NLTK semparse package

1 Files

This section lists and the files in the `nltk.semparse` package and summarizes their contents and functions.

1. `data/`: config, ccg lexicon, and test files. Of particular interest are `data/lib/{english.txt, english_pos.txt}`, which define language-specific information.
2. `build_ccglex`: script for converting output of the C&C supertagger into a CCG lexicon that meets the formatting requirements of `nltk.ccg.lexicon`.
3. `composer.py`: defines `SemanticComposer`, the class for building logical expressions from syntactic parse trees.
4. `config.py`: methods for accessing configuration files.
5. `extract_template_grammar.py`: script for extracting generalized CCG category combination rules from a list of example combinations. See `data/grammars/`.
6. `get_supertags`: bash script for running the C&C supertagger.
7. `parseconverter.py`: defines `CCGParseConverter`, which converts AUTO string representations of CCG syntactic parses into nltk.Trees.

8. `rules.py`: functions for defining the word-specific semantic expressions for different types of lexical items.

9. `semanticcategory.py`: defines `SemanticCategory`, which governs the construction of logical expressions for lexical items.

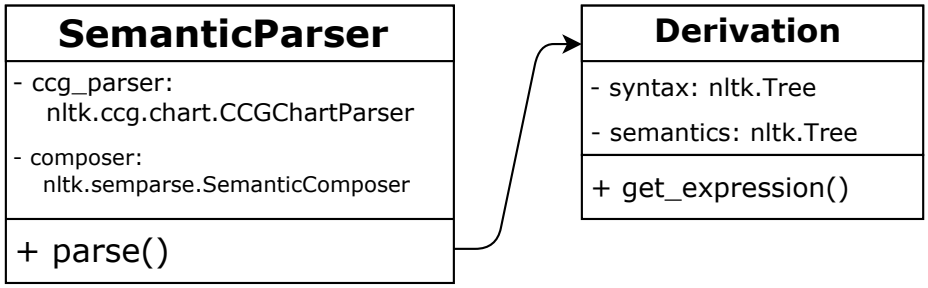
10. `semanticparser.py`: defines `SemanticParser`, the root class for the semantic parser. Governs the syntactic and semantic parsing and formatting of the input sentence.

11. `semparser`: interactive semantic parser.

12. `syntacticcategory`: defines `SyntacticCategory`, the class for representing CCG syntactic categories.

13. `test.py`: unit tests for `nltk.semparse`.

2 Code Overview



The `nltk.semparse.SemanticParser` class encapsulates two classes: `nltk.ccg.chart.CCGChartParser` and `nltk.semparse.SemanticComposer`. `CCGChartParser` syntactically parses the input sentence and passes this parse to `SemanticComposer` which builds a semantic expression based on the syntax. This entire process is carried out by the `parse()` method.

`parse()` returns a `Derivation` object, which holds the syntactic and semantic parses for the input sentence (as instances of `nltk.Tree`. The `get_expression()` method returns the

semantic expression for the input sentence, i.e. the root of the semantic derivation tree.

These classes are defined in the file `nltk.semparse.semanticparser.py`.

`nltk.semparse.SemanticComposer` has one main method, `build_expressions()` which traverses the CCG syntactic parse and builds a semantic derivation as an nltk.Tree. The leaves of the tree are logical expressions corresponding to individual lexical items in the input sentence and the root is the logical expression for the entire input sentence. Each node of the semantic derivation tree is an instance of `nltk.sem.logic.LogicalExpression`. `SemanticComposer` also has methods which carry out the composition, substitution, and type-raising operations on logical expressions. The `SemanticComposer` class is defined in the file `nltk.semparse.composer.py`.

When `SemanticComposer` reaches the leaves of the syntactic parse tree the logical expressions for the lexical items are built by the function `get_semantic_categories`. This finds all possible logical expressions for a given lexical item. Each possible logical expression is a `nltk.semparse.SemanticCategory` instance. A `SemanticCategory` is built in the following manner:

1. A `SyntacticCategory` instance is built from a string representation of the CCG syntactic category for the lexical item.
2. The `SyntacticCategory` instance is then passed to `SemanticCategory` along with the lexical item and its POS tag.
3. The semantic type of the lexical item is determined using `data/lib/english_pos.txt`.
4. `SemanticCategory.generate_expression()` determines if the lexical item is a special case using the rules defined in `data/lib/english.txt`.
    - If it is a special case the rule defined by `english.txt` specifies what the logical expression will be.

- If it is not a special case, first a stem logical expression is built using `SemanticCategory._get_stem` from the structure of the CCG syntactic category. Then the stem is augmented with word-specific information using the functions defined in `rules.py`.

5. The result of step 4 is the logical expression for the lexical item and is assigned to `SemanticCategory._expression`. This is accessed via `SemanticCategory.get_expression()`.