
elit Documentation

Gary Lai

Feb 15, 2019

1	Installation	3
2	Decode with Web API	5
3	Decode with Python API	9
4	Deploy to ELIT	11
5	Available Models	13
6	Output Format	17
7	Decode with CLI	19
8	Train with CLI	21
9	Structures	23
10	Tokenizers	31
	Python Module Index	33

The ELIT project brings the latest natural language processing (NLP) technology to the community. The primary focus of this project is to present an end-to-end NLP framework that is scalable for big data analysis using rich resources in the cloud. ELIT facilitates the process of developing, accessing, and deploying cutting-edge NLP tools by allowing researchers to quickly prototype innovative ideas using plug-and-play modules, supporting web-APIs to access built-in models, and providing a cloud platform to deploy comprehensive systems. ELIT is developed by the [NLP Research Group](#) at Emory University with an active collaboration with the AWS [MXNet](#) team.

CHAPTER 1

Installation

Note that if

Environmental Setup

To be filled.

Python

ELIT requires a Python version ≥ 3.5 . You can check which version of Python 3.x installed on your machine

```
` sudo add-apt-repository ppa:jonathonf/python-3.6 sudo apt-get update sudo  
apt-get install python3.6 python3.6-dev python3-setuptools `
```

Virtual Environment

The elit project only supports Python with version ≥ 3.5 . Please make sure you have installed correct version of the Python on the machine.

It is as simple as one line to install elit by:

```
` pip install elit `
```

Decode with Web API

ELIT provides web API that allows anyone to decode raw text into NLP structures using its [built-in models](#). The web API does not require the [ELIT installation](#) and can be used by any programming language that supports HTTP request/response.

2.1 Single Document

The following codes take an input document and run the NLP pipeline for tokenization, named entity recognition, part-of-speech tagging, morphological analysis, dependency parsing, and coreference resolution using the default parameters:

Python

```
import requests

url = 'https://elit.cloud/api/public/decode/doc'

doc = 'Jinho Choi is a professor at Emory University in Atlanta, GA. ' \
      'Dr. Choi started the Emory NLP Research Group in 2014. ' \
      'He is the founder of the ELIT project.'

models = [
    {'model': 'elit_tok_lexrule_en'},
    {'model': 'elit_ner_bilstm_en_ontonotes'},
    {'model': 'elit_pos_bilstm_en_mixed'},
    {'model': 'elit_morph_lexrule_en'},
    {'model': 'uw_coref_e2e_en_ontonotes'}]

request = {'input': doc, 'models': models}
r = requests.post(url, json=request)
print(r.text)
```

Java

```

import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.ContentType;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.util.EntityUtils;

public class ELITWebAPI
{
    public static void main(String[] args) throws Exception
    {
        HttpPost post = new HttpPost("https://elit.cloud/api/public/decode/doc");
        HttpClient client = HttpClientBuilder.create().build();

        String doc = "Jinho Choi is a professor at Emory University in Atlanta, GA. "
↪+
                "Dr. Choi started the Emory NLP Research Group in 2014. " +
                "He is the founder of the ELIT project.";

        String models = "[" +
                "{\"model\": \"elit-tok-lexrule-en\"}," +
                "{\"model\": \"elit-ner-flair-en-ontonotes\"}," +
                "{\"model\": \"elit-pos-flair-en-mixed\"}," +
                "{\"model\": \"elit-morph-lexrule-en\"}," +
                "{\"model\": \"elit-dep-biaffine-en-mixed\"}," +
                "{\"model\": \"uw-coref-e2e-en-ontonotes\"}"];

        String request = "{\"input\": " + doc + ", \"models\": " + models + "}";

        post.setEntity(new StringEntity(request, ContentType.create("application/json
↪"))));
        HttpResponse response = client.execute(post);
        System.out.println(EntityUtils.toString(response.getEntity()));
    }
}

```

Add the following dependency to pom.xml.

```

<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpclient</artifactId>
  <version>4.5.6</version>
</dependency>

```

Ruby

```

text = 'Jinho Choi is a professor at Emory University in Atlanta, GA. ' \
      'Dr. Choi started the Emory NLP Research Group in 2014. ' \
      'He is the founder of the ELIT project.'

```

Node.js

```

text = 'Jinho Choi is a professor at Emory University in Atlanta, GA. ' +
      'Dr. Choi started the Emory NLP Research Group in 2014. ' +
      'He is the founder of the ELIT project.'

```

The following shows the output in the JSON format (see the [output format](#) for more details):

```

{"sens":
  [{"sen_id": 0,
    "tok": ["Jinho", "Choi", "is", "a", "professor", "at", "Emory", "University", ".
↪"],
    "ner": [[0, 2, "PERSON"], [6, 8, "ORG"]],
    "pos": ["NNP", "NNP", "VBZ", "DT", "NN", "IN", "NNP", "NNP", "."],
    "mor": [{"("jinho", "NN")}, [{"("choi", "NN")}, [{"("be", "VB"), ("", "I_3PS")},
      [{"("a", "DT")}, [{"("pro+", "P"), ("fess", "VB"), ("+or", "N_ER")},
        [{"("at", "IN")}, [{"("emory", 'NN')}, [{"("university", "NN")}, [{"(".", "PU
↪")}]},
    "dep": [[1, "compound"], [2, "nsbj"], [4, "cop"], [4, "det"], [-1, "root"],
      [7, "case"], [7, "compound"], [4, "ppmod"], [4, "punct"]]},
    {"sen_id": 1,
      "tok": ["He", "is", "the", "director", "of", "EmoryNLP", "in", "Atlanta", ",", "GA", "."], ↵
↪...},
    {"sen_id": 2,
      "tok": ["Dr.", "Choi", "is", "happy", "to", "be", "at", "AWS", "re:Invent", "2018", "."], .
↪...}],
    "coref": [{"[0, 0, 2], [1, 0, 1], [2, 0, 2]}]}

```

See the [available models](#) for the list of all built-in models and their parameter settings.

2.2 Multiple Documents

To be filled.

CHAPTER 3

Decode with Python API

To be filled.

CHAPTER 4

Deploy to ELIT

Please contact us: gary.lai@emory.edu

Available Models

All models take the following naming convention:

```
[team]-[task]-[method]-[language]-[training_data]*
```

- `team`: the team who developed this model.
- `task`: the task that this model is developed for.
- `method`: the method used to develop this model.
- `language`: the input language (ISO 639-1; use `un` for universal models).
- `training_data`: the training data used to build this model (if applicable).

5.1 English Models

- `PRE`: tools that need to be run prior to the model.
- `DATA`: the dataset used to build the model.
- `EVAL`: evaluation on the dataset that the model is trained on.
- `BM`: the standard benchmark evaluation for the task.
- `tokseg`: any *tokenizer* with sentence segmentation.
- `inmixed`: any dataset in the `mixed` corpus.

5.1.1 Tokenization

5.1.2 Morphological Analysis

5.1.3 Part-of-Speech Tagging

- EVAL: accuracy.
- BM: accuracy on the Wall Street Journal portion of the [Penn Treebank](#) using the standard split (trn: 0-18; dev: 19-21; tst: 22-24).

5.1.4 Named Entity Recognition

- EVAL: F1-score.
- BM: F1-score on the English dataset distributed by the [CoNLL'03 shared task](#).

5.1.5 Dependency Parsing

- EVAL: UAS (unlabeled attachment score) / LAS (labeled attachment score).
- BM: UAS/LAS on the Wall Street Journal portion of the [Penn Treebank](#) using the standard split (trn: 2-21; dev: 22, 24; tst: 23) and the [Stanford typed dependencies](#).

5.1.6 Coreference Resolution

- EVAL: F1-scores of MUC/B3/CEAF.
- [CoNLL12](#): the English dataset distributed by the [CoNLL'12 shared task](#).

5.2 English Datasets

- sc = sentence count.
- tc = token count.

5.2.1 CRAFT

- [Colorado Richly Annotated Full Text Corpus](#)
 - Biomedical journal articles (sc = 19,792, tc = 554,539)

5.2.2 BOLT

- [Broad Operational Language Translation](#)
 - Conversational telephone speech (sc = 11,552, tc = 160,319)
 - Discussion forum (sc = 17,382, tc = 396,584)
 - SMS message (sc = 22,883, tc = 260,431)

5.2.3 EWT

- English Web Treebank
 - Question-answer (sc = 3,089, tc = 50,404)
 - Email (sc = 3,436, tc = 51,504)
 - Newsgroup (sc = 2,122, tc = 41,891)
 - Review (sc = 2,951, tc = 45,864)
 - Weblog (sc = 1,886, tc = 42,988)

5.2.4 OntoNotes

- OntoNotes 5.0
 - Broadcasting conversation (sc = 14,648, tc = 239,940)
 - Broadcasting news (sc = 11,867, tc = 240,241)
 - News magazine (sc = 7,960, tc = 194,926)
 - Newswire (sc = 40,491, tc = 1,038,190)
 - Pivot text (sc = 24,386, tc = 339,013)
 - Telephone conversation (sc = 10,955, tc = 112,847)
 - Weblog (sc = 11,800, tc = 262,049)

5.2.5 QuestionBank

- QuestionBank Revised
 - Question (sc = 3,989, tc = 38,100)

5.2.6 MiPACQ

- Multi-source Integrated Platform for Answering Clinical Questions
 - Clinical note (sc = 9,706, tc = 132,235)
 - Clinical question (sc = 1,980, tc = 37,178)
 - Medpedia (sc = 2,921, tc = 49,252)
 - Pathological note (sc = 1,182, tc = 22,088)

5.2.7 SHARP

- Strategic Health IT Advanced Research Projects
 - Clinical note (sc = 7,841, tc = 111,789)
 - Seattle group health note (sc = 8,268, tc = 110,208)
 - Stratified (sc = 5,022, tc = 51,629)
 - Stratified Seattle group health note (sc = 15,948, tc = 165,960)

5.2.8 THYME

- Temporal History of Your Medical Events
- Brain cancer note (sc = 21,284, tc = 263,011)
- Clinical/Pathological note (sc = 30,090, tc = 448,603)

5.2.9 Mixed

A combined dataset consisting of CRAFT, BOLT, EWT, OntoNotes, QuestionBank, MiPACQ, SHARP, and THYME.

Part-of-Speech Tags

Words:

Punctuation:

Named Entity Tags

Named entities:

Other entities:

Dependency Labels

See the [Deep Dependency Guidelines](#) for more details:

CHAPTER 6

Output Format

CHAPTER 7

Decode with CLI

To be filled.

CHAPTER 8

Train with CLI

To be filled.

`elit.structure.to_gold(key: str) → str`

`elit.structure.to_out(key: str) → str`

class `elit.structure.Sentence` (*d=None*, ***kwargs*)

Bases: `dict`

`__init__` (*d=None*, ***kwargs*)

Parameters

- **d** (*dict*) – a dictionary, if not None, all of whose fields are added to this sentence.
- **kwargs** – additional fields to be added; if keys already exist; the values are overwritten with these.

tokens

Returns the list of tokens in the sentence.

Return type list of str

part_of_speech_tags

Returns the list of part-of-speech tags corresponding to the tokens in this sentence.

Return type list of str

class `elit.structure.Document` (*d=None*, ***kwargs*)

Bases: `dict`

`__init__` (*d=None*, ***kwargs*)

Parameters

- **d** (*dict*) – a dictionary, if not None, all of whose fields are added to this document.
- **kwargs** – additional fields to be added; if keys already exist; the values are overwritten with these.

sentences

Returns the list of sentences in the document.

Return type list of Sentence

tokens

Returns list of tokens across all sentences.

Return type list of str

add_sentence (*sentence: elit.structure.Sentence*)

add_sentences (*sentences: Sequence[elit.structure.Sentence]*)

class `elit.structure.BILOU`

Bases: `object`

B = 'B'

I = 'I'

L = 'L'

O = 'O'

U = 'U'

classmethod `to_chunks` (*tags: List[str], fix: bool = False*) → List[Tuple[int, int, str]]

Parameters

- **tags** – a list of tags encoded by BILOU.
- **fix** – if True, fixes potential mismatches in BILOU (see `heuristic_fix()`).

Returns a list of tuples where each tuple contains (begin index (inclusive), end index (exclusive), label) of the chunk.

classmethod `heuristic_fix` (*tags*)

Use heuristics to fix potential mismatches in BILOU. :param tags: a list of tags encoded by BLIOU.

class `elit.structure.Node`

Bases: `object`

Node is used for constructing the NLP structure.

`__init__` ()

Initialize self. See help(type(self)) for accurate signature.

parent

Returns parent of current node

Return type *Node*

left_sibling

Returns left libling of current node

Return type *Node*

right_sibling

Returns right libling of current node

Return type *Node*

children

the list of children

Returns the list of children

Return type `list`

index_of (*child*)

the index of the child

Parameters **child** (`Node`) – a child node

Returns the index of the child node. return `None` if the given node is not a child

Return type `int` or `None`

child (*index*)

Parameters **index** (`int`) – index of a child node

Returns the index'th child of this node if exists; otherwise, `None`

Return type `Node` or `None`

first_child ()

Returns the first child of this node if exists; otherwise, `None`

Return type `Node` or `None`

last_child ()

Returns the last child of this node if exists; otherwise, `None`

Return type `Node` or `None`

find_child_by (*matcher=<function Node.<lambda>>, index=0*)

Parameters

- **matcher** (`lambda`) – the condition
- **index** (`int`) – displacement (0: 1st, 1: 2nd, etc.)

Returns the order'th child matching the specific condition of this node if exists; otherwise, `None`

Return type `Node` or `None`

add_child (*node, index=None*)

Adds a node as the index'th child of this node if it is not already a child of this node.

Parameters

- **node** (`Node`) – node is added to children
- **index** (`int`) – add child to this position

Returns if the node is added successfully, return `True`; otherwise, `False`

Return type `bool`

set_child (*node, index*)

Sets a node as the index'th child of this node if it is not already a child of this node.

Parameters

- **node** (`Node`) – node is set at the index position
- **index** (`int`) – set a child to this position

Returns the previously index'th node if added; otherwise, `None`

Return type `Node` or `None`

remove_child (*node*)

Removes a child from this node.

Parameters **node** (*Node*) – remove node

Returns True if the node exist; otherwise, False

Return type *bool*

replace_child_from (*old_node, new_node*)

Replaces the child from old_node to new_node

Parameters

- **old_node** (*Node*) – old child
- **new_node** (*Node*) – new child

Returns True if the child is replaced successfully; otherwise, False

Return type *bool*

remove ()

Removes this node from its parent. If this is the only child, removeToken its parent from its grandparent, and applies this logic recursively to the ancestors.

has_child ()

Whether current node has any child or not

Returns True if this node has any child; otherwise, False

Return type *bool*

is_child_of (*node*)

Whether current node is a child of the node or not

Parameters

- **node** – *Node*
- **node** – *Node*

Returns True if this node is a child of the specific node; otherwise, False

Return type *bool*

contains_child (*matcher=<function Node.<lambda>>*)

Whether current node contains a child with the matcher or not

Parameters **matcher** (*lambda*) – The matcher of condition

Returns True if this node contains a child of the matcher; otherwise, False

Return type *bool*

num_of_children ()

Returns numbers of children

Return type *int*

find_children_in (*first_child_index=0, last_child_index=None*)

The sublist begins at the specific position and extends to the end.

Parameters

- **first_child_index** (*int*) – the ID of the first child (inclusive)
- **last_child_index** (*int*) – the ID of the last child (exclusive)

Returns an immutable list of sub-children

Return type `list`

find_children_by (*matcher*=<function Node.<lambda>>)

Parameters **matcher** (*lambda*) – the condition

Returns the list of children matching the specific condition.

Return type `list`

grand_children ()

Returns the list of grand-children

Return type `list`

find_first_descendant_by (*matcher*=<function Node.<lambda>>)

Parameters **matcher** (*lambda*) – the condition

Returns the first descendant matching the specific condition

Return type `Node` or `None`

find_first_lowest_chained_descendant_by (*matcher*)

Parameters **matcher** (*lambda*) – the condition

Returns the first lowest descendant whose intermediate ancestors to this node all match the specific condition

Return type `Node` or `None`

descendants ()

Returns the list of all descendents (excluding this node).

Return type `list`

find_descendants_in (*depth*)

Parameters **depth** (*int*) – the level of the descendents to be retrieved (1: children, 2: childreNLPNode + grand-children, etc.).

Returns the list of descendents (excluding this node).

Return type `list`

find_single_chained_by (*matcher*=<function Node.<lambda>>)

Parameters **matcher** (*lambda*) – the condition

Returns a node in this node's subtree (including self) matching the specific condition if it is single-chained.

Return type `Node` or `None`

adapt_descendants_from (*node*)

Parameters **node** (`Node`) –

is_descendants_of (*node*)

Parameters **node** (`Node`) – the node

Returns True if the node is a descendant of the specific node

Return type `bool`

grand_parent ()

Returns the grandparent of this node if exists; otherwise, None

Return type *Node* or None

ancestor (*height*)

Parameters **height** (*int*) – height of the ancestor from this node (1: parent, 2: grandparent, etc.).

Returns the lowest ancestor matching the specific condition

Return type *Node*

find_lowest_ancestor_by (*matcher*=<function *Node*.<lambda>>)

Parameters **matcher** (*lambda*) – the condition

Returns the lowest ancestor matching the specific condition

Return type *Node*

find_highest_ancestor_by (*matcher*=<function *Node*.<lambda>>)

Parameters **matcher** (*lambda*) – the condition

Returns the highest ancestor where all the intermediate ancestors match the specific condition

Return type *Node*

lowest_common_ancestor (*node*)

Parameters **node** (*Node*) – the node

Returns the lowest common ancestor of this node and the specified node

Return type *Node* or None

ancestor_set ()

Returns the setToken of all ancestors of this node

Return type *set*

is_parent_of (*node*)

Parameters **node** (*Node*) – the node

Returns True if this node is the parent of the specific node; otherwise, False

Return type *bool*

is_ancestor_of (*node*)

Parameters **node** (*Node*) – the node

Returns True if the node is a descendant of the specific node; otherwise, False

Return type *bool*

has_parent (*matcher*=<function *Node*.<lambda>>)

Parameters **matcher** – the condition

Returns True if this node has a parent; otherwise, False

Return type *bool*

has_grand_parent ()

Returns True if this node has a grand parent; otherwise, False

Return type `bool`

siblings ()

Returns the list of all siblings

Return type `list`

left_nearest_sibling (*order=0*)

Parameters **order** (*int*) – displacement (0: left-nearest, 1: 2nd left-nearest, etc.)

Returns the order'th left-nearest sibling of this node if exists; otherwise, None

Return type `Node` or `None`

find_left_nearest_sibling_by (*matcher=<function Node.<lambda>>*)

Parameters **matcher** (*lambda*) – the condition

Returns the left nearest sibling that matches the specific condition

Return type `Node`

right_nearest_sibling (*order=0*)

Parameters **order** (*int*) – displacement (0: left-nearest, 1: 2nd left-nearest, etc.)

Returns the order'th right-nearest sibling of this node if exists; otherwise, None

Return type `Node` or `None`

find_right_nearest_sibling_by (*matcher=<function Node.<lambda>>*)

Parameters **matcher** (*lambda*) – the condition

Returns the right nearest sibling that matches the specific condition

Return type `Node`

has_left_sibling (*matcher=<function Node.<lambda>>*)

Parameters **matcher** (*lambda*) – the condition

Returns True if this node has a left sibling; otherwise, False.

has_right_sibling (*matcher=<function Node.<lambda>>*)

Parameters **matcher** (*lambda*) – the condition

Returns True if this node has a right sibling; otherwise, False.

is_sibling_of (*node*)

Parameters **node** (`Node`) – the node

Returns True if this node is a sibling of the specific node.

Return type `bool`

is_left_sibling_of (*node*)

Parameters **node** (`Node`) –

Returns True if this node is a left sibling of the specific node

Return type `bool`

is_right_sibling_of (*node*)

Parameters `node` (*Node*) –

Returns True if this node is a right sibling of the specific node

Return type *bool*

`find_node_in` (*order*, *getter*)

Parameters

- **order** (*int*) – 0: self, 1: nearest, 2: second nearest, etc.
- **getter** (*str*) – takes a node and returns a node.

Returns the order'th node with respect to the getter.

Return type *Node*

`find_nearest_node_by` (*getter*, *matcher*=<function *Node*.<lambda>>)

Parameters

- **getter** (*str*) – takes a node and returns a node.
- **matcher** (*lambda*) – takes a node and the supplement, and returns true if its field matches to the specific predicate.

Returns the first node matching the specific condition

Return type *Node* or *None*

`isolate` ()

Isolates this node from its parent, children, and siblings

`second_order` (*getter*)

Parameters **getter** (*str*) – takes a node and returns a list of nodes.

Returns the list of second order elements according to the getter.

Return type *list*

CHAPTER 10

Tokenizers

e

`elit.structure`, 23

Symbols

`__init__()` (elit.structure.Document method), 23

`__init__()` (elit.structure.Node method), 24

`__init__()` (elit.structure.Sentence method), 23

A

`adapt_descendants_from()` (elit.structure.Node method), 27

`add_child()` (elit.structure.Node method), 25

`add_sentence()` (elit.structure.Document method), 24

`add_sentences()` (elit.structure.Document method), 24

`ancestor()` (elit.structure.Node method), 28

`ancestor_set()` (elit.structure.Node method), 28

B

B (elit.structure.BILOU attribute), 24

BILOU (class in elit.structure), 24

C

`child()` (elit.structure.Node method), 25

children (elit.structure.Node attribute), 24

`contains_child()` (elit.structure.Node method), 26

D

`descendants()` (elit.structure.Node method), 27

Document (class in elit.structure), 23

E

elit.structure (module), 23

F

`find_child_by()` (elit.structure.Node method), 25

`find_children_by()` (elit.structure.Node method), 27

`find_children_in()` (elit.structure.Node method), 26

`find_descendants_in()` (elit.structure.Node method), 27

`find_first_descendant_by()` (elit.structure.Node method), 27

`find_first_lowest_chained_descendant_by()` (elit.structure.Node method), 27

`find_highest_ancestor_by()` (elit.structure.Node method), 28

`find_left_nearest_sibling_by()` (elit.structure.Node method), 29

`find_lowest_ancestor_by()` (elit.structure.Node method), 28

`find_nearest_node_by()` (elit.structure.Node method), 30

`find_node_in()` (elit.structure.Node method), 30

`find_right_nearest_sibling_by()` (elit.structure.Node method), 29

`find_single_chained_by()` (elit.structure.Node method), 27

`first_child()` (elit.structure.Node method), 25

G

`grand_children()` (elit.structure.Node method), 27

`grand_parent()` (elit.structure.Node method), 27

H

`has_child()` (elit.structure.Node method), 26

`has_grand_parent()` (elit.structure.Node method), 28

`has_left_sibling()` (elit.structure.Node method), 29

`has_parent()` (elit.structure.Node method), 28

`has_right_sibling()` (elit.structure.Node method), 29

heuristic_fix() (elit.structure.BILOU class method), 24

I

I (elit.structure.BILOU attribute), 24

`index_of()` (elit.structure.Node method), 25

`is_ancestor_of()` (elit.structure.Node method), 28

`is_child_of()` (elit.structure.Node method), 26

`is_descendants_of()` (elit.structure.Node method), 27

`is_left_sibling_of()` (elit.structure.Node method), 29

`is_parent_of()` (elit.structure.Node method), 28

`is_right_sibling_of()` (elit.structure.Node method), 29

`is_sibling_of()` (elit.structure.Node method), 29

`isolate()` (elit.structure.Node method), 30

L

L (elit.structure.BILOU attribute), 24

`last_child()` (elit.structure.Node method), 25
`left_nearest_sibling()` (elit.structure.Node method), 29
`left_sibling` (elit.structure.Node attribute), 24
`lowest_common_ancestor()` (elit.structure.Node method),
28

N

`Node` (class in elit.structure), 24
`num_of_children()` (elit.structure.Node method), 26

O

`O` (elit.structure.BILOU attribute), 24

P

`parent` (elit.structure.Node attribute), 24
`part_of_speech_tags` (elit.structure.Sentence attribute), 23

R

`remove()` (elit.structure.Node method), 26
`remove_child()` (elit.structure.Node method), 25
`replace_child_from()` (elit.structure.Node method), 26
`right_nearest_sibling()` (elit.structure.Node method), 29
`right_sibling` (elit.structure.Node attribute), 24

S

`second_order()` (elit.structure.Node method), 30
`Sentence` (class in elit.structure), 23
`sentences` (elit.structure.Document attribute), 23
`set_child()` (elit.structure.Node method), 25
`siblings()` (elit.structure.Node method), 29

T

`to_chunks()` (elit.structure.BILOU class method), 24
`to_gold()` (in module elit.structure), 23
`to_out()` (in module elit.structure), 23
`tokens` (elit.structure.Document attribute), 24
`tokens` (elit.structure.Sentence attribute), 23

U

`U` (elit.structure.BILOU attribute), 24