UDC 004[822+823]

# SEMANTIC FRAMES: INFERENCE ENGINE FOR KNOWLEDGE EXTRACTION

*ALIAKSANDR DUNCHANKA, DZMITRY PIATKIN*
Polotsk State University, Belarus

*This article describes a process of designing an inference engine for extraction of knowledge from preprocessed sentences in English, which uses semantic frames as a mean for representation of both the production rules and the acquired information.*

In order to engineer an *inference engine* (IE) [1] for knowledge extraction two conditions must be met. First – the sentences being passed to the engine must be already lexically, grammatically and syntactically analyzed. Second – the result of this preprocessing must be represented in a unified way. The best approach would be to pass it as a group of interlinked objects of certain types which provide methods for requesting needed information.

We will start off by defining two primitive classes (Fig. 1): *<base-node>* encapsulates a type of a node, *<text-node>* contains a string representing an individual word.

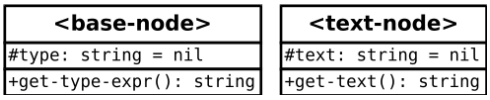


Fig. 1. Top-level classes

Now we proceed to define two intermediate classes (Fig. 2). *<node>* is a generic class that describes a node's syntactic role and holds a reference to its parent node. *<compound-node>* is intended to represent constructs, consisting of several nested simple or compound nodes.

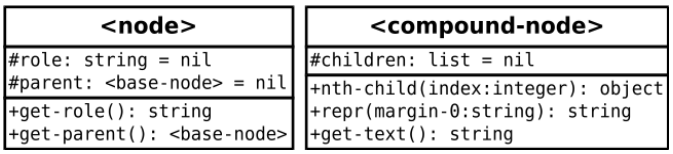


Fig. 2. Middle-level classes

Note that *<compound-node>* is not directly derived from *<text-node>*, but still responds to *get-text*. That is, it returns a string obtained by merging results of calling *get-text* on its children.

Having established a set of basic types, we can then design the actual classes necessary to represent the result of sentence analysis (Fig. 3–4).

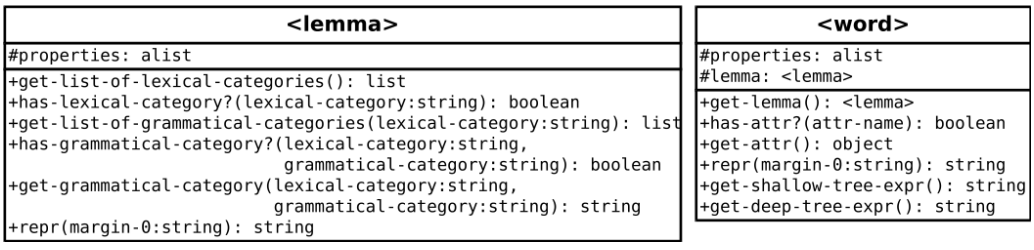


Fig. 3. Classes for representation of simple nodes

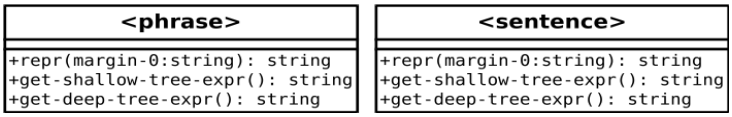


Fig. 4. Classes for representation of complex nodes

Relations between all previously mentioned classes are shown in Fig. 5.

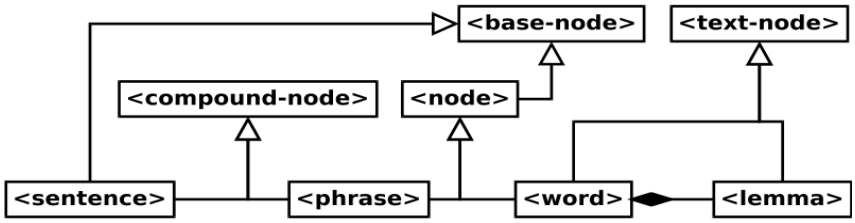


Fig. 5. Generic class diagram

Since we will implement our IE in *Common Lisp* [2], there is no need to introduce any additional abstract classes (interfaces) as a containers for methods *repr*, *get-shallow-tree-expr* and *get-deep-tree-expr*, because we have a powerful mechanism of *generic functions* [3] at our disposal to deal with this.

Every invocation of IE must supply two required parameters: one is an instance of *<sentence>* class and another is a pool of *semantic frames* (SF) [4][5] which provides the essential knowledge – *production rules* [6]. These rules are generated on-the-fly from a *S-expression* [7] based representation. The general layout of both extraction and subsumption rules, defined using *EBNF* [8], is the following:

*rule list* = *pool spec;*
*pool spec* = *'(pool', pool name, shared data, {frame spec}, ')';*
*pool name* = *node name;*
*shared data* = *'(', {binding}, ')';*
*binding* = *'(', data id, data, [data type], ')';*
*data id* = *SYMBOL | KEYWORD;*
*frame spec* = *'(frame', [frame reference], frame name, frame classifiers, {slot spec}, ')';*
*frame reference* = *node reference;*
*frame name* = *node name;*
*frame classifiers* = *node classifiers;*
*slot spec* = *regular slot spec | shared slot ref;*
*regular slot spec* = *'(slot', [slot reference], slot name, slot classifiers, slot data spec, ')';*
*shared slot ref* = *'(slot-ref', slot reference, ')';*
*slot reference* = *node reference;*
*slot name* = *node name;*
*slot classifiers* = *node classifiers;*
*slot data spec* = *data spec | data ref spec | reference spec | frame spec;*
*data spec* = *'data', data, [data type];*
*data* = *VALUE;*
*data type* = *KEYWORD;*
*data ref spec* = *'from', data id;*
*reference spec* = *'node', reference;*
*reference* = *KEYWORD;*
*node reference* = *KEYWORD;*
*node name* = *STRING | NIL;*
*node classifiers* = *'(', {classifier}, ')';*
*classifier* = *SYMBOL.*

An extraction rule is a *RELATION*-frame consisting of one or many *CONDITION*- and *ACTIVITY*-slots, as well as an arbitrary number of sub-relation slots (without classifiers). The format of the corresponding symbolic expression obeys the following syntax:

*rule frame* = *frame head, '(relation)', {relation attribute}, {node condition slot}, {node activity slot}, {subnode relation slot}, tail;*
*frame head* = *'(frame', name;*

*slot head* = *'(slot', name;*
*name* = *STRING | NIL;*
*tail* = *')';*
*relation attribute* = *slot head, '(attribute) data', VALUE, tail;*
*node condition slot* = *slot head, '(in condition', [condition negation], tail, condition slot data | node condition frame, tail;*
*node condition frame* = *frame head, '(condition', [condition negation], condition modifier, tail, {condition slot}, tail;*
*condition quantifier* = *'necessary' | 'possibly';*
*condition negation* = *'not';*
*condition modifier* = *'and' | 'or' | 'xor' | 'imp' | 'eqv';*
*condition slot* = *slot head, '(condition', [condition quantifier], [condition negation], tail, condition slot data | node condition frame, tail;*
*node activity slot* = *make instruction slot | {name instruction slot} | [data instruction slot] | {classifiers instruction slot} | {clarifiers instruction slot} | {export instruction slot} | {import instruction slot} | compound instruction slot;*
*export instruction slot* = *'(slot "export" (out activity) data (', entity selection, {target context level}, ')', tail;*
*import instruction slot* = *'(slot "import" (out activity) data (', entity selection, {source context level}, ')', tail;*
*entity selection* = *':frame' | ':slot';*
*target context level* = *context level;*
*source context level* = *context level;*
*context level* = *'top' | ':top' | '0' | POSITIVE INTEGER;*
*make instruction slot* = *'(slot "make" (out activity) data ', make instruction | ('(', {make instruction}, ')'), tail;*
*make instruction* = *':frame' | ':pslot/frame' | ':cslot/frame' | ':pframe/slot' | ':cframe/slot';*
*name instruction slot* = *'(slot "name" (out activity) data (', target context level, entity selection, text source, [text filter], '))';*
*text source* = *':form-text' | ':lemma-text';*
*text filter* = *f-expression;*
*f-expression* = *STRING;*
*data instruction slot* = *'(slot "data" (out activity) data (', target context level, text source, [text filter], '))';*
*classifiers instruction slot* = *'(slot "classifiers" (out activity) data (', target context level, entity selection, entity classifiers, '))';*
*entity classifiers* = *{SYMBOL};*
*clarifiers instruction slot* = *'(slot "clarifiers" (out activity) data (', target context level, entity selection, entity clarifiers, '))';*
*entity clarifiers* = *{f-expression};*
*compound instruction slot* = *'(slot nil (out activity) (frame nil nil ', {node activity slot}, '))'.*

The inference process has a top-down depth-first direction, i.e. it starts with top-level nodes and for each of them: checks the node itself, recursively checks its children and only then selects the next one on the same level. IE iterates over all rules (in order they are defined) and if some relation holds for the current node (i.e. conjunction of evaluated conditions equals TRUE), then IE executes its instructions sequentially. For a compound node IE will also try to apply sub-relations (if they are present) to its children. When any particular relation holds and it has an attribute «*once-only*» set to TRUE, IE would stop the iteration and would not attempt to apply remaining rules to the current node.

Conditions are a fundamental part of decision-making strategies. They allow to pick important pieces of a sentence and reject other insignificant or redundant information. Every condition name is associated with a corresponding function that retrieves a specific node's attribute. Evaluation of a condition involves invocation of such a function with the current node as an argument and comparing the result with a regular expression, stored in the slot. Table 1 provides short descriptions for all possible conditions.

Table 1. – Conditions

| Name | Description |
|---|---|
| type | Node's type expression |
| stree | Node's shallow tree expression |
| dtree | Node's deep tree expression |
| role | Node's role expression |
| text | Node's text |
| has-misc-attrs | List of node's miscellaneous attributes |
| has-attrs | One or list of node's grammatical categories |
| attrs | List of node's grammatical categories and their values |
| lemma-text | Lemma's text (words only) |
| lemma-has-type | One or list of lemma's lexical categories (words only) |
| lemma-has-attrs | Lists of lemma's lexical plus grammatical categories (words only) |
| lemma-type | Lemma's lexical category (words only) |
| lemma-attrs | List of node's grammatical categories and their values for a given lexical category (words only) |
| dir-parent-type | Type of a direct parent node |
| top-parent-type | Type of a top-level parent node |
| dir-parent-stree | Shallow tree of a direct parent node |
| top-parent-stree | Shallow tree of a top-level parent node |
| dir-parent-dtree | Deep tree of a direct parent node |
| top-parent-dtree | Deep tree of a top-level parent node |
| dir-parent-role | Role of a direct parent node |
| top-parent-role | Role of a top-level parent node |
| dir-parent-text | Text of a direct parent node |
| top-parent-text | Text of a top-level parent node |
| sentence-type | Sentence type |
| sentence-text | Sentence text |

For every sentence node being processed IE creates a separate context with two unset references – for a frame and for a slot; and two links – to a parent context and to the node itself. Because all contexts are connected all the way to the top, it is possible to define relations between frames using «*make*» instruction (and cross-references using «*import*»/«*export*» commands). The behavior of this instruction can be controlled by specifying one or several of symbolic constants described in Table 2.

Table 2. – Arguments to «*make*» instruction

| Literal | Description |
|---|---|
| frame | Create a new frame and assign it to the reference in the current context |
| pslot/frame | Create a new frame and assign it both to the reference in the current context and as data of a slot in the parent context |
| cslot/frame | Create a new frame and assign it both to the reference and as data of a slot in the current context |
| pframe/slot | Create a new slot of a frame in the parent context and assign it to the reference in the current context |
| cframe/slot | Create a new slot of a frame in the current context and assign it to the reference in the current context |

After the extraction of knowledge is completed, the pool of newly-created frames is passed further to the subsumption procedure which performs normalization of relations and updates the existing knowledge base.

REFERENCES

1. Inference engine [Electronic resource] / Wikipedia – The Free Encyclopedia. – Mode of access: https://en.wikipedia.org/wiki/Inference_engine. – Date of access: 10.01.2018.
2. Common Lisp [Electronic resource] / Wikipedia – The Free Encyclopedia. – Mode of access: https://en.wikipedia.org/wiki/Common_Lisp. – Date of access: 11.01.2018.
3. Generic function [Electronic resource] / Wikipedia – The Free Encyclopedia. – Mode of access: https://en.wikipedia.org/wiki/Generic_function. – Date of access: 12.01.2018.
4. Semantic network [Electronic resource] / Wikipedia – The Free Encyclopedia. – Mode of access: https://en.wikipedia.org/wiki/Semantic_network. – Date of access: 13.01.2018.
5. Frame (artificial intelligence) [Electronic resource] / Wikipedia – The Free Encyclopedia. – Mode of access: https://en.wikipedia.org/wiki/Frame_(artificial_intelligence). – Date of access: 14.01.2018.
6. Rule-based system [Electronic resource] / Wikipedia – The Free Encyclopedia. – Mode of access: https://en.wikipedia.org/wiki/Rule-based_system. – Date of access: 15.01.2018.
7. S-expression [Electronic resource] / Wikipedia – The Free Encyclopedia. – Mode of access: https://en.wikipedia.org/wiki/S-expression. – Date of access: 16.01.2018.
8. Extended Backus–Naur form [Electronic resource] / Wikipedia – The Free Encyclopedia. – Mode of access: https://en.wikipedia.org/wiki/Extended_Backus–Naur_form. – Date of access: 17.01.2018.