented normalization as well.

statistical features are calculated and employed to derive a confidence score (vor der Brück, 2010a; vor der Brück and Helbig, 2010).

Let us look at the first validation step more closely. First we give a short overview on ontological sorts and semantic features.

The ontological sorts (MultiNet defines 45 sorts) form a taxonomy. In contrast to other taxonomies, ontological sorts are not necessarily lexicalized, i.e., they do not necessarily denote lexical entries. The following list shows a small selection of ontological sorts below the sort *object* (*o*):

- Objects
  - Concrete objects
    - Discrete objects: e.g. '*chair*'
    - Substances: e.g. '*milk*', '*honey*'
  - Abstract objects: e.g. '*race*', '*robbery*'

Semantic features are certain properties which can be set, unset or underspecified. The following semantic features are defined:

- ANIMAL
- ANIMATE
- AXIAL
- ARTIF (artificial)
- GEOGR (geographic)
- HUMAN
- INFO
- INSTIT (institution)
- INSTRU (instrument)
- LEGPER (legal person)
- MENTAL
- METHOD
- MOVABLE
- POTAG (potential agent)
- SPATIAL
- THCONC (theoretical concept)

Sample characteristics for the concept *bear.1.1* (The suffix .1.1 denotes the reading numbered .1.1 of the given word.): discrete object; ANIMAL +, ANIMATE +, ARTIF -, HUMAN -, SPATIAL +, THCONC -, ...

Each hypothesis stored in the database has passed a consistency check using ontological sorts and features. For hyponymy hypotheses, the ontological sorts/features of the hypernym must subsume the sorts/features of the hyponym. For synonymy hypotheses, ontological sorts and features must be identical for the two compared concepts. For meronymy hypotheses, the allowed combinations are learned by a tree-augmented naïve Bayes algorithm. Examples:

- *bear.1.1* (animal:+) can be no hyponym/synonym of *house* (animal:-)
- *milk.1.1* (substance) can be no hyponym/synonym of *house* (discrete object)
- *house.1.1* (movable:-) can be no hyponym/synonym of *chair.1.1* (movable:+)
- *living_ being.1.1* (animal:underspecified) can be a hypernym but no hyponym/synonym of *bear.1.1* (animal:+)
- *child.1.1* (human:+, etype:0) can be no meronym of *man.1.1* (human:+, etype:0)

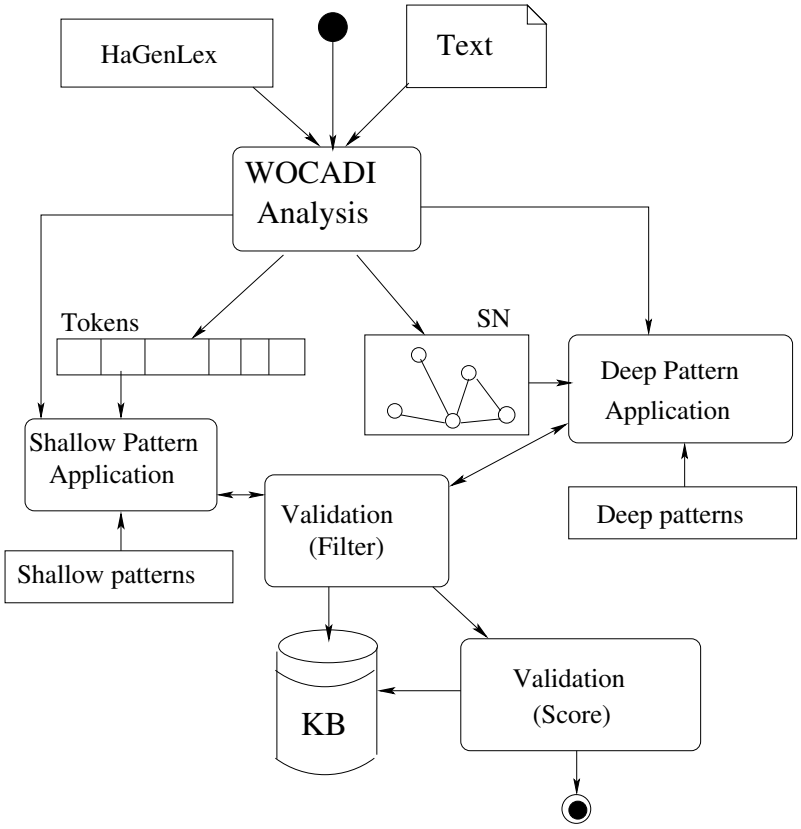

Figure 8: System Architecture of SemQuire.

7.4 System Architecture: Relation Extraction

To find meronymy relations from a text, this text is processed by our knowledge acquisition tool called SemQuire.[9]

1. The sentences of the German Wikipedia are analyzed by the deep linguistic parser WOCADI. As a result of the parsing, a token list, a syntactic dependency tree, and a semantic network are created.

2. Shallow patterns, consisting of a regular expression in the premise, are applied to the token lists, and deep patterns are applied to the SNs to generate proposals for meronymy relations.

3. A validation tool using ontological sorts and semantic features checks whether the proposals are technically admissible to reduce the amount of data stored in the knowledge base (KB).

4. If the validation is successful, the meronymy candidate pair is added to the KB. Steps 2–4 are repeated until all sentences are processed.

5. Each meronymy candidate pair in the KB is assigned a confidence score estimating the likelihood of its correctness.

6. The correct hyponymy/meronymy subrelation is determined. (No further subrelation exists for synonymy.)

[9]SemQuire is derived from *semantically acquire knowledge*.

8 Annotation GUI

A GUI-based tool called SemChecker is provided for annotating the extracted hyponym / meronym / synonym hypotheses for correctness Danninger (2009). SemChecker is primarily an annotation tool for relation hypotheses but can be used also to identify inconsistent hypotheses. SemChecker displays the hypotheses contained in the database including its associated database attributes. In the upper part of the window in Figure 9 the contents of the database table RELATION is given. If one of these entries is selected, the associated entries of the SOURCE table, which specify the source of that entry, is displayed. Additionally, by means of the attributes FILENAME and SENTENCE_INDEX, the sentence from which the hypothesis was extracted from is identified and shown. This sentence is quite vital for the annotator to decide whether a hypothesis is correct or not. This is especially the case for technical terms which are not generally known. Without the text, the annotator is frequently not able to decide about the hypothesis' correctness. In addition, this sentence is often quite useful to identify shortcomings of the relation extraction process and helps therefore to further improve this process. Furthermore, several bits of information for the concepts involved in the relation hypothesis from the lexicon are shown, including ontological sorts and semantic features as well as an example sentence which clarifies the intended reading. In the shown case, the mistake that the wrong reading was chosen for one of the words of the hypothesis can be identified easily. Usually such errors are based on the fact that a reading was incorrectly chosen by the word sense disambiguation of the parser.

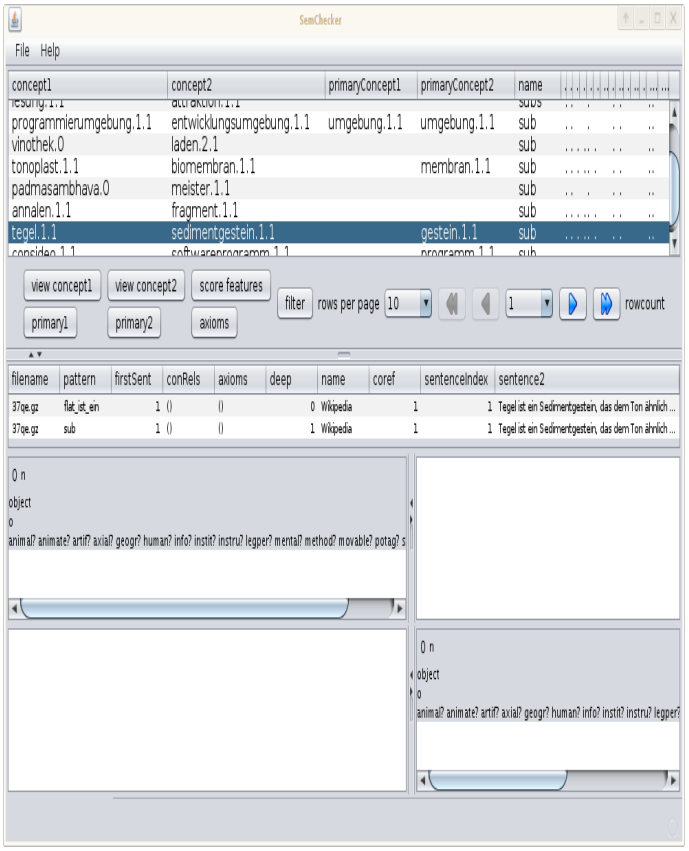

Figure 9: Screenshot of the annotation tool.

In the middle of the graphical user interface, there are several buttons whose functions are defined as follows:

- `view concept1`: Display the first concept (hyponym / meronym / first synonym) in LIA by employing the LIA remote control. In order for this to work, LIA has to be started previously.
- `view concept2`: Display the second concept (hypernym / holonym / second synonym) in LIA.
- `view primary1`: Display the associated reading belonging to the base word (primary word) of the first concept , if existing, in LIA.
- `view primary2`: Display the associated reading belonging to the base word (primary word) of the second concept, if existing, in LIA.
- `score features`: Displays the features which were used to calculate the confidence score for the selected hypothesis. This value must not be confused with the semantic features of MultiNet.
- `axioms`: Displays the axioms that were applied to derive a contradiction for this hypothesis. This is only possible if the automated theorem prover was able to derive a contradiction for the selected hypothesis.
- `filter`: Activates a filter, which causes only the hypotheses to be displayed whose database attributes fulfill the filtering conditions. SemChecker supports filtering according to equality or interval checks for numerical values.

Additionally, SemChecker provides the possibility to sort the shown data according to the database attributes. For this, the user just selects the table border above the desired column.

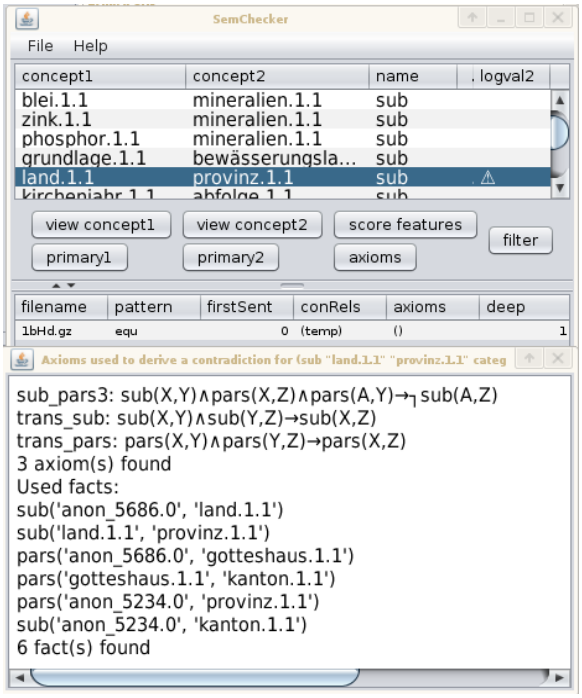

Figure 10: Graphical user interface of SemChecker for displaying the result of the logical validation, which marks the entry SUB(*land.1.1*, *provinz.1.1*) as potentially defective.

Beside its use as an annotation tool, SemChecker can also be used to show inconsistent relation hypotheses. Entries which caused a contradiction are marked

by an *attention* symbol. Additionally, if the button *axioms* is selected, SemChecker displays all axioms which were required for deriving the contradiction proof for the selected hypothesis. This is illustrated in Figure 10.

9 Extraction of Entailments

Next to semantic relations, a collection of entailments can be useful for plagiarism or duplicate detection. An entailment is a relationship between two expressions which holds, if the truth of the first expressions (the base expression/text) implies the truth of the other (the hypothesis). Three different approaches for entailment extraction were devised which are described in the following three subsections.

9.1 Extracting Entailments Employing a Search Engine

We basically follow the approach of Ravichandran and Hovy (2002) which identifies entailments by collecting texts with identical noun phrases, using the assumption that such texts often contain similar contents. Consider for example the noun phrases: *John McEnroe*, *Björn Borg*, *Wimbledon*, *1980*. Sentences containing such expressions could be:

- *John McEnroe lost against Björn Borg in the final of Wimbledon 1980.* or
- *Björn Borg beat John McEnroe in the final of Wimbledon 1980.*

Thus, if a lot of such texts are examined, the entailments

- X beat $Y \rightarrow Y$ lost against$X$ and
- Y lost against$X \rightarrow X$ beat Y

can eventually be learned. The entailment extraction is done in the following steps:

- SNs are created for all texts which contain the given noun phrases.
- The texts are extracted by web search engine queries.
- Nominal phrases which are employed for the search are replaced by variables.
- Frequently occurring substructures S in these SN are learned by following the Minimum Description Length Principle (Cook and Holder, 1994).
- Entailments are created by building the Cartesian product over $S : S \times S$, i.e., the first component of a pair $s \in S \times S$ represents the base expression, the second the hypothesis.

Entailments extracted by this approach are given in Appendix A.

9.2 Extracting Entailments from SNs Basically Following Ravichandran and Hovy

The approach described in Section 9.1 profits from a large corpus (the part of the web indexed by the search engine) and the precision is quite good due to the fact that the method is semi-automatic and a human user has to provide the search tuples. However, this can also be a disadvantage since human interaction is required and a high amount of labor is required to extract a high number of entailments. Furthermore, the approach is not fully semantic-oriented since surface strings are used for the search engine queries. Therefore, we devised an additional approach which is fully automatic and is also based on the approach of Ravichandran and Hovy (2002). In this approach, we extract entailments from a newsfeed corpus in form

Table 1: Assumed equivalent formulas with given confidence score.

Formula 1	Formula 2	Score
compl1 (holen) nach.dircl	compl1 (werden) an.loc (institut) in.loc	1.000
compl1 (holen) nach.dircl	compl1 (niederlassen) in.compl2	1.000
compl1 (holen) nach.dircl	compl1 (gehen) nach.dircl	0.219
adj (studieren) compl1	in.loc (promoviern) compl2	0.106
compl1 (kommen) auf.circ (Weg) zu.compl2	compl1 (zur"uckkehren) zu.purp	1.000

of semantic networks. The semantic networks were derived by a deep syntactico-semantic analysis (the WOCADI parser). The search tuples were determined by corpus statistics following the method introduced by Szpektor et al. (2004) called *TE/ASE*.

9.3 Extracting Entailments Basically Following Lin and Pantel

We further tested a different entailment extraction approach (Zauner, 2009) which is based on the method of Lin and Pantel (2001). For this method it is not necessary to construct any search tuples like for the two other approaches, which is not trivial and can result in either many work or considerably computations if done automatically. However, the other two approaches created several intermediate results which could be used to manually influence the final result which is not possible here.

This approach extracts all paths from a dependency tree which connect two nouns with each other. These nouns are referred to as slot fillers. Two text passages are considered as similar (and therefore as paraphrases) if the associated paths connect essentially the same nouns with the same frequency with each other. For this purpose, every path is assigned two vectors which specify the number of occurrences of the slot filler expressions in the text corpus. For the determination of the similarity of the vector pairs, a distance function is defined, where the occurrence of usually rare nouns has more influence for the similarity calculation than of more frequent ones. Note that this algorithm is quite similar to the synonymy/hyponymy recognition by textual context analysis (Cimiano et al., 2005).

Instead of dependency trees, this approach was applied to SNs following the MultiNet formalism. Furthermore, a hybrid was realized which is based on dependency trees but replaces the edge labels of the trees by the semantic relations. Some of the extracted entailments are shown in Table 1.

10 The Deep Duplicate Detector SemDupl-deep

To adequately handle linguistic phenomena, i.e., identify paraphrase phenomena discussed in Section 6.1 and to not get disturbed by non-paraphrase phenomena discussed in Section 6.2, a deep semantic approach to duplicate detection has been developed. It integrates existing tools for producing semantic representations for

text: the WOCADI parser and the CORUDIS coreference resolver. In an indexing phase, all texts in the base corpus are transformed into semantic representations by WOCADI and CORUDIS (Hartrumpf, 2003).

In the detection step, the duplicate candidate is analyzed in the same way as the texts of the base corpus. For each sentence in the candidate, a semantic search query is sent to a retrieval system that contains all the semantic representations for the base corpus. Matches are collected and after all sentences of the candidate have been investigated, scores are calculated from the results for the text sentences.

The average overlap score over all candidate sentences is a good score. The individual overlap score is calculated by the retrieval system, based on distances of related concepts and the distance between the left-hand side and right-hand side of inference rules.

The power and potential of this detector was illustrated by some examples in Section 6. Some more real-world examples from the SemDupl corpus are presented next. The following four examples can be detected as semantically equivalent by a unit conversion rule between meters and kilometers and abbreviation definitions for '*m*' and '*km*': '*Die Frau lief 10 Kilometer/10000 Meter/10 km/10000 m.*'/'*The woman ran 10 kilometers/10000 meters/10 km/10000 m.*'

Another example that shows the inferential power of SemDupl-deep in the area of verb meanings are the three following sentences: '*Der alte Parlamentarier stellt einen Antrag auf eine Beratung.*'/'*The old parliamentarian files an application for a debate.*', a variant with the SVC replaced by a full verb '*Der alte Parlamentarier beantragt eine Beratung.*'/'*The old parliamentarian applies for a debate.*' and finally a variant with an infinitive clause replacing the NP taking the object role of the verb: '*Der alte Parlamentarier beantragt zu beraten.*'/'*The old parliamentarian applies to debate.*' (literally translated). All three semantic duplicates can be linked; This linking successfully used knowledge about support verb constructions ('*einen Antrag stellen*'/'*to file an application*' and '*beantragen*' / '*to apply*') and relationships between verbs and their nominalizations ('*beraten*'/'*to debate*' and '*Beratung*' / '*debate*').

From the independent translations subcorpus, the following is a illustrative example that shows what the deep method can detect: '*Man weiß ferner, dass sie noch im Besitze des Dokumentes ist.*'/'*Furthermore, one knows that she still holds possession of the document.*' (corpus text brief1_01) vs. '*Man weiß ebenfalls, daß sich das Schriftstück noch in ihrem Besitz befindet.*'/'*Likewise one knows that the document is still in her possession.*' (corpus text brief2_01).

To evaluate the inferential aspects of SemDupl-deep, the English test set from RTE-2 (Recognizing Textual Entailment Challenge) was translated to German by a native German speaker. Then, the translations of the IE and IR parts were controlled by an English native speaker in order to check that no information from the English versions was missing or mistranslated and finally revised by a second German native speaker.

11 Combination

The architecture of the Combiner is illustrated in Figure 11. First, SemDupl-shallow, SemDupl-quick and SemDupl-deep produce XML output containing the feature values for a given training set of annotated text pairs. The annotation is 1 for duplicates

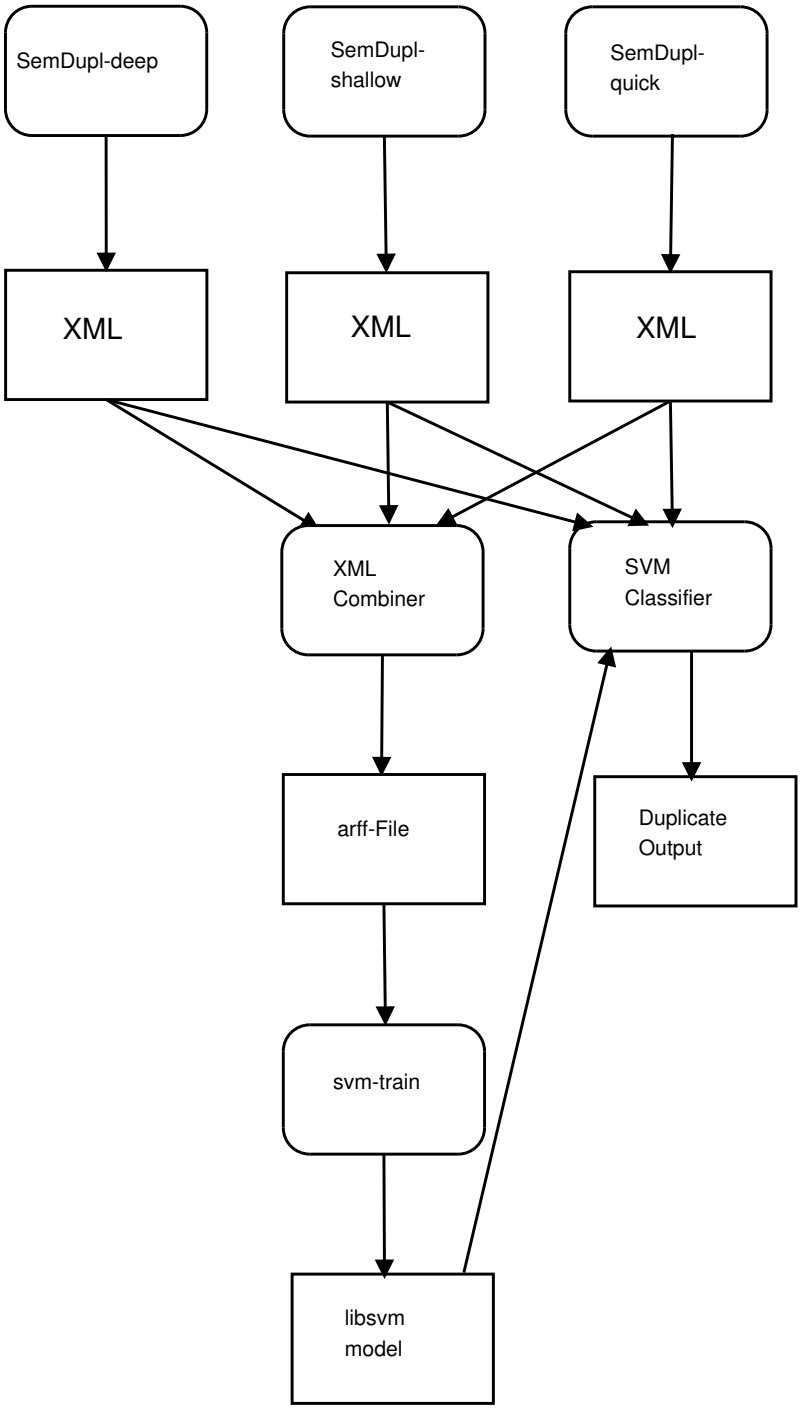

Figure 11: System architecture of the Combiner.

Table 2: Confusion matrix for WCopyFind. D=Duplicate, ND=No Duplicate, PD=Predicated Duplicate, PND=Predicted Non-Duplicate.

	PD	PND
D	39	215
ND	8	21645

and 0 for non-duplicates. The XML output is combined by the XMLCombiner which produces a single LIBSVM file as the result. This file is used by the support vector machine LIBSVM (Chang and Lin, 2001) to create a model file. Afterwards, a classification for a previously unseen data set can be done. Again, SemDupl-shallow, SemDupl-quick and SemDupl-deep produce XML output for this dataset. This output is combined by the `svm_classifier.out` which additionally employs the model file to classify each instance of this dataset. The classification output consists of a list of 4-tuples
$(t_1, t_2, r, p) \in T \times T \times \{0, 1\} \times [0, 1]$ where

- T: the text corpus
- t_1: a text from the text corpus
- t_2: the text t_1 is compared with
- r: the classification value which can be duplicate (1) or non-duplicate (0).
- p: the probability estimate that t_1 and t_2 are actually duplicates of each other.

Note that we use the single features from the ShallowChecker and not the total score as determined by the decision tree.

12 Evaluation

The three individual detectors (shallow, quick, deep) as well as the combined system have been evaluated on the SemDupl corpus, which is annotated for duplicates. For each text pair and each approach, a set of feature values is generated where high values indicate the texts being duplicates. These values are combined by the support vector machine classifier WLSVM (EL-Manzalawy and Honavar, 2005), which is based on LIBSVM (Chang and Lin, 2001) as described in the last section. For training this classifier, the text pairs of our corpus were used (in 10-fold cross-validation). The confusion matrices calculated for shallow and deep approaches are shown in Table 3 and Table 4.

Furthermore the knowledge extraction process was evaluated. Hyponyms, meronyms, synonyms and entailments were automatically extracted. Table 7 shows a number extracted relation hypotheses, including the number of hypotheses which were manually annotated and which were annotated as correct. The precision depending on the validation score is given in Figure 12. Part of the extracted knowledge is listed in Appendix A.

Furthermore, the number of source code lines of the individual systems were determined and is given in Table 8.

Table 3: Confusion matrices for shallow approaches. SQ=SemDupl-quick, SS=SemDupl-shallow, D=duplicate, ND=no duplicate, PD=predicted duplicate, PND=predicted non-duplicate.

	SQ		SS		SQ+SS	
	PD	PND	PD	PND	PD	PND
D	97	157	200	54	201	53
ND	16	21637	14	21639	13	21640

Table 4: Confusion matrices for deep approaches. D=duplicate, ND=no duplicate, PD=predicted duplicate, PND=predicted non-duplicate, SD=SemDupl-deep.

	SD		SD+SQ		SD+SQ+SS	
	PD	PND	PD	PND	PD	PND
D	42	212	106	148	202	52
ND	5	21648	16	21637	11	21642

Table 5: F-Measure, precision, recall, and accuracy for shallow approaches.

Measure	WCopyfind	SQ	SS	SQ+SS
F-Measure	0.259	0.529	0.855	0.859
Precision	0.830	0.858	0.935	0.939
Recall	0.154	0.382	0.787	0.791
Accuracy	0.990	0.992	0.997	0.997

Table 6: F-Measure, precision, recall, and accuracy for deep approaches.

Measure	SD	SD+SQ	SD+SQ+SS
F-Measure	0.279	0.564	0.865
Precision	0.894	0.869	0.948
Recall	0.165	0.417	0.795
Accuracy	0.990	0.993	0.997

Table 7: Extracted semantic and lexical relation hypotheses.

Type of Relation	Hypotheses	Annotated	Annotated+Correct
Synonymy	88 624	996	164
Hyponymy	391 153	29 523	7617
Meronymy	1 483 701	49 839	1421
Entailments	150 000	413	91

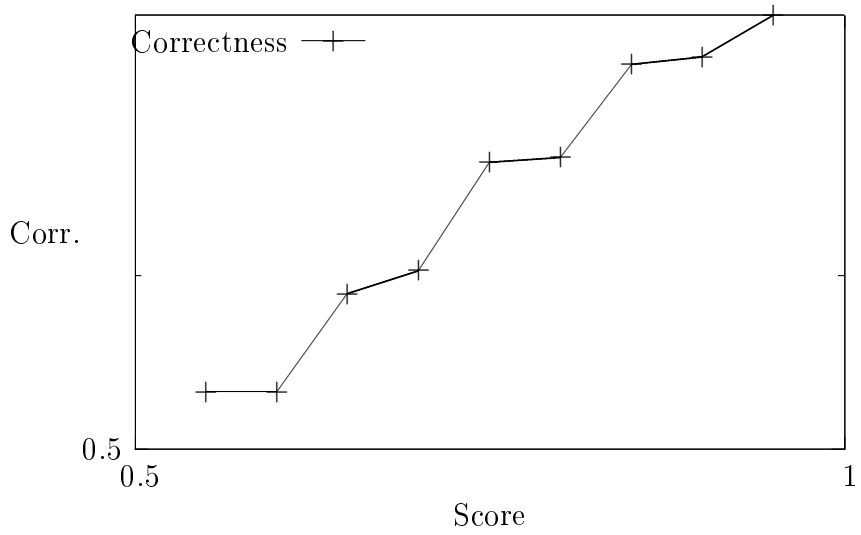

Figure 12: Precision of hyponymy hypotheses depending on score.

13 Evaluation Interpretation and Conclusions

In order to compare the results of the combined system with plagiarism detection software, WCopyfind was evaluated on our text corpus, see Table 2 and Table 5 for details. Table 5–Table 7 show the results of our approach. It can be seen that each approach of our system generates significantly better results in terms of F-measure, precision, and recall than WCopyfind (determined with confidence intervals of level 99%).

Similarly (at 95% confidence), the best combined shallow+deep approach outperforms the best shallow approach. Furthermore, F-measure, precision, and recall of the combined shallow+deep system are considerably higher than the associated values of the combination of the two shallow systems. Note that the SemDupl-deep approach can show its full potential only in more professionally constructed duplicates; for example, on the semdupl-units subcorpus (which was excluded from the evaluation because it was constructed), it performs much better than any shallow system.

Table 8: Lines of source code of the SemDupl components.

Component	Number of lines of source code
SemDupl-quick	4000
SemDupl-deep	19 000
SemDupl-shallow	8000
SemQuire	100 000

The SemDupl system shows that a deep, semantic approach for duplicate detection pays off. The combination of deep and shallow methods outperforms each single method.

In future work, we want to improve the coverage of the deep approach by further extending its knowledge bases by (semi-)automatic means.

A Extracted Knowledge

Part of the extracted knowledge is given in the tables below. Table 9 contains a selection of hyponymy hypotheses, Table 10 contains highly scored meronymy hypotheses. Synonym hypotheses are listed in Table 11. Finally, some entailment examples are shown in Table 12.
It must be remarked that automatic extraction of entailments from texts is one of the most ambitious tasks in automatic knowledge acquisition. Therefore, this task is far from being completed and has to be considered as a research field in itself. Further research should be carried out in the following directions:

- Including new methods into the automatic acquisition of entailments.
- Enlarging the base of textual corpora for this task.
- Improving the computation of scores judging the quality of the entailments.
- Refining the logical validation of entailments and finding generalizations over the achieved results (generation of axiom schemata).

Table 9: Hypernymy relations with high confidence score.

Relation	Score
SUB(*silber.1.1*, *metall.1.1*, *categ*, *situa*)	0.9737
SUB(*bronze.1.1*, *werkstoff.1.1*, *categ*, *situa*)	0.9696
SUB(*wasserdampf.1.1*, *gas.1.1*, *categ*, *situa*)	0.9696
SUB(*trompete.1.1*, *instrument.1.1*, *categ*, *situa*)	0.9591
SUB(*gold.1.1*, *metall.1.1*, *categ*, *situa*)	0.9589
SUB(*blei.1.1*, *metall.1.1*, *categ*, *situa*)	0.9572
SUB(*gitarre.1.1*, *instrument.1.1*, *categ*, *situa*)	0.9567
SUB(*blut.1.1*, *körperflüssigkeit.1.1*, *categ*, *situa*)	0.9548
SUB(*physik.1.1*, *naturwissenschaft.1.1*, *categ*, *situa*)	0.9546
SUB(*aluminium.1.1*, *leichtmetall.1.1*, *categ*, *situa*)	0.9521
SUB(*schmuck.1.1*, *gegenstand.1.1*, *categ*, *situa*)	0.9518
SUB(*kunststoff.1.1*, *material.1.1*, *categ*, *situa*)	0.9504
SUB(*buch.1.1*, *schrift.1.1*, *categ*, *situa*)	0.9486
SUB(*präsident.1.1*, *staatsoberhaupt.1.1*, *categ*, *situa*)	0.9485
SUB(*kruzifix.1.1*, *element.1.1*, *categ*, *situa*)	0.9458
SUB(*tierarzt.1.1*, *person.1.1*, *categ*, *situa*)	0.9458
SUB(*vesper.1.1*, *ding.1.1*, *categ*, *situa*)	0.9458
SUB(*sultan.1.1*, *person.1.1*, *categ*, *situa*)	0.9458
SUB(*knochen.1.1*, *gegenstand.1.1*, *categ*, *situa*)	0.9458
SUB(*aluminium.1.1*, *stoff.1.1*, *categ*, *situa*)	0.9458
SUB(*milchstrae.1.1*, *objekt.1.1*, *categ*, *situa*)	0.9458
SUB(*dolmen.1.1*, *struktur.1.1*, *categ*, *situa*)	0.9458
SUB(*gewerkschaft.1.1*, *bündnis.1.1*, *categ*, *situa*)	0.9458
SUB(*heizöl.1.1*, *stoff.1.1*, *categ*, *situa*)	0.9458
SUB(*energie.1.1*, *stoff.1.1*, *categ*, *situa*)	0.9458
SUB(*editor.1.1*, *programm.1.1*, *categ*, *situa*)	0.9458
SUB(*vollholz.1.1*, *werkstoff.1.1*, *categ*, *situa*)	0.9458
SUB(*erbauer.1.1*, *person.1.1*, *categ*, *situa*)	0.9458
SUB(*honig.1.1*, *stoff.1.1*, *categ*, *situa*)	0.9458
SUB(*regierung.1.1*, *institution.1.1*, *categ*, *situa*)	0.9458
SUB(*gummi.1.1*, *kunststoff.1.1*, *categ*, *situa*)	0.9458
SUB(*körperschaft.1.1*, *institution.1.1*, *categ*, *situa*)	0.9458
SUB(*erde.1.1*, *existenzbereich.1.1*, *categ*, *situa*)	0.9458
SUB(*alkohol.1.1*, *droge.1.1*, *categ*, *situa*)	0.9452
SUB(*pflanzenöl.1.1*, *flüssigkeit.1.1*, *categ*, *situa*)	0.9425
SUB(*zoll.1.1*, *behörde.1.1*, *categ*, *situa*)	0.9424
SUB(*klavier.1.1*, *instrument.1.1*, *categ*, *situa*)	0.9415
SUB(*kupfer.1.1*, *material.1.1*, *categ*, *situa*)	0.9377
SUB(*biologie.1.1*, *naturwissenschaft.1.1*, *categ*, *situa*)	0.9375
SUB(*vater.1.1*, *person.1.1*, *categ*, *situa*)	0.9368

Table 9: Hypernymy relations with high confidence score.

Relation	Score
SUB(*kleidung.1.1*, *ding.1.1*, *categ*, *situa*)	0.9356
SUB(*phosphor.1.1*, *element.1.1*, *categ*, *situa*)	0.9356
SUB(*schule.1.1*, *gebäude.1.1*, *categ*, *situa*)	0.9352
SUB(*schlagzeug.1.1*, *instrument.1.1*, *categ*, *situa*)	0.9318
SUB(*methan.1.1*, *treibhausgas.1.1*, *categ*, *situa*)	0.9315
SUB(*frucht.1.1*, *pflanzenteil.1.1*, *categ*, *situa*)	0.9298
SUB(*englisch.2.1*, *sprache.1.1*, *categ*, *situa*)	0.9291
SUB(*käse.1.1*, *milchprodukt.1.1*, *categ*, *situa*)	0.9286
SUB(*chemie.1.1*, *naturwissenschaft.1.1*, *categ*, *situa*)	0.9281
SUB(*holz.1.1*, *stoff.1.1*, *categ*, *situa*)	0.9281
SUB(*aluminiumoxid.1.1*, *material.1.1*, *categ*, *situa*)	0.9279
SUB(*wysiwyg-texteditor.1.1*, *programm.1.1*, *categ*, *situa*)	0.9279
SUB(*cayennepfeffer.1.1*, *gewürz.1.1*, *categ*, *situa*)	0.9279
SUB(*carolinalilie.1.1*, *pflanze.1.1*, *categ*, *situa*)	0.9279
SUB(*havelland.2.1*, *landschaft.1.1*, *categ*, *situa*)	0.9279
SUBS(*karnevalssession.1.1*, *veranstaltung.1.1*, *categ*, *situa*)	0.9279
SUB(*exoskeletts.1.1*, *tier.1.1*, *categ*, *situa*)	0.9279
SUB(*zypressenwolfsmilch.1.1*, *pflanze.1.1*, *categ*, *situa*)	0.9279
SUB(*qawwali.1.1*, *musikstil.1.1*, *categ*, *situa*)	0.9279
SUB(*hammaburg.1.1*, *kirche.1.1*, *categ*, *situa*)	0.9279
SUB(*verbandsvorsitzend.1.1*, *mitglied.1.1*, *categ*, *situa*)	0.9279
SUB(*untergrund.1.2*, *bestandteil.1.1*, *categ*, *situa*)	0.9279
SUB(*knaanischen.1.1*, *sprache.1.1*, *categ*, *situa*)	0.9279
SUB(*jugendbildungswerk.1.2*, *organisation.1.1*, *categ*, *situa*)	0.9279
SUB(*wrangelschen.1.1*, *gebäude.1.1*, *categ*, *situa*)	0.9279
SUB(*kriebelmücke.1.1*, *insekt.1.1*, *categ*, *situa*)	0.9279
SUB(*dozentin.1.1*, *frau.1.1*, *categ*, *situa*)	0.9279
SUB(*betelwachs.1.1*, *produkt.1.1*, *categ*, *situa*)	0.9279
SUB(*beteiligungshaushalt.1.1*, *form.1.1*, *categ*, *situa*)	0.9279
SUB(*neokonservatismus.1.1*, *form.1.1*, *categ*, *situa*)	0.9279
SUB(*tongji-universität.1.1*, *einrichtung.1.2*, *categ*, *situa*)	0.9279
SUB(*phosphor.1.1*, *mineralien.1.1*, *categ*, *situa*)	0.9279
SUB(*plektrum.1.1*, *musiker.1.1*, *categ*, *situa*)	0.9279
SUB(*sünderin.1.1*, *film.1.1*, *categ*, *situa*)	0.9279
SUB(*senf.1.1*, *gewürz.1.1*, *categ*, *situa*)	0.9261
SUB(*aluminium.1.1*, *metall.1.1*, *categ*, *situa*)	0.9249
SUB(*gemüse.1.1*, *produkt.1.1*, *categ*, *situa*)	0.9231

Table 10: Meronymy hypotheses with high confidence score.

Relation	Score
$\text{SUB}(x, haut.1.1, categ, categ) \wedge \text{PARS}(x, fisch.1.1, categ, proto)$	1.0000
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, haus.1.1, categ, proto)$	0.9999
$\text{SUB}(x, motor.1.1, categ, categ) \wedge \text{PARS}(x, fahrzeug.1.1, categ, proto)$	0.9993
$\text{PARS}(gepäckwagen.1.1, zug.1.1, proto, proto)$	0.9990
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, auto.1.1, categ, proto)$	0.9982
$\text{SUB}(x, kopf.1.1, categ, categ) \wedge \text{PARS}(x, mensch.1.1, categ, proto)$	0.9975
$\text{SUB}(x, ufer.1.1, categ, categ) \wedge \text{PARS}(x, flus.1.1, categ, proto)$	0.9973
$\text{SUB}(x, schwanz.1.1, categ, categ) \wedge \text{PARS}(x, tier.1.1, categ, proto)$	0.9973
$\text{SUB}(x, turm.1.1, categ, categ) \wedge \text{PARS}(x, burg.1.1, categ, proto)$	0.9971
$\text{SUB}(x, kind.1.1, categ, categ) \wedge \text{PARS}(x, paar.3.1, categ, proto)$	0.9967
$\text{SUB}(x, wagen.1.1, categ, categ) \wedge \text{PARS}(x, zug.1.1, categ, proto)$	0.9964
$\text{SUB}(x, fakultät.1.1, categ, categ) \wedge \text{SUBM}(x, hochschule.1.1, categ, ktype)$	0.9964
$\text{SUB}(x, zeiger.1.1, categ, categ) \wedge \text{PARS}(x, uhr.1.1, categ, proto)$	0.9963
$\text{SUB}(x, kopf.1.1, categ, categ) \wedge \text{PARS}(x, fisch.1.1, categ, proto)$	0.9962
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, saal.1.1, categ, proto)$	0.9962
$\text{PARS}(fassade.1.1, querhaus.1.1, categ, situa)$	0.9962
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, hauptschiff.1.1, categ, proto)$	0.9962
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, seitenschiff.1.1, categ, proto)$	0.9962
$\text{SUB}(x, giebel.1.1, categ, categ) \wedge \text{PARS}(x, gebäude.1.1, categ, proto)$	0.9961
$\text{PARS}(haut.1.1, auge.1.1, categ, situa)$	0.9959
$\text{SUB}(x, fassade.1.1, categ, categ) \wedge \text{PARS}(x, haus.1.1, categ, proto)$	0.9959
$\text{SUB}(x, fassade.1.1, categ, categ) \wedge \text{PARS}(x, nachbarhaus.1.1, categ, proto)$	0.9959
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, neubau.1.1, categ, proto)$	0.9956
$\text{SUB}(x, bau.1.1, categ, categ) \wedge \text{PARS}(x, gebiet.1.1, categ, proto)$	0.9956
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, fahrzeug.1.1, categ, proto)$	0.9956
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, dom.1.1, categ, proto)$	0.9955
$\text{SUB}(x, hafen.1.1, categ, categ) \wedge \text{PARS}(x, schiff.1.1, categ, proto)$	0.9954
$\text{SUB}(x, herz.1.1, categ, categ) \wedge \text{PARS}(x, mensch.1.1, categ, proto)$	0.9954
$\text{SUB}(x, netzhaut.1.1, categ, categ) \wedge \text{PARS}(x, auge.1.1, categ, proto)$	0.9953
$\text{PARS}(ortsteil.1.1, stadt.1.1, proto, proto)$	0.9952
$\text{SUB}(x, haut.1.1, categ, categ) \wedge \text{PARS}(x, wirtstier.1.1, categ, proto)$	0.9952
$\text{SUB}(x, haut.1.1, categ, categ) \wedge \text{PARS}(x, tier.1.1, categ, proto)$	0.9952
$\text{SUB}(x, haut.1.1, categ, categ) \wedge \text{PARS}(x, frosch.1.1, categ, proto)$	0.9952
$\text{SUB}(x, haut.1.1, categ, categ) \wedge \text{PARS}(x, arm.1.1, categ, proto)$	0.9952
$\text{SUB}(x, musik.1.1, categ, categ) \wedge \text{PARS}(x, szene.1.1, categ, proto)$	0.9951
$\text{SUB}(x, dorf.1.1, categ, categ) \wedge \text{PARS}(x, gemeinde.1.1, categ, proto)$	0.9951
$\text{PARS}(gegenstand.1.1, welt.1.1, proto, proto)$	0.9951
$\text{SUB}(x, erdgeschoss.1.1, categ, categ) \wedge \text{PARS}(x, haus.1.1, categ, proto)$	0.9951
$\text{SUB}(x, etage.1.1, categ, categ) \wedge \text{PARS}(x, haus.1.1, categ, proto)$	0.9951
$\text{SUB}(x, erdgeschoss.1.1, categ, categ) \wedge \text{PARS}(x, gebäude.1.1, categ, proto)$	0.9950

Table 10: Meronymy hypotheses with high confidence score.

Relation	Score
$\text{SUB}(x, \textit{sockel.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{gebäude.1.1}, \textit{categ}, \textit{proto})$	0.9950
$\text{SUB}(x, \textit{innenraum.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{gebäude.1.1}, \textit{categ}, \textit{proto})$	0.9950
$\text{SUB}(x, \textit{fundament.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{gebäude.1.1}, \textit{categ}, \textit{proto})$	0.9950
$\text{SUB}(x, \textit{obergescho.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{haus.1.1}, \textit{categ}, \textit{proto})$	0.9950
$\text{SUB}(x, \textit{fundament.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{haus.1.1}, \textit{categ}, \textit{proto})$	0.9950
$\text{SUB}(x, \textit{dach.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{gebäude.1.1}, \textit{categ}, \textit{proto})$	0.9949
$\text{SUB}(x, \textit{spitzdach.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{turm.1.1}, \textit{categ}, \textit{proto})$	0.9949
$\text{SUB}(x, \textit{wagen.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{d-zug.1.1}, \textit{categ}, \textit{proto})$	0.9949
$\text{PARS}(\textit{bord.1.1}, \textit{flugzeugträger.1.1}, \textit{proto}, \textit{proto})$	0.9949
$\text{SUB}(x, \textit{bord.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{raumschiff.1.1}, \textit{categ}, \textit{proto})$	0.9949
$\text{SUB}(x, \textit{bord.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{boot.1.1}, \textit{categ}, \textit{proto})$	0.9949
$\text{SUB}(x, \textit{bord.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{satellit.1.1}, \textit{categ}, \textit{proto})$	0.9949
$\text{SUB}(x, \textit{stockwerk.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{gebäude.1.1}, \textit{categ}, \textit{proto})$	0.9949
$\text{SUB}(x, \textit{bord.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{fangschiff.1.1}, \textit{categ}, \textit{categ})$	0.9949
$\text{SUB}(x, \textit{fassade.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{gebäude.1.1}, \textit{categ}, \textit{proto})$	0.9949
$\text{SUB}(x, \textit{fraktion.1.1}, \textit{categ}, \textit{categ}) \wedge \text{SUBM}(x, \textit{partei.1.1}, \textit{categ}, \textit{ktype})$	0.9947
$\text{SUB}(x, \textit{bündnis.1.1}, \textit{categ}, \textit{categ}) \wedge \text{SUBM}(x, \textit{partei.1.1}, \textit{categ}, \textit{ktype})$	0.9947
$\text{SUB}(x, \textit{titel.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{werk.1.1}, \textit{categ}, \textit{proto})$	0.9946
$\text{SUB}(x, \textit{ast.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{baum.1.1}, \textit{categ}, \textit{proto})$	0.9946
$\text{SUB}(x, \textit{wurzel.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{baum.1.1}, \textit{categ}, \textit{proto})$	0.9946
$\text{SUB}(x, \textit{schnauze.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{tier.1.1}, \textit{categ}, \textit{proto})$	0.9946
$\text{SUB}(x, \textit{körper.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{tier.1.1}, \textit{categ}, \textit{proto})$	0.9946
$\text{SUB}(x, \textit{körper.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{pferd.1.1}, \textit{categ}, \textit{proto})$	0.9946
$\text{SUB}(x, \textit{deckel.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{topf.1.1}, \textit{categ}, \textit{proto})$	0.9945
$\text{SUB}(x, \textit{hilfsmotor.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{segelschiff.1.1}, \textit{categ}, \textit{proto})$	0.9945
$\text{SUB}(x, \textit{antriebshebel.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{maschine.1.1}, \textit{categ}, \textit{proto})$	0.9945
$\text{SUB}(x, \textit{dach.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{waisenhaus.1.1}, \textit{categ}, \textit{proto})$	0.9945
$\text{SUB}(x, \textit{dach.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{mittelschiff.1.1}, \textit{categ}, \textit{proto})$	0.9945
$\text{SUB}(x, \textit{dach.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{wohngebäude.1.1}, \textit{categ}, \textit{proto})$	0.9945
$\text{SUB}(x, \textit{dach.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{reichstagsgebäude.1.1}, \textit{categ}, \textit{proto})$	0.9945
$\text{SUB}(x, \textit{dach.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{flughafengebäude.1.1}, \textit{categ}, \textit{proto})$	0.9945
$\text{SUB}(x, \textit{dach.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{hauptgebäude.1.1}, \textit{categ}, \textit{proto})$	0.9945
$\text{SUB}(x, \textit{dach.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{kirchenschiff.1.1}, \textit{categ}, \textit{proto})$	0.9945
$\text{SUB}(x, \textit{dach.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{wolkenkratzer.1.1}, \textit{categ}, \textit{proto})$	0.9945
$\text{SUB}(x, \textit{dach.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{nachbarhaus.1.1}, \textit{categ}, \textit{proto})$	0.9945
$\text{SUB}(x, \textit{dach.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{schulhaus.1.1}, \textit{categ}, \textit{proto})$	0.9945
$\text{SUB}(x, \textit{dach.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{sockelbau.1.1}, \textit{categ}, \textit{proto})$	0.9945
$\text{SUB}(x, \textit{dach.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{münster.1.1}, \textit{categ}, \textit{proto})$	0.9945
$\text{SUB}(x, \textit{dach.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{haupthaus.1.1}, \textit{categ}, \textit{proto})$	0.9945
$\text{SUB}(x, \textit{dach.1.1}, \textit{categ}, \textit{categ}) \wedge \text{PARS}(x, \textit{schulgebäude.1.1}, \textit{categ}, \textit{proto})$	0.9945

Table 10: Meronymy hypotheses with high confidence score.

Relation	Score
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, fahrerhaus.1.1, categ, proto)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, wasserhochbehälter.1.1, categ, proto)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, rauchsalon.1.1, categ, proto)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, parlamentsgebäude.1.1, categ, proto)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, westbau.1.1, categ, proto)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, zugfahrzeug.1.1, categ, proto)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, kraftfahrzeug.1.1, categ, proto)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, bus.1.1, categ, proto)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, studio.1.1, categ, proto)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, torbau.1.1, categ, proto)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, priesterhaus.1.1, categ, proto)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, landhaus.1.1, categ, proto)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, glockenhaus.1.1, categ, proto)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, führerhaus.1.1, categ, proto)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, hochbau.1.1, categ, categ)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, zwerchhaus.1.1, categ, proto)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, hubschrauber.1.1, categ, proto)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, mittelbau.1.1, categ, proto)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, oiya\text{-}gebäude.1.1, categ, categ)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, festsaal.1.1, categ, proto)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, bahnhofsgebäude.1.1, categ, categ)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, amtsgebäude.1.1, categ, categ)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, reaktorgebäude.1.1, categ, proto)$	0.9945
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, generatorhaus.1.1, categ, categ)$	0.9945
$\text{SUB}(x, umschlag.1.1, categ, categ) \wedge \text{PARS}(x, hafen.1.1, categ, proto)$	0.9944
$\text{SUB}(x, fu.1.1, categ, categ) \wedge \text{PARS}(x, tier.1.1, categ, proto)$	0.9944
$\text{SUB}(x, etage.1.1, categ, categ) \wedge \text{PARS}(x, gebäude.1.1, categ, proto)$	0.9944
$\text{SUB}(x, vorderseite.1.1, categ, categ) \wedge \text{PARS}(x, haus.1.1, categ, proto)$	0.9944
$\text{SUB}(x, schauseite.1.1, categ, categ) \wedge \text{PARS}(x, haus.1.1, categ, proto)$	0.9944
$\text{SUB}(x, titel.1.1, categ, categ) \wedge \text{PARS}(x, novelle.1.1, categ, proto)$	0.9943
$\text{SUB}(x, regierung.1.1, categ, categ) \wedge \text{PARS}(x, welt.1.1, categ, proto)$	0.9943
$\text{SUB}(x, fu.1.1, categ, categ) \wedge \text{PARS}(x, patient.1.1, categ, proto)$	0.9943
$\text{SUB}(x, hand.1.1, categ, categ) \wedge \text{PARS}(x, mensch.1.1, categ, proto)$	0.9943
$\text{SUB}(x, gesicht.1.1, categ, categ) \wedge \text{PARS}(x, mensch.1.1, categ, proto)$	0.9943
$\text{SUB}(x, ohr.1.1, categ, categ) \wedge \text{PARS}(x, mensch.1.1, categ, proto)$	0.9943
$\text{SUB}(x, auge.1.1, categ, categ) \wedge \text{PARS}(x, mensch.1.1, categ, proto)$	0.9943
$\text{SUB}(x, dach.1.1, categ, categ) \wedge \text{PARS}(x, wagen.1.1, categ, proto)$	0.9942
$\text{SUB}(x, kopf.1.1, categ, categ) \wedge \text{PARS}(x, tier.1.1, categ, proto)$	0.9942
$\text{SUB}(x, hornhaut.1.1, categ, categ) \wedge \text{PARS}(x, auge.1.1, categ, proto)$	0.9942
$\text{SUB}(x, konstruktion.1.1, categ, categ) \wedge \text{PARS}(x, wagen.1.1, categ, proto)$	0.9942

Table 11: Synonymy hypotheses with high confidence score.

Relation	Score
SYNO(*kopfschmerz.1.1*, *cephalgie.1.1*, categ, categ)	0.7107
SYNO(*parallelepiped.1.1*, *spat.1.1*, categ, categ)	0.7107
SYNO(*kopfschmerz.1.1*, *kephalgie.1.1*, categ, categ)	0.7107
SYNO(*kopfschmerz.1.1*, *kephalalgie.1.1*, categ, categ)	0.7107
SYNO(*kopfschmerz.1.1*, *zephalgie.1.1*, categ, categ)	0.7107
SYNO(*kopfschmerz.1.1*, *cephalaea.1.1*, categ, categ)	0.7107
SYNO(*refsum-syndrom.1.1*, *refsum-thiébaut-krankheit.1.1*, categ, categ)	0.7107
SYNO(*honigmagen.1.1*, *honigblase.1.1*, categ, categ)	0.7107
SYNO(*rhesusinkompatibilität.1.1*, *rh-inkompatibilität.1.1*, categ, categ)	0.7107
SYNO(*ott-zeichen.1.1*, *ott-maß.1.1*, categ, categ)	0.7107
SYNO(*versetzungszeichen.1.1*, *akzidens.1.1*, categ, categ)	0.7107
SYNO(*surrogatmarker.1.1*, *surrogatparameter.1.1*, categ, categ)	0.7107
SYNO(*glucose-6-phosphat.1.1*, *robisonester.1.1*, categ, categ)	0.7107
SYNO(*vierhügelplatte.1.1*, *lamina.1.1*, categ, categ)	0.7107
SYNO(*nierenzellkarzinom.1.1*, *grawitz-tumor.1.1*, categ, categ)	0.7107
SYNO(*hornschwiele.1.1*, *tylositas.1.1*, categ, categ)	0.7107
SYNO(*rhesusinkompatibilität.1.1*,*rhesusunverträglichkeit.1.1*, categ, categ)	0.7107
SYNO(*verkehrsleistung.1.1*, *beförderungsleistung.1.1*, categ, categ)	0.7107
SYNO(*kreiselkäfer.1.1*, *breithalskäfer.1.1*, categ, categ)	0.7107
SYNO(*honigmagen.1.1*, *sozialmagen.1.1*, categ, categ)	0.7107
SYNO(*verkehrsleistung.1.1*, *transportleistung.1.1*, categ, categ)	0.7107
SYNO(*taiwan_ taoyuan_ international_ airport.0*, *tty_ airport.0*, categ, categ)	0.7107
SYNO(*chiang_ kai-shek_ international_ airport.0*, *cks_ airport.0*, categ, categ)	0.7107
SYNO(*karnaugh-veitch-diagramm.1.1*, *kv-diagramm.1.1*, categ, categ)	0.7107
SYNO(*propagandistin.1.1*, *propagator.1.1*, categ, categ)	0.7107
SYNO(*schober-zeichen.0*, *schober'sches_ zeichen.0*, categ, categ)	0.7107
SYNO(*alpha-1-antitrypsinmangel.1.1*, *laurell-eriksson-syndrom.1.1*, categ, categ)	0.7107
SYNO(*refsum-syndrom.1.1*, *heredopathia.1.1*, categ, categ)	0.7107
SYNO(*propagandistin.1.1*, *verkaufsförderer.1.1*, categ, categ)	0.7107
SYNO(*parallelepiped.1.1*, *parallelflach.2.1*, categ, categ)	0.7107
SYNO(*alpha-1-antitrypsinmangel.1.1*, *proteaseinhibitormangel.1.1*, categ, categ)	0.7107
SYNO(*versetzungszeichen.1.1*, *akzidentalen.1.1*, categ, categ)	0.7107
SYNO(*lapacho.1.1*, *iperoxo.1.1*, categ, categ)	0.7107
SYNO(*la_ crosse-enzephalitis.0*, *crosse_ la.0*, categ, categ)	0.7107
SYNO(*mbtps1.0*, *s1p.0*, categ, categ)	0.7107
SYNO(*laurent.0*, *saint_ laurent.0*, categ, categ)	0.7107

Table 11: Synonymy hypotheses with high confidence score.

Relation	Score
SYNO(*zenionidae.1.1*, *zeniontidae.1.1*, categ, categ)	0.7107
SYNO(*parallelepiped.1.1*, *parallelotop.1.1*, categ, categ)	0.7107
SYNO(*erythropoetin.1.1*, *erythropoietin.1.1*, categ, categ)	0.7107
SYNO(*erythropoetin.1.1*, *epoetin.1.1*, categ, categ)	0.7107
SYNO(*zenionidae.1.1*, *macrurocyttidae.1.1*, categ, categ)	0.7107
SYNO(*laurent.0*, *pinot_saint_laurent.0*, categ, categ)	0.7107
SYNO(*fingerperimetrie.1.1*, *konfrontationsperimetrie.1.1*, categ, categ)	0.7107
SYNO(*perimetrie.1.1*, *goldmannperimetrie.1.1*, categ, categ)	0.7107
SYNO(*perimetrie.1.1*, *computerperimetrie.1.1*, categ, categ)	0.7107
SYNO(*invagination.1.1*, *intussuszeption.1.1*, categ, categ)	0.7107
SYNO(*nierenzellkarzinom.1.1*, *hypernephrom.1.1*, categ, categ)	0.7107
SYNO(*colitis.1.1*, *kollagenkolitis.1.1*, categ, categ)	0.7107
SYNO(*hornschwiele.1.1*, *tylosis.1.1*, categ, categ)	0.7107
SYNO(*colitis.1.1*, *kollagencolitis.1.1*, categ, categ)	0.7107
SYNO(*perimetrie.1.1*, *schwellenperimetrie.1.1*, categ, categ)	0.7107
SYNO(*ethylenimin.1.1*, *aziridin.1.1*, categ, categ)	0.7106
SYNO(*basaliom.1.1*, *basalzellkarzinom.1.1*, categ, categ)	0.7106
SYNO(*hornschwiele.1.1*, *tylom.1.1*, categ, categ)	0.7106
SYNO(*art.1.1*, *alpha-fehler.1.1*, categ, categ)	0.7105
SYNO(*filzstift.1.1*, *filzschreiber.1.1*, categ, categ)	0.7105
SYNO(*verhalten.1.1*, *verhaltensformung.1.1*, categ, categ)	0.7105
SYNO(*filzstift.1.1*, *filzmaler.1.1*, categ, categ)	0.7105
SYNO(*filzstift.1.1*, *faserschreiber.1.1*, categ, categ)	0.7105
SYNO(*filzstift.1.1*, *fasermaler.1.1*, categ, categ)	0.7105
SYNO(*krankheit.1.1*, *morbus.1.1*, categ, categ)	0.7105
SYNO(*loránd-eötvös-universität_budapest.0*, *elte.0*,categ,categ)	0.7105
SYNO(*amyotrophe_lateralsklerose.0*, *als.0*, categ, categ)	0.7105
SYNO(*sender_policy_framework.0*, *spf.0*, categ, categ)	0.7105
SYNO(*financial_reporting.0*, *icofr.0*, categ, categ)	0.7105
SYNO(*Åtvidabergs_ff.0*, *åff.0*, categ, categ)	0.7105
SYNO(*unshiu_mikan.0*, *mikan.0*, categ, categ)	0.7105
SYNO(*initial-v.0*, *dulv.0*, categ, categ)	0.7105
SYNO(*hunter's_rank.0*, *hr.0*, categ, categ)	0.7105
SYNO(*bande_dessinée.0*, *dnap.0*, categ, categ)	0.7105
SYNO(*radboud-universität_nimwegen.0*, *ru.0*, categ, categ)	0.7105
SYNO(*sociaal-democratische_arbeiderspartij.0*, *sdap.0*, categ, categ)	0.7105
SYNO(*azienda_trasporti_milanesi.0*, *atm.0*, categ, categ)	0.7105
SYNO(*koca_mimar_sinan_aga.0*, *mimar_sinan.0*, categ, categ)	0.7105
SYNO(*integrada.0*, *oilb.0*,categ,categ)	0.7105

Table 12: Selection of entailments.

Premise	Conclusion	Score
TEMP(d, b)∧ SUBS(d, *tod.1.1*)∧ AFF(d, a)	SUBS(d, *versterben.1.1*)∧ TEMP(d, b)∧ AFF(d, a)	0.9081
SUBS(d, *versterben.1.1*)∧ TEMP(d, b)∧ AFF(d, a)	TEMP(d, b)∧ SUBS(d, *tod.1.1*)∧ AFF(d, a)	0.9081
SUBS(e, *entdecken.1.1*)∧ OBJ(e, b)∧ EXP(e, a)	SUBS(m, *entdeckung.1.1*)∧ TEMP(n, *present.0*)∧ OBJ(m, b)∧ ASSOC(n, m)∧ AGT(n, a)	0.5567
SUBS(e, *entdeckung.1.1*)∧ TEMP(f, *present.0*)∧ OBJ(e, b)∧ ASSOC(f, e)∧ AGT(f, a)	SUBS(m, *entdecken.1.1*)∧ OBJ(m, b)∧ EXP(m, a)	0.5567
SUBS(e, *bekräftigen.1.1*)∧ TEMP(e, *past.0*)∧ OBJ(e, d)∧ SUB(d, *forderung.1.1*)∧ OBJ(d, a) ∧ ANTE(b, e)	SEMREL(e, d)∧ SUBS(d, *aussprechen.1.4*)∧ PURP(d, b)∧ AGT(d, a)	0.5464
ATTCH(b, e)∧ TEMP(d, c)∧ SUBS(d, *gewinnen.1.1*)∧ EXP(d, a)	ASSOC(e, b)∧ TEMP(e, *past.0*)∧ TEMP(e, c)∧ SUBS(d, *sieg.1.1*)∧ SCAR(e, d)∧ EXP(d, a)	0.5464
ASSOC(e, b)∧ TEMP(e, *past.0*)∧ TEMP(e, c)∧ SUBS(d, *sieg.1.1*)∧ SCAR(e, d)∧ EXP(d, a)	ATTCH(b, e)∧ TEMP(d, c)∧ SUBS(d, *gewinnen.1.1*)∧ EXP(d, a)	0.5464
SUBS(d, *sterben.1.1*)∧ TEMP(d, b)∧ AFF(d, a)	SUBS(d, *versterben.1.1*)∧ TEMP(d, b)∧ AFF(d, a)	0.5461
SUBS(d, *versterben.1.1*)∧ TEMP(d, b)∧ AFF(d, a)	SUBS(d, *sterben.1.1*)∧ TEMP(d, b)∧ AFF(d, a)	0.5461
SUBS(e, *gewinnen.1.1*)∧ TEMP(e, c)∧ ATTCH(b, f)∧ EXP(e, a)	TEMP(m, *past.0*)∧ SUBS(n, *sieg.1.1*)∧ TEMP(m, c)∧ ASSOC(m, b)∧ SCAR(m, n)∧ EXP(n, a)	0.5459
TEMP(e, *past.0*)∧ SUBS(f, *sieg.1.1*)∧ TEMP(e, c)∧ ASSOC(e, b)∧ SCAR(e, f)∧ EXP(f, a)	SUBS(m, *gewinnen.1.1*)∧ TEMP(m, c)∧ ATTCH(b, n)∧ EXP(m, a)	0.5459
SUBS(e, *entdecken.1.1*)∧ OBJ(e, b)∧ EXP(e, a)	TEMP(m, *erst.2.1*)∧ SUBS(m, *landen.1.2*)∧ TEMP(m, *past.0*)∧ *IN(n, b)∧ LOC(m, n)∧ AGT(m, a)	0.5461

Table 12: Selection of entailments (ctd.).

Premise	Conclusion	Score
TEMP(e, *erst.2.1*)∧ SUBS(e, *landen.1.2*)∧ TEMP(e, *past.0*)∧ *IN(f, b)∧ LOC(e, f)∧ AGT(e, a)	SUBS(m, *entdecken.1.1*)∧ OBJ(m, b)∧ EXP(m, a)	0.5461
SUB(e, *forderung.1.1*)∧ ANTE(b, e)∧ OBJ(d, e)∧ TEMP(d, *present.0*)∧ SUBS(d, *pochen.1.1*)∧ AGT(d, a)	OBJ(d, b)∧ SUBS(d, *fordern.1.1*)∧ AGT(d, a)	0.5461
SUBS(e, *entdecken.1.1*)∧ OBJ(e, b)∧ EXP(e, a)	SUBR(m, *attch.0*)∧ SUBS(n, *entdecken.1.2*)∧ TEMP(o, *past.0*)∧ ARG1(m, p)∧ ARG2(m, b)∧ OBJ(n, b)∧ MCONT(o, m)∧ MEXP(o, a)	0.5461
SEMREL(e, d)∧ SUBS(d, *aussprechen.1.4*)∧ PURP(d, b)∧ AGT(d, a)	SUB(e, *forderung.1.1*)∧ ANTE(b, e)∧ OBJ(d, e)∧ TEMP(d, *present.0*)∧ SUBS(d, *pochen.1.1*)∧ AGT(d, a)	0.5461
SUB(e, *forderung.1.1*)∧ ANTE(b, e)∧ OBJ(d, e)∧ TEMP(d, *present.0*)∧ SUBS(d, *pochen.1.1*)∧ AGT(d, a)	SEMREL(e, d)∧ SUBS(d, *aussprechen.1.4*)∧ PURP(d, b)∧ AGT(d, a)	0.5461
OBJ(d, b)∧ SUBS(d, *pochen.1.1*)∧ AGT(d, a)	SUBS(e, *bekräftigen.1.1*) TEMP(e, *past.0*)∧ OBJ(e, d)∧ SUB(d, *forderung.1.1*)∧ OBJ(d, a) ∧ ANTE(b, e)∧	0.5461
SUBS(e, *bekräftigen.1.1*)∧ TEMP(e, *past.0*)∧ OBJ(e, d)∧ SUB(d, *forderung.1.1*)∧ OBJ(d, a) ∧ ANTE(b, e)	OBJ(d, b)∧ SUBS(d, *pochen.1.1*)∧ AGT(d, a)	0.5461
OBJ(d, b)∧ SUBS(d, *pochen.1.1*)∧ AGT(d, a)	OBJ(d, b)∧ SUBS(d, *fordern.1.1*)∧ AGT(d, a)	0.5461
OBJ(d, b)∧ SUBS(d, *fordern.1.1*)∧ AGT(d, a)	OBJ(d, b)∧ SUBS(d, *pochen.1.1*)∧ AGT(d, a)	0.5461
OBJ(e, b)∧ *MODS(f, *besonders.1.1*, *fordern.1.1*)∧ SUBS(e, f)∧ SOURC(e, d)∧ PRED(d, *reihe.1.1*)∧ ATTCH(a, d)	SEMREL(e, d)∧ SUBS(d, *aussprechen.1.4*)∧ PURP(d, b)∧ AGT(d, a)	0.5461
OBJ(d, b)∧ SUBS(d, *pochen.1.1*)∧ AGT(d, a)	SUB(d, *forderung.1.1*)∧ ANTE(b, d)∧ OBJ(d, a)	0.5461
SUB(e, *erfinder.1.1*)∧ ARG2(f, g)∧ EQU(a, e)∧ OBJ(h, b)∧ ARG1(f, a)∧ AGT(h, e)	SUBS(m, *erfinden.1.1*)∧ AGT(m, b)∧ AGT(m, a)	0.1131
SUBS(e, *erfinden.1.1*)∧ AGT(e, b)∧ AGT(e, a)	SUB(m, *erfinder.1.1*)∧ ARG2(n, o)∧ EQU(a, m)∧ OBJ(p, b)∧ ARG1(n, a)∧ AGT(p, m)	0.1131

B Program Documentation and Manuals

The SemDupl system consist of three individual detectors, a combination module, and a GUI. The GUI is the easiest access to the detectors, but hides most of their details. Therefore, the GUI is mainly intended for the casual user, while for experimentation the single detectors should be called with explicit parameter settings as described in their manual sections below.

B.1 SemDupl-deep

The deep detector is implemented in the Scheme program `semdupl`. This program is contained in the directory `semdupl-deep` and can be built by calling `make`. The main module is `semdupl`, which needs several other modules as defined in `semdupl.mod`.

SemDupl-deep uses a special key-value store in order to access large collections of semantic networks efficiently. The current implementation employs a Tokyo Tyrant server for this. It must be started (currently on the server named ki205) with the following script:

```
start-gnet-servers.205
```

First, texts must be registered with SemDupl-deep. To register the text poe13, issue the following command:

```
semdupl --register poe13 ...
```

Registration assigns a unique ID consisting of four characters to each text. These IDs must be used in the following if referring to texts.

If later on, a text with ID Fzzz should be discarded, the following command suffices:

```
semdupl --delete Fzzz ...
```

This deletes the registration information and removes all representations for the text with ID Fzzz.

Before a text can be used in the deep detection of duplicates it must be analyzed by the parser and the coreference resolver. For text Fzzz, the command is:

```
semdupl --analyze Fzzz ...
```

To process a long list of texts, the following command is often more convenient:

```
xargs -a <file-with-text-ids> semdupl --analyze
```

In addition, some metadata statistics (like number of paragraphs and sentences) must be collected for SemDupl-deep:

```
find /cl/sd/net/ -type f | sort |
  xargs wocadi-tool -collect-metadata > /cl/registry/db_metadata.tsv
cd /cl/registry/
tchmgr importtsv db_metadata.tch < db_metadata.tsv
```

To detect duplicates for a given text (e.g. with id Fzzz), the test option is needed:

```
semdupl --test Fzzz
```

This command accepts several additional options:

–**start-sentence <N>** : For detection, sentences in the candidate text are only considered from the Nth sentence onwards. In other words, the first N-1 sentences are ignored.

–**stop-sentence <N>** : The detection stops after the Nth sentence in the candidate text. The main purpose for this would be to speed up the detection because the complexity of SemDupl-deep is linear to the number of sentences in the candidate text.

–**html** : Output is written in HTML format instead of plain text format.

–**show-sentence** : The corresponding sentence pairs between the candidate text and the duplicate at hand are shown in the output.

To transform the output of SemDupl-deep into a format accepted by the evaluation module, the XML results option (short: -x) is needed, e.g.

```
semdupl --xml-results semdupl-deep.log > eval.semdupl-deep.xml
```

B.2 SemDupl-shallow

All options of this program can be set by command line arguments. Since SemDupl-shallow is usually called with certain attributes which are normally constant, there is the possibilities to specify default options in a configuration file in XML format. If not specified otherwise the file `CErkenner.properties` contains the configuration. Listing 1 shows the settings which are given in this file.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM
   "http://java.sun.com/dtd/properties.dtd">
<properties>
  <entry key="tree">data/tree.mod</entry>
  <entry key="stopwords">data/stop.dat</entry>
  <entry key="synonyms">data/synonyme.dat</entry>
  <entry key="interpretation">standard</entry>
  <entry key="corpusdir">corpus</entry>
  <entry key="aspellpath">/usr/bin/aspell</entry>
  <entry key="output">stdout</entry>
  <entry key="loglevel">warning</entry>
  <entry key="outlevel">duplikat keinduplikat verdacht
     </entry>
  <entry key="error">stderr</entry>
</properties>
```

Listing 1: Contents of the file `CErkenner.properties` with default values of SemDupl-shallow.

If a setting is specified both in the configuration file and in the command line options, the options given in the command line is used. In this way, it is possible to override default configuration entries without changing the configuration file.

SemDupl-shallow is a console tool and can be started using either the script `SemDupl-shallow.bat` for Windows or `SemDupl-shallow` for Unix-based operating systems. The following command line options are defined:

`-text` *<filename 1> [<filename 2>...]* Space-separated list of filenames which should be checked by SemDupl-shallow. If this option is given at the very beginning, the keyword `-text` can be omitted. The filenames must be specified either absolutely or relatively to the base directory of SemDupl-shallow. The use of wildcards is also possible

`-textdir` *<directory>* Instead of filenames, a directory can be specified using this option. All files are tested which are located inside this directory and end with the suffix `.txt` or `.jtxt`. The directory must be specified absolutely or relatively to the base directory of SemDupl-shallow.

`-textfile` *<filename>* A further alternative to specify the input texts. In this case, the given file contains a list of filenames which should be checked where each filename must be given in a separate line.

To specify the comparison texts, the following options are available:

`-x` The input texts are compared with each other. This means no external corpus is used.

`-corpus` *<filename 1> [<filename 2>...]* Space-separated list of files, which are used for comparison. The files again must be specified absolutely or relatively to the base directory of SemDupl-shallow.

`-corpusdir` *<directory name>* All files located in this directory and end with the suffix `.jtxt` are used for comparison. The directory must be specified absolutely or relatively to the base directory of SemDupl-shallow.

`-corpusfile` *<filename>* A further options is the specification of a file which contains a list of filenames, each on a separate line. The filename again must be specified absolutely or relatively to the base directory of SemDupl-shallow.

If no location of the input files is specified on the command line, the location which is given in the XML file `CErkenner.properties` is employed. The following optional options can be used (default parameter values are printed in bold):

`-error` *<filename> oder stderr* This option specifies the destination of the error stream.

`stderr` The error stream is written to the console (standard error)

`<filename>` The error stream is written to the given file

`-interpretation` *<Option>* This option controls the classification if a leaf of the decision tree is reached and there exist examples for both classes (duplicate / non-duplicate).

`standard` Standard decision procedure. The class is selected to which the majority of examples belong to.

`poi` *presumption of innocence* The class *no duplicate* is selected if there exists at least one example of this class.

`doubt` The class *duplicate* is selected if there exists at least one example of this class.

`trivalent` Tri-value, i.e., *duplicate*, *suspicion*, or *no duplicate*. The selected class is *duplicate*, if the leaf node contains exclusively examples of this class. *Suspicion* is selected if there are more examples of class *duplicate* than of type *no duplicate*. In all other cases the class *no duplicate* is selected.

`-loglevel <loglevel>` The option specifies the debug logging. Possible parameters are:

`off` no logging at all

`severe` only error output

`warning`

`config`

`fine`

`finer`

`finest`

`all` the most detailed output

`-output <filename>` or `stdout` This option controls the destination for textual output

`stdout` The output is written to the console.

`<filename>` The output is written to the file `<filename>`

`-proxy` In addition to the classification *duplicate*, *no duplicate* or *suspicion*, the arithmetic mean of the feature values is displayed. For this, all feature values have to be determined which causes a higher execution time.

`-stopwords <filename>` the file containing a list of stop words. (`data/stop.dat`).

`-synonyms <filename>` the file containing the employed synonyms. (`data/synonyms.dat`).

`-processors <number>` The number of processors used, usually `1`, `2`, `4` or `0` for all processors.

`-properties <filename>` Load the file `<filename>` instead of the default file `CErkenner.properties`.

`-tree <filename>` The decision tree of the system as is created by *RapidMiner* as an XML file with the extension `mod`. As default, the file `data/tree.mod` is used.

`-export <filename>` Export the features in XML format to the given file.

Examples for an applications The following guide is for the use of SemDupl-shallow from Linux. Task: *Examine the file* `google_000.jtxt` *against all files in the corpus and use two processors.*

```
SemDupl-shallow -text corpus/google_000.jtxt -corpusdir corpus
                -processors 2
```

Task: *Check all files, which can be matched by the pattern* `google_00?.jtxt`*, where the question mark (?) is a wildcard for exactly one arbitrary character, against all files in the corpus which can be matched by* `google_*.jtxt` *and use all available processors. The wildcard * matches to a sequence of arbitrary characters of any length.*

```
SemDupl-shallow -text corpus/google_00?.jtxt
                -corpus corpus/google_*.jtxt
```

Task: *Test all files in the directory* corpus *against each other, which start with the prefix* `rss`*.*

```
SemDupl-shallow corpus/rss* -x
```

B.3 SemDupl-quick

The default values are specified in the file `Flatchecker.properties` in XML format. This file contains the following:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
  <comment>Einstellungen fuer Flatchecker</comment>
  <entry key="aspellpath">/usr/bin/aspell</entry>
  <entry key="lang">de</entry>
  <entry key="encoding">UTF-8</entry>
  <entry key="dbpath">semdupl</entry>
  <entry key="output">stdout</entry>
  <entry key="wortschatzpasswd">anonymous</entry>
  <entry key="minsim">0.001</entry>
  <entry key="loglevel">warning</entry>
  <entry key="dbpasswd">semdupl</entry>
  <entry key="dbuser">semdupl</entry>
  <entry key="sourcepath">/cl/sd/utf</entry>
  <entry key="processors">1</entry>
  <entry key="error">stderr</entry>
  <entry key="wortschatzuser">anonymous</entry>
  <entry key="dbip">132.176.72.201</entry>
  <entry key="minfreq">17</entry>
  <entry key="maxfreq">MAX</entry>
</properties>
```

The SemDupl-quick program is called via the Bash script `SemDupl-quick` with the following options:

`-put` or `-p` The contents of the given files is inserted into the database if not already contained.

`-check` or `-c` Checks the given texts for duplicates

`-noput` The contents of the given files is not inserted into the database

`-text` *<filename 1> [<filename 2>...]* Space separated list of texts which should be checked by SemDupl-quick. If this options is specified as very first option then the keyword `-text` can be omitted. The text must be specified relatively to SemDupl-quick base directory or absolutely. The use of wildcards (e.g. * or *?*) is also possible.

`-textdir` *<directory>* All files inside this directory are included into the database. This directory can be specified absolutely or relatively of the SemDupl-quick base directory.

`-textfile` *<filename>* A text file is specified which contains a list of filenames, each on a separate line. All these files are included into the database.

`-R` If this option is specified, a given directory is searched for files recursively.

`-loglevel` *<loglevel>* This option specifies the logging level for debug and error output. The options for this output are:

`off` no logging output at all

`severe`

`warning`

`info`

`config`

`fine`

`finer`

`finest`

`all` most detailed output

`-aspellpath` *<Path>* to the executable of GNU Aspell.

`-lang` *<mnemonic>* mnemonic for the language for GNU Aspell (`de` for German)

`-encoding` *<Encoding>* The encoding used by GNU Aspell (`ISO-8859-1` for latin1 or `utf8` for UTF-8)

`-dbpath` *<database name>* Name of the database

`-dbip` *<IP>* IP address of the database server

`-dbpasswd` *<Password>* Password for the database server

`-wortschatzpasswd` *<Password>* Password for the database server for the Leipzig Wortschatz system.

`-sourcepath <path>` Optional parameter of the source path for the specified files. In this way, there is no need to state the path of each file separately.

`-processors <numbers>` The number of the processors, usually 1, 2, 4, or 0 for all processors.

`-error <filename>` or *stderr* Controls the error output of the program.

`stderr` Write all output to the console (stderr stream).

`<filename>` Errors are written into the file with name `<filename>`

`-maxfreq <Integer>` or *MAX*, frequence which is assumed if `wortschatz.uni-leipzig.de` returns no result for the query for the frequency class of a word. Use the keyword MAX for the greatest possible integer number (`Integer.MAX`).

B.4 Combiner

This section describes how duplicate features from the individual classifiers SemDupl-deep SemDupl-quick and SemDupl-shallow can be combined to a decision *duplicate* or *no duplicate*. To create such classification values for previously unseen data do the following:

- Run SemDupl-quick, SemDupl-deep and SemDupl-shallow on this data and copy the files, as created by these classifiers, to the locations expected by `svm.classify.sh` which are
  - `/home_pc/gesamt/semdupl/input_data/semdupl-deep.xml` for SemDupl-deep
  - `/home_pc/gesamt/semdupl/input_data/semdupl-quick.xml` for SemDupl-quick
  - `/home_pc/gesamt/semdupl/input_data/semdupl-shallow.xml` for SemDupl-shallow
- Switch to the directory `/home_pc/gesamt/semdupl/Combiner/scripts`
- Type: ./svm_classify.sh

B.5 Knowledge Acquisition

B.5.1 Hyponyms, Meronyms and Synonyms

The knowledge acquisition process of semantic and lexical relations is done in the following steps:

- Learning deep and shallow patterns
- Applying deep and shallow patterns to extract relations hypotheses
- Filtering hypotheses
- Determining features
- Assigning a quality score to the hypotheses

Learning deep and shallow patterns Shallow patterns can be learned using the relation extraction system developed by Duman (2008) which is not described in this paper.

In order to extract deep patterns, first a set of semantic networks containing a known hyponymy/meronymy or synonymy relation have to be collected.

Switch to the directory `trunk/src/pattern_extraction/tools` and enter

```
./example_searcher.out -d <dir> -m <int> -n <string> -r <rel>
    -s <start-index> -y <hyperrel>
```

where the parameters have the following meaning

-d directory where the SNs reside (omitting the directory *net*, key in configuration file: directory)

-m maximum number of files used for learning

-n negative-examples: file to store the negative examples where no meronym or hyponym shows up

-p output directory

-r relation-list file which contain a list of relations (hyponyms/meronyms)

-s start-index: numerical index where to start

-y hyperrelation: `sub0` or `mero` (`syno` not yet supported)

Afterwards the pattern can be learned. For that switch to the directory `src/pattern_extraction/deep` and type

```
./mdl_pattern_extractor.out -c <string> -d <dir> -m <num> -p <string>
    -o <string>
```

The parameters have the following meaning:

-c either mdl (minimum description length) or kernel. Recommendation: use the minimum description principle for learning

-d directory where the SNs reside

-m number of file use for learning

-o file where to store the patterns to

-p directory where the positive examples reside as determined by `example_searcher.out`

Application of Deep Patterns Deep patterns have to be defined in FOIL (First Order Inductive Learner) input format. An example line is:

```
sub0(A,B) :- sub(C,A), f_itms(D,C), pred(E,B), f_itms(D,E),
     h_itms(C,E), prop(E,ander#1#1),  refer(E,det) ;0.1;0.1;
     undandere_neighbor_sub
```

The period (.) has a special meaning in FOIL and is therefore replaced by a "#". The Conclusion is given by sub0(A,B), ":-" is the inference operator ($\leftarrow$). The premise

is given by a comma-separated list which is interpreted as logical conjunction. The predicate f is used for functions with variable number of arguments like *ITMS. It make it easier to define patterns where such function appear in. f_x(C0,C1) means that C1 appears as parameter for function x and the result is C0. The predicate g_x(C0,C1) is used to specify that C0 precedes C1 in the argument list of function x. The predicate h_x(C0,C1) specifies that argument C0 immediately precedes C1 in the argument list of function x.

The two numbers (here 0.1) are currently not used. The last argument is the name of the pattern. Patterns can be commented out by a preceding "//".

All extracted relation hypotheses are stored in a database. This database can be created by:

```
cd acquire/trunk/src/pattern_application/tools
./database_table_creator_mysql.out.out -b <database_name>
```

Afterwards the database can be filled with extracted hyponymy hypotheses by:

```
cd acquire/trunk/src/pattern_application/deep
./pattern_applier.out -d <directory> -p <patternfile> [-e ]
     -b <database_name> -u <user_name> -q <password>
```

where the parameters have the following meaning:

-d <directory>where the semantic network reside (key in configuration file: directory)

-b <database name>(key in configuration file: db_name)

-q <database password>(key in configuration file: db_password)

-u <database username>(key in configuration file: db_username)

-e exclude hyponyms with compounds (e.g. *Graubär* is a *Bär*)

-i <history filename>(key in configuration file: history_deep_hyponyms)

Similarly, meronyms meronyms can be extracted by:

```
cd acquire/trunk/src/pattern_application/deep
./meronym_pattern_applier.out -d <directory> -p <patternfile>
    -b <database_name> -u <user_name> -q <password>
```

The parameters are mostly identical to those of `./pattern_applier.out`.

Synonyms can be extracted by:

```
cd acquire/trunk/src/pattern_application/deep
./synonym_pattern_applier.out -d <directory> -p <patternfile>
     -b <database_name> -u <user_name> -q <password>
```

Application of Shallow Patterns Besides the usage of deep patterns there is also the possibility to apply shallow patterns. Shallow patterns are applied on the contents of the analysis-ml tag (token list) of WOCADI. The format of a pattern is: `(conclusion premise num1 num2 name)`.

The parameters num1 and num2 are currently not used. An example entry is:

```
(((syno a b)) (a ((word ‘‘respektive’’)) b) 0.5 0.5 ‘‘shallow_respektiv’’)
```

The variables *a* and *b* are matched with concepts of type object. The token entries (e.g. `((word "respektive"))` are tried to be unified with the token list provided

by the parser. The character "?" is used to specify an optional element, e.g.
`? (( cat (adj)))`.

The wildcard "*" is used to specify zero or more occurrences of one expression, like: `* ((( word ",")) b)`. The variables a and b are allowed to appear several times in a token list. In this case the Cartesian product of the bound constants is created as the set of extracted relations. d is used to specify a disjunction, e.g.
`(d ((((word ‘‘und’’))) (((word ‘‘oder’’))))))`.
The shallow hyponym pattern applier can be called in the following way:

```
cd trunk/src/pattern_application/shallow
./shallow_pattern_applier.out -d <directory> -p <patternfile>
      -b <database_name> -u <user_name> -q <password>
```

Shallow meronym patterns can be applied by:

```
cd trunk/src/pattern_application/shallow
./shallow_meronym_applier.out -d <directory> -p <patternfile>
      -b <database_name> -u <user_name> -q <password>
```

Synonym patterns can be applied by:

```
cd trunk/src/pattern_application/shallow
./shallow_synonym_applier.out -d <directory> -p <patternfile>
      -b <database_name> -u <user_name> -q <password>
```

Feature Calculation First the features have to be calculated. The dataset is divided into training and evaluation partitions:
Type: `cd acquire_repos_scheme/trunk/src/feature_combination/tools`
and

```
eval_training_set_divider.out -b <database_name> -u <user_name>
      -q <password> -f <factor>
```

where <factor>is the relative amount of training data (e.g. 0.7=7 parts training data, 3 parts evaluation data) Afterwards, calculate the features in the following way:
First type: `cd acquire_repos_scheme/trunk/src/feature_combination/`
Make sure that the history file is removed:
`rm feature_calculation.hist` Then start the feature calculation process:

```
./feature_calculator.out -b <directory> -p <patternfile>
      -b <database_name> -u <user_name> -q <password> -F
```

The current progress is always written to the file feature_calculation.hist. Thus, if the program crashes, the feature calculation process can be continued just by restarting the feature calculator as described above.

Score Calculation Finally, the score must be calculated. First, a certain subset of the entire annotated data is selected. To do this, switch to the directory `acquire_repos_scheme/trunk/src/feature_combination/tools`.
Type:

```
./combination_training_set_creator.out -e <num_examples_pattern>
      -m <num_examples>-r <hyper-relation> -i <inital_set> [-I]
      -o <output_filename> -d <directory> -p <patternfile>
```

```
    -b <database_name> -u <user_name> -q <password>
```
The parameters have the following meaning:

-e num_examples_pattern specifies the number of examples for hyponymy / meronymy / synonymy hypotheses per pattern

-m num_example total number of patterns

-r generalized relation name of the generalized relation, can be `sub0`, `mero`, `syno`.

-i initial_set this option can be used to enlarge an existing training set

-I This option is used to specify that the number of positive and negative examples need not to coincide

-o Output file suffix containing the selected training data. The files: <output_filename_<hyper-relation>.scm and <output_filename>_<hyper-relation>.svm will be created

Next step is to learn a model. For that the toolkit LIBSVM is employed. Switch to the directory app_data and type:

```
./svmtrain.sh <generalized_relation> <output_filename>
```
The created model can then be used to create the probability estimates which are provided by LIBSVM and serve as confidence score in SemQuire. To do this, switch to the directory: `src/feature_combination` and type:

```
./score_calculator -l <learning_method> -r <hyper-relation>
    -s <svm_model> -w <feature_list> -b <database_name>
    -u <user_name> -n <host_name> -q <password>
```
where the meaning of the parameters is defined as follows:

-l learning_method: Allowed values are

0 for a confidence score determined by a linear combination usually be determined by ordinary least square (OLS) regression (parameter values are contained in the feature list)

1 for a confidence score determined by a support vector machine classifier (as described above)

2 for a confidence score determined by a support vector machine employing a graph kernel function (computation-intensive)

- svm_model svm model file, only needed for learning method number 1 (svm)
- database_name, user_home, host_name, password: database parameters

There is also the possibility to do all steps (pattern application / training set selection / score computation) in one. To do this switch to the directory `src/feature_combination/tools` and type:

```
./extract_all.out -b <database_name> -u <user_name> -q <password>
    -n <hostname>
```
where

-b,u,q,n database parameters
-T directory containing the results
-r name of the extracted generalized relation (SUB0, MERO or SYNO)

Similarly, evaluation output can be generated automatically for a given training set where all steps are done automatically which includes:

- Automatic feature selection
- Evaluation results with ordinary features
- Evaluation results with string and graph kernel
- Evaluation results with additional grid search
- Signficance scores for graph / string kernel based evaluation in comparison with only feature-based validation

To do this, switch to the directory:
`src/feature_combination/tools` and type:

```
./learn_all.out -b <database_name> -u <user_name> -q <password>
    -n <hostname>
```

where
-b,u,q,n database parameters
-T directory containing the results
-r name of the extracted generalized relation (SUB0, MERO or SYNO)

The validation with ontological sorts and semantic features is done automatically, i.e., the sort/features of all hypotheses in the database are consistent. However, it may be the case that a collection of relations should be verified regarding sorts and features which were not extracted by our proposed pattern matching approach. For instance, we currently try to map Cyc relations to MultiNet. Furthermore, the lexicon is continuously extended which means that a sort/feature consistency check which can not be done in the moment could be possible in the future. Therefore, the possibility is included to check the hypotheses in the database for sort/feature consistency. The check is done as follows:
Switch to the directory `src/feature_combination/tools` and type:

```
./database_checker.out -b <database_name> -u <user_name>
    -q <password> -w <weka buffer output> -f <weka-features>
```

The parameters have the following meaning:
-b,u,q,n database parameters
-w weka buffer output which describes the network structure of the tree augmented naïve Bayes algorithm
-f feature file in arff format

The annotations can be exported by the tool `backup_annotations.out` and restored with the tool `restore_annotations.out`. The advantage of this tools is that annotations from one database can be used for another, e.g. the annotations for an older Wikipedia can be used for a newer one. The parameters of `backup_annotations.out` are defined as follows:
-b,u,q,n database parameters
-r name of the generalized relation name (either `mero`, `sub0`, or `pars`)
-o name of the backup file

The parameters of `restore_annotations.out` are given below:

-b,u,q,n database parameters
-r name of the generalized relation name (either `mero`, `sub0`, or `pars`)
-i name of the annotation file
The database entries can for example be exported by the tool `data_exporter.out` in the following way:
`./database_exporter -b <kb> -g mero.net -r mero` exports all meronym hypotheses of the database 'kb' which were annotated as correct to the file `mero.net`.
`./database_exporter -b <kb> -s mero.net -r mero -l 1000` exports the 1000 meronym hypotheses with the highest score to the file `mero.net`.

Logical Validation of the Knowledge Base The knowledge base can be validated employing an automated theorem prover. To do this, install the automated theorem prover Ekr-hyper (Baumgartner et al., 2007) and switch to the directory `src/logval` and type

```
./log_validator.out -s <sub0> -m <mero> -k <kb1,...,kbn>
      -b <database_name> -u <user_name> -n <host_name> -q <password>
```

where the parameters are defined as follows:
-b,u,q,n database parameters
-m name of the meronymy database
-s name of the hyponymy database
-k comma-separated list of validated knowledge bases
Hypotheses found as inconsistent are marked by the theorem prover in the database column LOGVAL.

B.5.2 Entailments

This tutorial describes the extraction of entailments employing a web search query. First step is to create the configuration file with the search tuples. An example file is given below:

```
(
(
(surface ("Mozart" "5. Dezember 1791"))
(deep (
  ((attr A B) (sub B "nachname.1.1") (val B "mozart.0"))
  ((attr A B) (sub B "jahr.1.1") (val B C) (card C 1791)
   (attr A D) (sub D "monat.1.1") (val D E) (card E 12)
   (attr A F) (sub F "tag.1.1") (val F G) (card G 5)))))
(
(surface ("Mozart" "27. Januar 1756"))
(deep (
  ((attr A B) (sub B "nachname.1.1") (val B "mozart.0"))
  ((attr A B) (sub B "jahr.1.1") (val B C) (card C 1756)
   (attr A D) (sub D "monat.1.1") (val D E) (card E 1)
   (attr A F) (sub F "tag.1.1") (val F G) (card G 27)))))
(
(surface ("Bundesbank" "Leitzins"))
(deep (
```

```
((sub A "bundesbank.1.1"))
((sub A "leitzins.1.1"))
)))

(
(surface ("Columbus" "Amerika"))
(deep (
       ((attr A B) (sub B "nachname.1.1") (val B "columbus.0"))
       ((attr A B) (val B "amerika.0"))))
)
(
(surface ("Edison" "Glühbirne"))
(deep (
       ((attr A B) (sub B "nachname.1.1") (val B "edison.0"))
       ((sub A "glühbirne.1.1")))
))
(
(surface ("Zuse" "Computer"))
(deep (((attr A B) (sub B "nachname.1.1") (val B "zuse.0"))
       ((sub A "computer.1.1")))
))
(
(surface ("Becker" "Wimbledon" "1985"))
(deep
(((attr A B) (sub B "nachname.1.1") (val B "becker.0"))
((attr A B) (sub B "name.1.1") (val B "wimbledon.0"))
((attr A B) (sub B "jahr.1.1") (val B C) (card C 1985))
)
))
(
(surface ("Deutschland" "Demokratie"))
(deep
(
((attr A B) (sub B "name.1.1") (val B "deutschland.0"))
((attr A B) (sub B "demokratie.1.1"))
))
)
(
(surface ("CDU" "Christlich Demokratische Union"))
(deep
(((attr A B) (sub B "name.1.1") (val B "cdu.0"))
((*modp B "christlich.1.1" "demokratisch.1.1" categ situa)
 (prop A B) (prob A "union.1.1"))
))
)
(
(surface ("Berlin" "Hauptstadt" "Deutschland"))
(deep
```

```
(((attr A B) (sub B "name.1.1") (val B "berlin.0"))
((sub A "hauptstadt.1.1"))
((attr A B) (sub B "name.1.1") (val B "deutschland.0"))
))
)
(
(surface ("Deutschland" "Polen" "1939"))
(deep
(((attr A B) (sub B "name.1.1") (val B "deutschland.0"))
((attr A B) (sub B "name.1.1") (val B "polen.0"))
((attr A B) (sub B "jahr.1.1") (val B C) (card C 1939)))
)
)
(
(surface ("DDR" "Warschauer Pakt" "1990"))
(deep
(((attr A B) (sub B "name.1.1") (val B "ddr.0"))
((attr A B) (sub B "name.1.1") (val B "warschauer_pakt.0"))
((attr A B) (sub B "jahr.1.1") (val B C) (card C 1990)))
)
)
(
(surface ("Mauer" "Berlin" "1961"))
(deep
(((sub A "mauer.1.1"))
((attr A B) (sub B "name.1.1") (val B "berlin.0"))
((attr A B) (sub B "jahr.1.1") (val B C) (card C 1961))
))
)
(
(surface ("Deutschland" "Argentinien" "1990"))
(deep
(((attr A B) (sub B "name.1.1") (val B "deutschland.0"))
((attr A B) (sub B "name.1.1") (val B "argentinien.0"))
((attr A B) (sub B "jahr.1.1") (val B C) (card C 1990))
))
)
(
(surface ("Deutschland" "Niederlande" "1974"))
(deep
(((attr A B) (sub B "name.1.1") (val B "deutschland.0"))
((attr A B) (sub B "name.1.1") (val B "niederlande.0"))
((attr A B) (sub B "jahr.1.1") (val B C) (card C 1990))
))
)
(
(surface ("Deutschland" "Italien" "1982"))
(deep
```

```
(((attr A B) (sub B "name.1.1") (val B "deutschland.0"))
((attr A B) (sub B "name.1.1") (val B "italien.0"))
((attr A B) (sub B "jahr.1.1") (val B C) (card C 1982))
))
)
(
(surface ("Deutschland" "Argentinien" "1986"))
(deep
(((attr A B) (sub B "name.1.1") (val B "deutschland.0"))
((attr A B) (sub B "name.1.1") (val B "argentinien.0"))
((attr A B) (sub B "jahr.1.1") (val B C) (card C 1986))
))
)
(
(surface ("Becker" "Stich" "1991"))
(deep
(((attr A B) (sub B "nachname.1.1") (val B "becker.0"))
((attr A B) (sub B "nachname.1.1") (val B "stich.0"))
((attr A B) (sub B "jahr.1.1") (val B C) (card C 1991))
)
))
(
(surface ("Becker" "Edberg" "1990"))
(deep
(((attr A B) (sub B "nachname.1.1") (val B "becker.0"))
((attr A B) (sub B "nachname.1.1") (val B "edberg.0"))
((attr A B) (sub B "jahr.1.1") (val B C) (card C 1990))
)
))
(
(surface ("Bayern München" "Manchester United" "1999"))
(deep
(
((attr A B) (sub B "name.1.1") (val B "bayern_münchen.0"))
((attr A B) (sub B "name.1.1") (val B "manchester_united.0"))
((attr A B) (sub B "jahr.1.1") (val B C) (card C 1999))
)
)
)
...
)
```

Each entry consists of a deep representation and an equivalent shallow one. Afterwards, web sites which contain the surface representation are searched with the `search_engine_learner` which is done as follows. Type

```
cd src/entailments
./search_engine_learner.out -d <entailments_dir>
```

This command creates a subdirectory for each search tuple and creates search engine

queries containing the surface texts. The found web sites in PDF or HTML format are stored in these directories. In the next step, the ASCII text is extracted from these web sites employing a self-written HTML to text converter and for PDF files the tool pdftotext. To do this type:

`./ascii_converter.out -d <entailments_dir>` Afterwards, the texts are segmented into single sentences where only the sentences are stored which contain all of the surface words. This is done with
`./sentence_segmenter.out -d <entailments_dir>`. This command creates files called `sentences.scm` in each subdirectory. In the next step these sentences need to be analyzed using WOCADI with
`./wocadi_applier.out -d <entailments_dir>`. This commands results in the creation of WOCADI .net-files for each input file in every subdirectory.

To replace the concepts in the semantic networks matching to an entry in the search tuples by the anchor variables, type:
`./variable_replacer.out -d <entailments_dir>`. Modified semantic networks are written in each subdirectories in files with the suffix *_var.net*.

Frequently occurring subnetworks (patterns) are learned from those modified semantic networks by typing `./search_engine_learner.out -d <entailments_dir>`
The patterns are stored in the file patterns.scm in each subdirectory.

In the last step the entailments are created by the Cartesian product of those patterns excluding the trivial entailments where a pattern entails itself. The entailments are then stored in the database. This process is executed by:

```
/se_entailments_storer.out -d <entailments_dir> -b <database_name>
      -u <user_name> -n <host_name> -q <password>
```

The parameters are the directory containing the entailments and search engine results and the database parameters where the entailments should be written to.

B.6 The GUI

The GUI (graphical user interface) provides a convenient access to the SemDupl prototype and its component detectors. It is written in Hop (`http://hop.inria.fr/`), a modern, portable, very concise Web 2.0 language, Hop is based on the functional programming language Scheme.

The screenshot in Figure 13 shows the main control elements. They are explained in the following.

SemDupl is a software system that finds duplicates for a given text (the candidate text) similar texts (semantic duplicates) in an existing text collection. Each text has a unique name (ID) of 4 letters or digits. In the standard text collection, these IDs come from the range Fz00 to Fzzz. The GUI shows a menu on the left hand side. All menu items are grouped in 3 groups, which are described in the following sections.

B.6.1 Text Collection Maintenance

There are two menu items for adding texts to the text collection: one for uploading a single text, one for uploading several texts.Furthermore, texts can be removed from the text collection.And finally, a keyword-based search allows to explore the

Figure 13: Screenshot of the GUI for SemDupl.

contents of the text collection. This is useful for human users in order to come up with ideas for possible duplicates.

B.6.2 Text Collection Analysis

After uploading a text, it must be analyzed so that the different duplicate detectors have access to the required input data.Two menu items are available for starting the analysis: the first one starts the analysis for the last uploaded text, the second one starts the analysis for all uploaded texts that have been analyzed before.

B.6.3 Text Duplicate Search

Finally, one can search for a text from the text collection all duplicates in the text collection. For this aim, four different methods are available: the deep one, the shallow one, the quick one, and the combined one.

B.7 Class Diagram SemDupl-shallow

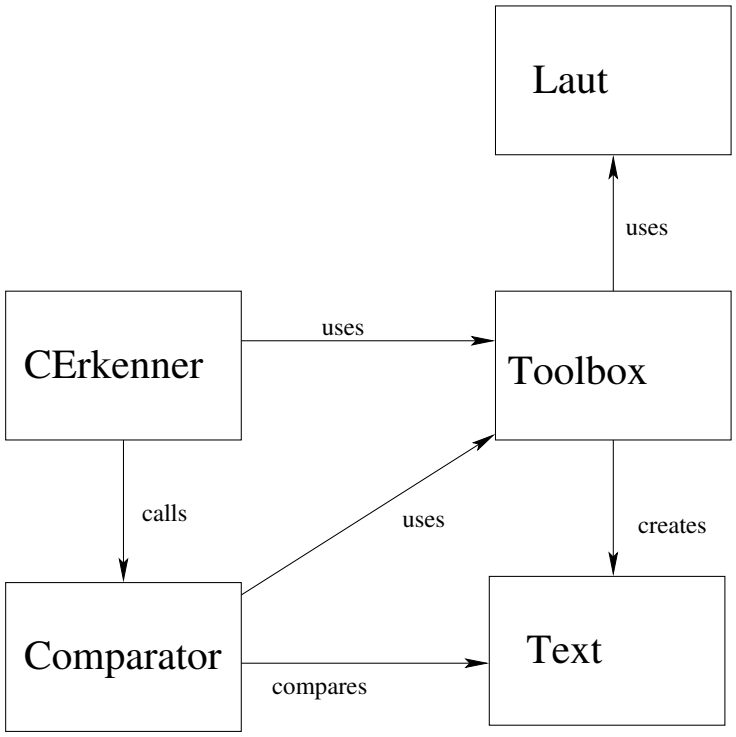

Figure 14: Class diagram of SemDupl-shallow.

SemDupl-shallow is implemented in Java and therefore follows an object oriented design. The class structure of SemDupl-shallow is given in Figure 14.

B.7.1 Class *Text*

The class *Text* is used for the internal representation of the texts which are to be checked. All texts are represented by an instance of this class.

Fields This class contains fields which are necessary for the comparison of two texts which are:

- The original text, the text after the removal of punctuation marks, the text after removal of stop words, and the text containing the word lemmas
- The entire list of words appearing in a text, the list of words after the removal of stop words, and after stemming
- The set of synonyms and the set of synonyms after stop word removal including the word itself
- The average word and sentence length of a text as a floating point number
- The average word and sentence lengths of each paragraph as list of floating point numbers
- The maximum deviation of the average word and sentence length in the paragraphs from the average
- The MD5 sum of a text as list of bytes
- 1-skip-2-grams over the entire text as list of strings

Methods *Text* contains mainly *getter*-methods, which means methods which return the values of the fields of the objects. In order to return the n-grams, the four methods `getGramm(int, int)`, `getGrammStop(int, int)`, `getGrammStemmed(int, int)` und `getGrammStemmedStop(int, int)` were implemented. These methods should be called with the skip width and the length of the queried n-grams. Is the queried n-gram contained in the list of the stored n-grams than this n-gram is returns. Otherwise it has to be calculated, stored in the text object and returned afterwards. Further methods are *load* and *save* for loading and saving the text with the extension *.jtxt* by employing the Java serialization.

The class text employs for the calculation of its stored information static methods of the class *Toolbox*.

B.7.2 Class *Toolbox*

The class *Toolbox* contains static methods which are required for the creation of *Text* objects. This class also contains the stop word and synonym list which only have to be kept in the memory once. It also makes use of the snowball-library which provides methods for stemming.

The class *Toolbox* contains only three class variables which are

- ***STEMMER*** An object instance of the class `snowball`
- ***STOP*** The stop word list
- ***SYNONYME*** A hash table which maps a word to its synonyms

The following methods are provided by this class

- ***calcMD5*** Determines the MD5 sum of a text

- ***getSynonyme*** Determines the synonyms of a word or word set.
- ***rechtschreibPruefung*** Checks the given text by means of the program *aspell*.
- ***satzzeichenEntferner*** Removes the punctuation marks in the given text.
- ***stem*** Does a stemming for the given word by means of the porter-stemmer-algorithm.
- ***stopwortEntferner*** Removes all stop-words from the given text.
- ***textStemmer*** Does a stemming for all words in a given text.
- ***woerterProSatz*** Determines the number of words per sentence of the given text.
- ***wortmenge*** Determines the word set of the given text.
- ***zeichenProWort*** Determines the average word length in the sentences of the given paragraph.

B.7.3 Class Comparer

This class contains the methods which are required for the comparison of two texts. There is one method for each feature implemented in SemDupl-shallow.

B.7.4 Class *LAUT*

LAUT is a global enumeration type and contains all possible onsets of the German language according to Brügge and Mohs (2003) with the value range {A, Ä, B, BL, BR, CH, D, DR, DSCH, E, F, FL, FR, G, GL, GN, GR, H, I, J, K, KL, KN, KR, KW, L, M, N, O, Ö, P, PFL, PFR, PL, PR, PS, R, S, SCH, SK, SKL, SL, SW, SZ, T, TR, TSCH, U, Ü, W, WR, X, Z, ZW}.

B.7.5 Class CErkenner

The class CErkenner combines the individual classes described so far. This class is the interface to the entire CErkenner duplicate recognition system and can be called from the command line.

This class consists several fields and methods for the interaction with the classes described above. Among others, it contains the main routine `main` which is used to start the SemDupl-shallow and specify the desired options.

Acknowledgment

This research was funded in part by the DFG project *Semantische Duplikatserkennung mithilfe von Textual Entailment* (HE 2847/11-1). Furthermore, we want to thank all members of our group who supported us in this work, especially Ingo Glöckner and Christoph Doppelbauer.

References

Balaguer, Enrique Vallés (2009). Putting ourselves in SME's shoes: Automatic detection of plagiarism by the WCopyFind tool. In *Proceedings of PAN Workshop and Competition*. Valencia, Spain.

Baumgartner, Peter; Ulrich Furbach; and Björn Pelzer (2007). Hyper tableaux with equality. In *Automated Deduction – CADE-21*, volume 4603 of *LNCS*, pp. 492–507. Heidelberg, Germany: Springer.

Brügge, Walburga and Katharina Mohs (2003). *Therapie der Sprachentwicklungsverzögerung*. Munich, Germany: Ernst Reinhardt, second edition.

Chang, Chih-Chung and Chih-Jen Lin (2001). *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Cimiano, Philip; Aleksander Pivk; Lars Schmidt-Thieme; and Steffen Staab (2005). Learning taxonomic relations from heterogeneous sources of evidence. In *Ontology Learning from text: Methods, evaluation and applications* (edited by Buitelaar, Paul; Philipp Cimiano; and Bernardo Magnini), pp. 59–73. Amsterdam, The Netherlands: IOS Press.

Cook, Diane J. and Lawrence B. Holder (1994). Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255.

Danninger, Christian (2009). *Werkzeug zur Validierung automatisch generierten Wissens*. Master's thesis, FernUniversit"at in Hagen, Hagen, Germany.

de Marneffe, Marie-Catherine and Christopher D. Manning (2008). *Stanford Typed Dependencies Manual*. Online at: `http://nlp.stanford.edu/software/dependencies_manual.pdf`.

Duman, Yilmaz (2008). *Automatische Extraktion von semantischen Relationen aus grošen Textkorpora*. Diplomarbeit, FernUniversität in Hagen, Hagen, Germany.

Eichhorn, Christian (2009). *Automatische Erkennung von Duplikaten in Textbeständen mithilfe flacher Verfahren*. Diplomarbeit, Technische Universität Dortmund, Dortmund, Germany.

EL-Manzalawy, Yasser and Vasant Honavar (2005). *WLSVM: Integrating LibSVM into Weka Environment*. Software available at `http://www.cs.iastate.edu/~yasser/wlsvm`.

Hamp, Birgit and Helmut Feldweg (1997). Germanet - a lexical-semantic net for german. In *Proceedings of the ACL workshop Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications*. Madrid, Spain.

Hartrumpf, Sven (2003). *Hybrid Disambiguation in Natural Language Analysis*. Osnabrück, Germany: Der Andere Verlag.

Hartrumpf, Sven; Hermann Helbig; and Rainer Osswald (2003). The semantically based computer lexicon HaGenLex – Structure and technological environment. *Traitement Automatique des Langues*, 44(2):81–105.

Hartrumpf, Sven; Tim vor der Brück; and Christian Eichhorn (2010a). Detecting duplicates with shallow and parser-based methods. In *Proceedings of the 6th IEEE International Conference on Natural Language Processing and Knowledge Engineering (NLPKE)*. Peking, China.

Hartrumpf, Sven; Tim vor der Brück; and Christian Eichhorn (2010b). Semantic duplicate identification with parsing and machine learning. In *Proceedings of the 13th International Conference on Text, Speech and Dialogue (TSD)*, volume 6231 of *Lecture Notes in Artificial Intelligence*, pp. 84–92. Brno, Czech Republic: Springer.

Hearst, Marti A. (1992). Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING 92)*, pp. 539–545. Nantes, France.

Helbig, Hermann (2006). *Knowledge Representation and the Semantics of Natural Language*. Heidelberg, Germany: Springer.

Lin, Dekang and Patrick Pantel (2001). Dirt - discovery of inference rules from text. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 323–328. San Francisco, California.

Mierswa, Ingo; Michael Wurst; Ralf Klinkenberg; Martin Scholz; and Timm Euler (2006). Yale: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining* (edited by Ungar, Lyle; Mark Craven; Dimitrios Gunopulos; and Tina Eliassi-Rad), pp. 935–940. Philadelphia, Pennsylvania: ACM.

Petersohn, Helge (2005). *Data Mining: Verfahren, Prozesse, Anwendungsarchitektur*. Munich, Germany: Oldenbourg.

Porter, Martin F. (1997). An algorithm for suffix stripping. In *Readings in Information Retrieval*, pp. 313–316. San Francisco, California: Morgan Kaufmann Publishers.

Quinlan, John R. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.

Ravichandran, Deepak and Eduard Hovy (2002). Learning surface text patterns for a question answering system. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 41–47. Philadelphia, Pennsylvania.

Rivest, Ronald L. (1992). The md5 message-digest algorithm. URL `http://tools.ietf.org/html/rfc1321`, online; 26. Februar 2009.

Szpektor, Idan; Ido Dagan; and Bonaventura Coppola (2004). Scaling web-based acquisition of entailment relations. In *Proceedings of the Conference on Empirical Methods on Natural Language Processing (EMNLP)*. Prague, Czech Republic.

Tim vor der Brück, Hermann Helbig (2010). Validating meronymy hypotheses with support vector machines and graph kernels. In *Proceedings of the 9th International Conference on Machine Learning and Applications (ICMLA)*, pp. 243–250. Washington, District of Columbia.

vor der Brück, Tim (2009). Hypernymy extraction based on shallow and deep patterns. In *From Form To Meaning: Processing Texts Automatically, Proc. of the Biennial GSCL Conference 2009* (edited by Chiarcos, Christian and Richard Eckart de Castilho), pp. 41–52. Potsdam, Germany.

vor der Brück, Tim (2010a). Hypernymy extraction using a semantic network representation. *International Journal of Computational Linguistics and Applications (IJCLA)*, 1(1).

vor der Brück, Tim (2010b). Learning deep semantic patterns for hypernymy extraction following the minimum description. In *Proceedings of the 29th International Conference on Lexis and Grammar (LGC)*, pp. 39–49. Belgrade, Serbia.

vor der Brück, Tim (2010c). Learning semantic network patterns for hypernymy extraction. In *Proceedings of the 6th Workshop on Ontologies and Lexical Resources (OntoLex)*, pp. 38–47. Peking, China.

vor der Brück, Tim (2011). Synonymy extraction using a semantic network representation. Submission planned.

vor der Brück, Tim and Hermann Helbig (2010). Retrieving meronyms from texts using an automated theorem prover. *Journal for Language Technology and Computational Linguistics*. Accepted.

vor der Brück, Tim and Holger Stenzhorn (2010). Logical ontology validation using an automatic theorem prover. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI)*, pp. 491–496. Lisbon, Portugal.

Weber-Wulff, Debora (2002). Der groše Online-Schwindel. *Spiegel Online*. URL `http://www.spiegel.de/unispiegel/studium/0,1518,221507,00.html`, online; 2008-10-15.

Weber-Wulff, Debora (2009). Softwaretest 2008. URL `http://plagiat.fhtw-berlin.de/software`, online at `http://plagiat.fhtw-berlin.de/software`.

Zauner, Michael (2009). *Lernverfahren für Inferenzregeln und Paraphrasierungen im Bereich deutscher Verben*. Diplomarbeit, FernUniversität in Hagen, Hagen, Germany.