Extracting Lexical Views from Software

Sattose

Jean-Rémy Falleri, Michel Dao, Marianne Huchard, Mathieu Lafourcade, Clémentine Nebut, Violaine Prince

LIRMM, CNRS, Université Montpellier 2 et France Télécom

mai 2009

Motivation

- Strong impact of identifiers for understanding programs
- Many reverse engineering tasks are based on names
  - detection of defects
  - program restructuring
  - aspect mining
- Linguistic resources not sufficient
  - wordnet does not contain part of software vocabulary (hashmap, thread)
  - terms are composed, contain abbreviations, etc.

Proposal: extract and organize main terms of the software

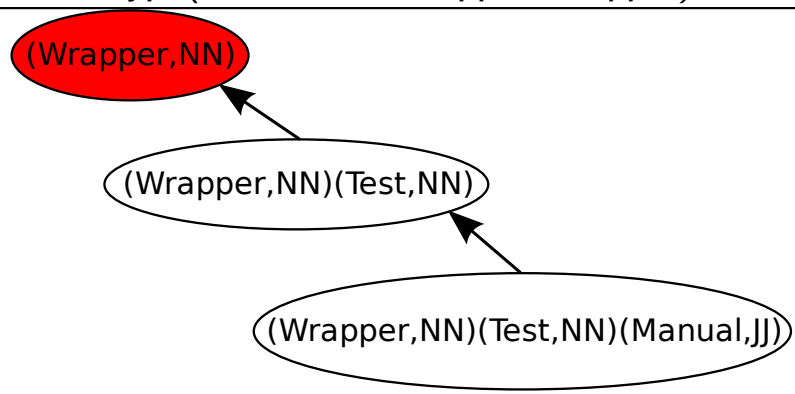
	Identifier 1	Identifier 2
Step	*TestWrapper*	*ManualTestWrapper*
Tokenization	*Test,Wrapper*	*Manual,Test,Wrapper*
POS tagging	*(Test,NN),(Wrapper,NN)*	*(Manual,JJ),(Test,NN),(Wrapper,NN)*
Dependency sorting	*(Wrapper,NN),(Test,NN)*	*(Wrapper,NN),(Test,NN),(Manual,JJ)*
Lexical enhancement	*(Wrapper,NN)*	
Lexical relations	*hypo(ManualTestWrapper,TestWrapper)* *hypo(TestWrapper,Wrapper)* *hypo(ManualTestWrapper,Wrapper)*	
Lexical view		

Table: Analysing two *wrapper* identifiers (from *Salome-TMF*)

Lexical relations

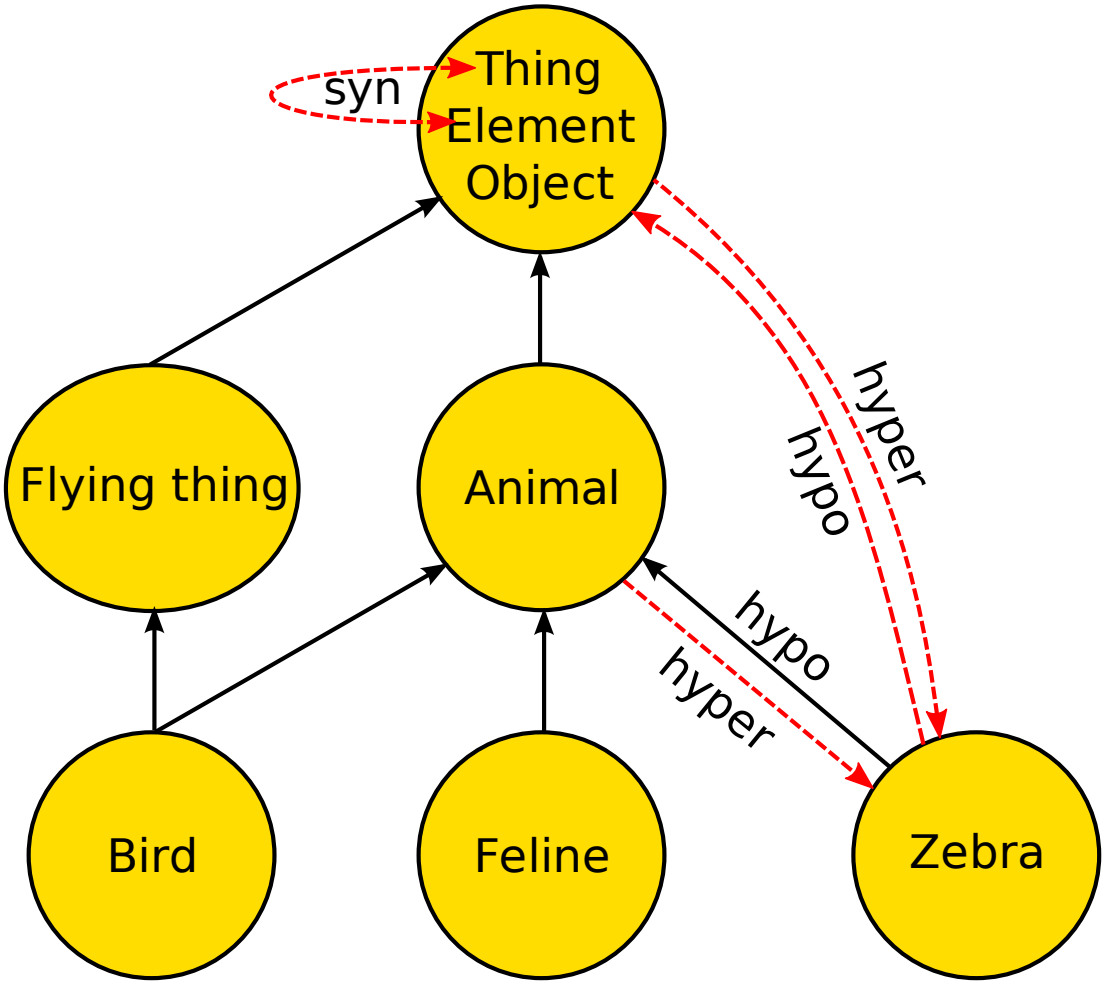


Figure: A sample lexical view

Tokenization (segmentation)

TestWrapper $\Rightarrow$ *Test*, *Wrapper*

Clues for cutting:

- Sequences of numeric characters (*block129*)
- Sequences of non alpha-numeric characters (*next_warning*)
- Case changes (as in *getNextWarning*)

Alternative strategies: use a dictionary (corruption risk)

POS tagging

Classifies the words into grammatical categories: noun, adjective, verb, etc.
Uses the Tree-tagger tool [Schmid 1994]

Test,Wrapper $\Rightarrow$ *(Test,NN),(Wrapper,NN)*

(Manual, Test, Wrapper,NN) $\Rightarrow$
(Manual,JJ),(Test,NN),(Wrapper,NN)

file, configuration $\Rightarrow$ *(file,NN), (configuration,NN)*
- unfortunately *(file,VV), (configuration,NN)*

Dependency analysis

(Test,NN),(Wrapper,NN) $\Rightarrow$ *(Wrapper,NN),(Test,NN)*

Sorts the words according to their importance for the meaning

Wrapper is the noun, Test is a precision about the noun

1. size(I) = 0 $\Rightarrow$ stop
2. size(I) = 1 $\Rightarrow$ insert the element of I at the end of N, and remove it from I.
3. size(I) = 2, the first element is a noun, while the second is not, $\Rightarrow$ the first element is added at the end of N and removed from I.
4. the first element of I is a verb $\Rightarrow$ it is added at the end of N and removed from I.
5. the first element of I is a preposition $\Rightarrow$ it is added at the end of N and removed from I.
6. I is the sequence (E=elements that are not prepositions, P=a preposition, R= the rest) $\Rightarrow$ apply rules to E and add the result to N, add P, apply rules to R and add the result to N
7. the last element of I is a number $\Rightarrow$ it is moved at the beginning of I.
8. (Default rule) the last element of I is added at the end of N and removed from I.

Dependency analysis

JavaBlock12

$I = (Java, NN), (Block, NN)(12, CD),\ N = \emptyset$

1. Rule 7 (last element is a number), move $(12, CD)$ at the beginning of I

   $I = (12, CD), (Java, NN), (Block, NN),\ N = \emptyset$

2. Rule 8 (default), transfer $(Block, NN)$ at the end of N.

   $I = (12, CD)(Java, NN),\ N = (Block, NN)$

3. Rule 8 (default), transfer $(Java, NN)$ at the end of N.

   $I = (12, CD),\ N = (Block, NN)(Java, NN)$

4. Rule 2 (size(I)=1), transfer $(12, CD)$ at the end of N

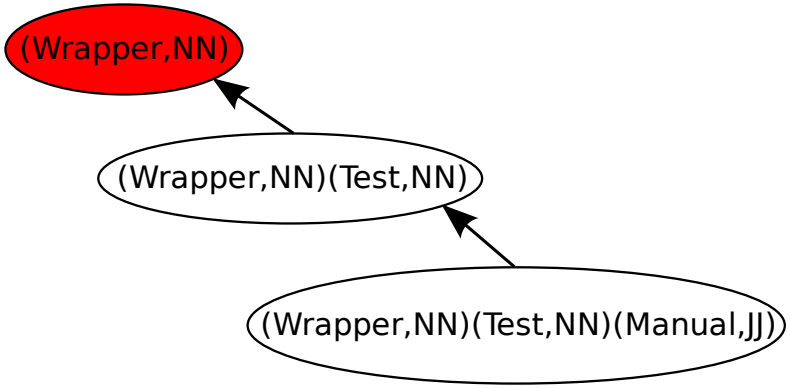
   $N = (Block, NN)(Java, NN)(12, CD)$

Lexical enhancement

With *TestWrapper* and *ManualTestWrapper*
Recognizing implicit concept *(Wrapper,NN)*
They are the common prefixes in output of previous step

Lexical view

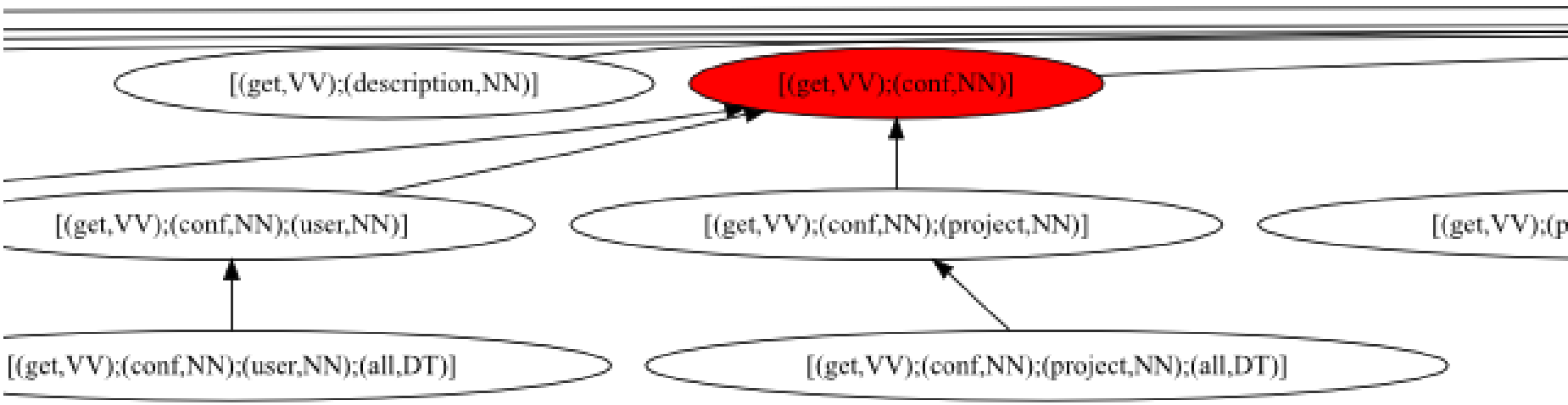
Recognizing lexical relations *(Wrapper,NN)*



Some synonym terms can be given in advance

- same length, same prefix = synonyms
- different sizes, common prefix covering one identifier = the longuest is the hyponym, the smallest is the hyperonym
- otherwise different sizes, common prefix = terms are co-hyponyms

Lexical view



Lexical view

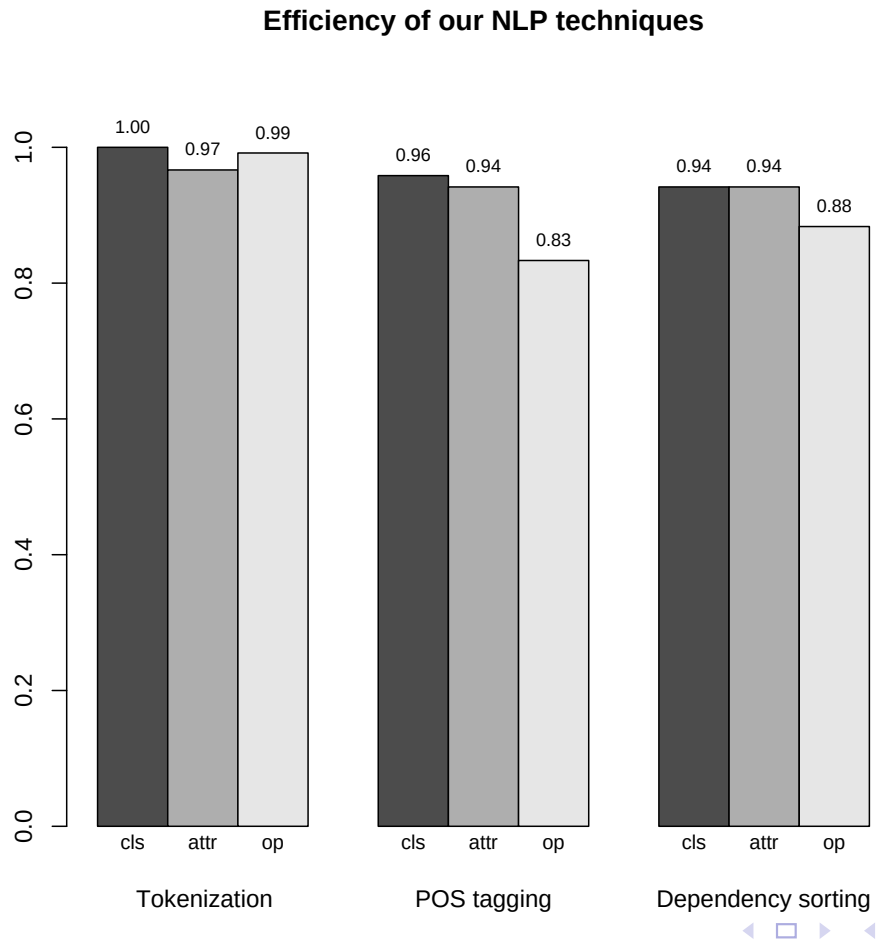
Validation

Extraction at random of 5 attributes, classes and operations identifiers from 24 Java open source softwares = 360 identifiers
The output of our technique is compared to manual operation results.

- ratio of identifiers that have been successfully tokenized,
- ratio of identifiers that have been affected correct parts-of-speech,
- ratio of identifiers for which the dependency sorting has been applied correctly.

Validation

- p^k_{tok} successfully tokenized
- p^k_{tag} correct parts-of-speech
- p^k_{dsort} correct dependency analysis



Ongoing work

- using the lexical relations in class hierarchy restructuring
- analysing the built lexical views
- finding users for the technique...