SEMSTR Documentation

Release 1.2.2

Daniel Hershcovich

Jun 28, 2021

Contents:

For more information about how to use this library, see the *API Documentation*.

CHAPTER 1

semstr.constraints Module

1.1 Functions

contains(s, tag)	
incoming_tags(node, except_edge)	
outgoing_tags(node, except_edge)	
set_prod(set1[, set2])	
tags(node, except_edge, direction)	

1.1.1 contains

semstr.constraints.**contains**(*s*, *tag*)

1.1.2 incoming_tags

semstr.constraints.**incoming_tags**(*node*, *except_edge*)

1.1.3 outgoing_tags

semstr.constraints.**outgoing_tags**(*node*, *except_edge*)

1.1.4 set_prod

semstr.constraints.**set_prod**(*set1*, *set2=None*)

1.1.5 tags

semstr.constraints.**tags**(*node*, *except_edge*, *direction*)

1.2 Classes

Constraints([multigraph, ...])	
Direction	An enumeration.
Enum	Generic enumeration.
TagRule(trigger[, allowed, disallowed])	
Valid([valid, message])	

1.2.1 Constraints

class semstr.constraints.**Constraints**(*multigraph=False*, *require_implicit_childless=True*, *allow_orphan_terminals=False*, *allow_root_terminal_children=False*, *top_level_allowed=None*, *top_level_only=None*, *possible_multiple_incoming=()*, *childless_incoming_trigger=()*, *childless_outgoing_allowed=()*, *unique_incoming=()*, *unique_outgoing=()*, *mutually_exclusive_incoming=()*, *mutually_exclusive_outgoing=()*, *exclusive_outgoing=()*, *required_outgoing=()*, *implicit=False*, ***kwargs*)

Bases: object

Methods Summary

allow_action(action, history)	
allow_child(node, tag)	
allow_edge(edge)	
allow_label(node, label)	
allow_parent(node, tag)	

Methods Documentation

allow_action(*action*, *history*)

allow_child(*node*, *tag*)

allow_edge(*edge*)

allow_label(*node*, *label*)

allow_parent(*node*, *tag*)

1.2.2 Direction

class semstr.constraints.**Direction**

Bases: enum.Enum

An enumeration.

Attributes Summary

incoming	
outgoing	

Attributes Documentation

incoming = 0

outgoing = 1

1.2.3 TagRule

class semstr.constraints.**TagRule**(*trigger*, *allowed=None*, *disallowed=None*)
Bases: object

Methods Summary

violation(node, tag, direction[, message])	

Methods Documentation

violation(*node*, *tag*, *direction*, *message=False*)

1.2.4 Valid

class semstr.constraints.**Valid**(*valid=True*, *message=''*)
Bases: object

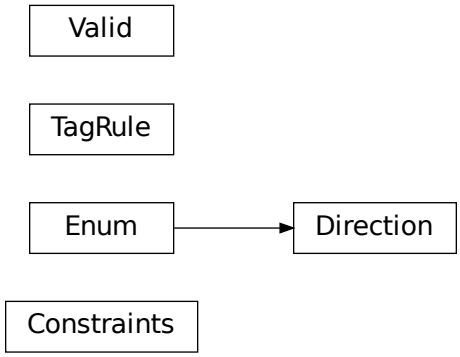
Methods Summary

__call__(valid[, message])	Call self as a function.

Methods Documentation

__call__(*valid*, *message=None*)
Call self as a function.

1.3 Class Inheritance Diagram



CHAPTER 2

semstr.convert Module

2.1 Functions

add_boolean_option(argparser, name, description)	
add_convert_args(p)	
add_verbose_arg(argparser, **kwargs)	
from_amr(lines[, passage_id, ...])	Converts from parsed text in AMR PENMAN format to a Passage object.
from_conll(lines, passage_id[, ...])	Converts from parsed text in CoNLL format to a Passage object.
from_conllu(lines[, passage_id, ...])	Converts from parsed text in Universal Dependencies format to a Passage object.
from_export(lines[, passage_id, return_original])	Converts from parsed text in NeGra export format to a Passage object.
from_sdp(lines, passage_id[, mark_aux, ...])	Converts from parsed text in SemEval 2015 SDP format to a Passage object.
glob(pathname, *[, recursive])	Return a list of paths matching a pathname pattern.
iter_files(patterns)	
iter_passages(patterns[, desc, ...])	
main(args)	
map_labels(passage, label_map_file)	
print_errors(errors, passage_id[, id_len])	
to_amr(passage[, metadata, wikification, ...])	Convert from a Passage object to a string in AMR PENMAN format (export)
to_conll(passage[, test, tree, preprocess])	Convert from a Passage object to a string in CoNLL-X format (conll)
to_conllu(passage[, test, enhanced, preprocess])	Convert from a Passage object to a string in Universal Dependencies format (conllu)

Continued on next page

Table 1 – continued from previous page

to_export(passage[, test, tree])	Convert from a Passage object to a string in NeGra export format (export)
to_sdp(passage[, test, tree, mark_aux, ...])	Convert from a Passage object to a string in SemEval 2015 SDP format (sdp)
validate(passage[, normalization, ...])	
write_passage(passage[, out_dir, ...])	

2.1.1 add_convert_args

semstr.convert.**add_convert_args**(*p*)

2.1.2 from_amr

semstr.convert.**from_amr**(*lines*, *passage_id=None*, *return_original=False*, *save_original=True*, *wikification=False*, *placeholders=True*, ***kwargs*)

Converts from parsed text in AMR PENMAN format to a Passage object.

Parameters

- **lines** – iterable of lines in AMR PENMAN format, describing a single passage.
- **passage_id** – ID to set for passage, overriding the ID from the file
- **save_original** – whether to save original AMR text in passage.extra
- **return_original** – return triple of (UCCA passage, AMR string, AMR ID)
- **wikification** – whether to use wikification for replacing node labels with placeholders based on tokens
- **placeholders** – introduce placeholders into node labels when they include the terminal's text?

:return generator of Passage objects

2.1.3 from_conll

semstr.convert.**from_conll**(*lines*, *passage_id*, *return_original=False*, *dep=False*, *preprocess=True*, ***kwargs*)

Converts from parsed text in CoNLL format to a Passage object.

Parameters

- **lines** – iterable of lines in CoNLL format, describing a single passage.
- **passage_id** – ID to set for passage
- **return_original** – return triple of (UCCA passage, CoNLL string, sentence ID)
- **dep** – return dependency graph rather than converted UCCA passage
- **preprocess** – preprocess the dependency graph before converting to UCCA (or returning it)?

:return generator of Passage objects

2.1.4 from_conllu

semstr.convert.**from_conllu**(*lines*, *passage_id=None*, *return_original=False*, *annotate=False*, *terminals_only=False*, *dep=False*, *enhanced=True*, *preprocess=True*, ***kwargs*)

Converts from parsed text in Universal Dependencies format to a Passage object.

Parameters

- **lines** – iterable of lines in Universal Dependencies format, describing a single passage.
- **passage_id** – ID to set for passage
- **return_original** – return triple of (UCCA passage, Universal Dependencies string, sentence ID)
- **annotate** – whether to save dependency annotations in "extra" dict of layer 0
- **terminals_only** – create only terminals (with any annotation if specified), no non-terminals
- **dep** – return dependency graph rather than converted UCCA passage
- **enhanced** – whether to include enhanced edges
- **preprocess** – preprocess the dependency graph before converting to UCCA (or returning it)?

:return generator of Passage objects

2.1.5 from_export

semstr.convert.**from_export**(*lines*, *passage_id=None*, *return_original=False*, ***kwargs*)

Converts from parsed text in NeGra export format to a Passage object.

Parameters

- **lines** – iterable of lines in NeGra export format, describing a single passage.
- **passage_id** – ID to set for passage, overriding the ID from the file
- **return_original** – return triple of (UCCA passage, Export string, sentence ID)

:return generator of Passage objects

2.1.6 from_sdp

semstr.convert.**from_sdp**(*lines*, *passage_id*, *mark_aux=False*, *return_original=False*, *dep=False*, *preprocess=True*, ***kwargs*)

Converts from parsed text in SemEval 2015 SDP format to a Passage object.

Parameters

- **lines** – iterable of lines in SDP format, describing a single passage.
- **passage_id** – ID to set for passage
- **mark_aux** – add a preceding # for labels of auxiliary edges added
- **return_original** – return triple of (UCCA passage, SDP string, sentence ID)
- **dep** – return dependency graph rather than converted UCCA passage

- **preprocess** – preprocess the dependency graph before converting to UCCA (or returning it)?

:return generator of Passage objects

2.1.7 iter_files

semstr.convert.**iter_files**(*patterns*)

2.1.8 iter_passages

semstr.convert.**iter_passages**(*patterns*, *desc=None*, *input_format=None*, *prefix=''*, *label_map=None*, *output_format=None*, ***kwargs*)

2.1.9 main

semstr.convert.**main**(*args*)

2.1.10 map_labels

semstr.convert.**map_labels**(*passage*, *label_map_file*)

2.1.11 to_amr

semstr.convert.**to_amr**(*passage*, *metadata=True*, *wikification=True*, *use_original=True*, *verbose=False*, *default_label=None*, ***kwargs*)

Convert from a Passage object to a string in AMR PENMAN format (export)

Parameters

- **passage** – the Passage object to convert
- **metadata** – whether to print ::id and ::tok lines
- **wikification** – whether to wikify named concepts, adding a :wiki triple
- **use_original** – whether to use original AMR text from passage.extra
- **verbose** – whether to print extra information
- **default_label** – label to use in case node has no label attribute

:return list of lines representing an AMR in PENMAN format, constructed from the passage

2.1.12 to_conll

semstr.convert.**to_conll**(*passage*, *test=False*, *tree=False*, *preprocess=True*, ***kwargs*)

Convert from a Passage object to a string in CoNLL-X format (conll)

Parameters

- **passage** – the Passage object to convert
- **test** – whether to omit the head and deprel columns. Defaults to False
- **tree** – whether to omit rows for non-primary parents. Defaults to False

- **preprocess** – preprocess the converted dependency graph before returning it?

:return list of lines representing the dependencies in the passage

2.1.13 to_conllu

semstr.convert.**to_conllu**(*passage*, *test=False*, *enhanced=True*, *preprocess=True*, ***kwargs*)

Convert from a Passage object to a string in Universal Dependencies format (conllu)

Parameters

- **passage** – the Passage object to convert
- **test** – whether to omit the head and deprel columns. Defaults to False
- **enhanced** – whether to include enhanced edges
- **preprocess** – preprocess the converted dependency graph before returning it?

:return list of lines representing the semantic dependencies in the passage

2.1.14 to_export

semstr.convert.**to_export**(*passage*, *test=False*, *tree=False*, ***kwargs*)

Convert from a Passage object to a string in NeGra export format (export)

Parameters

- **passage** – the Passage object to convert
- **test** – whether to omit the edge and parent columns. Defaults to False
- **tree** – whether to omit columns for non-primary parents. Defaults to False

:return list of lines representing a (discontinuous) tree structure constructed from the passage

2.1.15 to_sdp

semstr.convert.**to_sdp**(*passage*, *test=False*, *tree=False*, *mark_aux=False*, *preprocess=True*, ***kwargs*)

Convert from a Passage object to a string in SemEval 2015 SDP format (sdp)

Parameters

- **passage** – the Passage object to convert
- **test** – whether to omit the top, head, frame, etc. columns. Defaults to False
- **tree** – whether to omit columns for non-primary parents. Defaults to False
- **mark_aux** – omit edges with labels with a preceding #
- **preprocess** – preprocess the converted dependency graph before returning it?

:return list of lines representing the semantic dependencies in the passage

2.1.16 write_passage

semstr.convert.**write_passage**(*passage*, *out_dir='.'*, *output_format=None*, *binary=False*, *verbose=False*, *label_map=False*, *split=False*, *join=None*, ***kwargs*)

CHAPTER 3

semstr.evaluate Module

3.1 Functions

add_boolean_option(argparser, name, description)	
add_verbose_arg(argparser, **kwargs)	
align_fields(fields, titles, title2index)	Make sure score fields for individual passage are aligned with summary result fields by inserting empties
evaluate_all(evaluate, files[, name, ...])	
evaluate_amr(*args, **kwargs)	
evaluate_conllu(*args, **kwargs)	
evaluate_sdp(*args, **kwargs)	
main(args)	
passage_format(filename)	
read_files(files[, verbose, force_basename])	
summarize(scores[, errors])	
write_csv(filename, rows)	

3.1.1 align_fields

semstr.evaluate.**align_fields**(*fields*, *titles*, *title2index*)

Make sure score fields for individual passage are aligned with summary result fields by inserting empties

3.1.2 evaluate_all

semstr.evaluate.**evaluate_all**(*evaluate*, *files*, *name=None*, *verbose=0*, *quiet=False*, *basename=False*, *matching_ids=False*, *units=False*, *unlabeled=False*, ***kwargs*)

3.1.3 evaluate_amr

semstr.evaluate.**evaluate_amr**(**args*, ***kwargs*)

3.1.4 evaluate_conllu

semstr.evaluate.**evaluate_conllu**(**args*, ***kwargs*)

3.1.5 evaluate_sdp

semstr.evaluate.**evaluate_sdp**(**args*, ***kwargs*)

3.1.6 main

semstr.evaluate.**main**(*args*)

3.1.7 passage_format

semstr.evaluate.**passage_format**(*filename*)

3.1.8 read_files

semstr.evaluate.**read_files**(*files*, *verbose=0*, *force_basename=False*, ***kw*)

3.1.9 summarize

semstr.evaluate.**summarize**(*scores*, *errors=False*)

3.1.10 write_csv

semstr.evaluate.**write_csv**(*filename*, *rows*)

3.2 Classes

ConvertedPassage(converted[, original, ...])	
Scores(scores)	Keeps score objects from multiple formats and/or languages
groupby(iterable[, key])	keys and groups from the iterable.
repeat(object [,times])	for the specified number of times.

3.2.1 ConvertedPassage

class semstr.evaluate.**ConvertedPassage**(*converted*, *original=None*, *passage_id=None*, *converted_format=None*, *in_converter=None*, *out_converter=None*)

Bases: object

3.2.2 Scores

class semstr.evaluate.**Scores**(*scores*)

Bases: object

Keeps score objects from multiple formats and/or languages

Methods Summary

aggregate(scores)
average_f1(*args, **kwargs)
details(average_f1)
fields(*args, **kwargs)
print(*args, **kwargs)
titles(*args[, prefix])

Methods Documentation

static aggregate(*scores*)

average_f1(**args*, ***kwargs*)

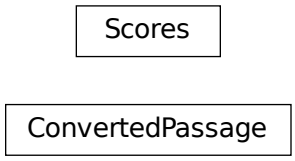
details(*average_f1*)

fields(**args*, ***kwargs*)

print(**args*, ***kwargs*)

titles(**args*, *prefix=True*, ***kwargs*)

3.3 Class Inheritance Diagram



CHAPTER 4

semstr.validation Module

4.1 Functions

amr_constraints(*args, **kwargs)	
check_implicit_children(constraints, node)	
check_multigraph(constraints, node)	
check_multiple_incoming(constraints, node)	
check_orphan_terminals(constraints, terminal)	
check_required_outgoing(constraints, node)	
check_root_terminal_children(constraints, ...)	
check_tag_rules(constraints, node)	
check_top_level_allowed(constraints, l1)	
check_top_level_only(constraints, l1, node)	
conllu_constraints(*args, **kwargs)	
detect_cycles(passage)	
join(edges)	
print_errors(errors, passage_id[, id_len])	
sdp_constraints(*args, **kwargs)	
ucca_constraints(*args, **kwargs)	
validate(passage[, normalization, ...])	

4.1.1 amr_constraints

semstr.validation.**amr_constraints**(**args*, ***kwargs*)

4.1.2 check_implicit_children

semstr.validation.**check_implicit_children**(*constraints*, *node*)

4.1.3 check_multigraph

semstr.validation.**check_multigraph**(*constraints*, *node*)

4.1.4 check_multiple_incoming

semstr.validation.**check_multiple_incoming**(*constraints*, *node*)

4.1.5 check_orphan_terminals

semstr.validation.**check_orphan_terminals**(*constraints*, *terminal*)

4.1.6 check_required_outgoing

semstr.validation.**check_required_outgoing**(*constraints*, *node*)

4.1.7 check_root_terminal_children

semstr.validation.**check_root_terminal_children**(*constraints*, *l1*, *terminal*)

4.1.8 check_tag_rules

semstr.validation.**check_tag_rules**(*constraints*, *node*)

4.1.9 check_top_level_allowed

semstr.validation.**check_top_level_allowed**(*constraints*, *l1*)

4.1.10 check_top_level_only

semstr.validation.**check_top_level_only**(*constraints*, *l1*, *node*)

4.1.11 conllu_constraints

semstr.validation.**conllu_constraints**(**args*, ***kwargs*)

4.1.12 detect_cycles

semstr.validation.**detect_cycles**(*passage*)

4.1.13 join

semstr.validation.**join**(*edges*)

4.1.14 print_errors

semstr.validation.**print_errors**(*errors*, *passage_id*, *id_len=None*)

4.1.15 sdp_constraints

semstr.validation.**sdp_constraints**(**args*, ***kwargs*)

4.1.16 ucca_constraints

semstr.validation.**ucca_constraints**(**args*, ***kwargs*)

4.1.17 validate

semstr.validation.**validate**(*passage*, *normalization=False*, *extra_normalization=False*, *ucca_validation=False*, *output_format=None*, ***kwargs*)

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

Symbols

A

C

D

E

F

I

J

M