SharedParser

Release 1.0

Bogdana Simionica, Stefan Stefanache

Dec 19, 2022

CONTENTS:

CHAPTER

ONE

FLCD_TEAM

1.1 grammar module

Implementation of grammar

`class grammar.GrammarParseSeparators(value)`

Bases: `Enum`

An enumeration.

`class grammar.Grammar(starting_symbol, terminals=NOTHING, nonterminals=NOTHING, productions=NOTHING)`

Bases: `object`

This class implements the basic functionality of working with a grammar, including parsing a grammar specification file written in a reduced form of EBNF.

`starting_symbol: str`

Initial state symbol (S)

`terminals: List[str]`

List of terminals

`nonterminals: List[str]`

List of nonterminals

`productions: Dict[str, List[str]]`

Mapping of nonterminals to lists of coresponding productions

`static get_grammar_from_file(file_name)`

Reads grammar specification from a file and returns the coresponding Grammar.

Raises

`The grammar specification must be written in the specified reduced` –

EBNF form and the file must exist (see example specification). Otherwise, an exception is thrown.

Parameters

`file_name` (`str`) – Path to the file containing the grammar definition

Return type

`Grammar`

Returns

Coresponding Grammar if file exists and contains a valid definition

get_productions_for_nonterminal(*nonterminal*)

Returns the productions coresponding to a given nonterminal.

Parameters
`nonterminal` (`str`) – The nonterminal to get productions for

Return type
`List[str]`

verify_CFG()

Checks if the Grammar is context-free.

Return type
`bool`

1.2 output module

class output.**ParserOutput**(*grammar*)

Bases: `object`

get_recursive_table(*index*, *parent*, *right_sibling*, *string_products*, *products_stack*)

get_string_products(*alpha*)

get_table(*alpha*)

print_pretty_table()

print_pretty_table_to_file(*filename*)

print_table()

class output.**Row**(*info*, *parent*, *right_sibling*)

Bases: `object`

return_data()

1.3 parser module

class parser.**Parser**(*grammar*, *state=ParsingStates.NORMAL_STATE*, *current_position=1*, *alpha=NOTHING*, *beta=NOTHING*, *w=NOTHING*, *index_error=0*)

Bases: `object`

advance()

algorithm_descendent_recursive(*filename*)

alpha: list

another_try()

back()

beta: list

build_string_of_prod()

current_position: int

expand()

grammar: *Grammar*

index_error: int

momentary_insuccess()

read_sequence_from_file(*filename*)

state: *ParsingStates*

success()

w: list

class parser.**ParsingStates**(*value*)

Bases: Enum

An enumeration.

BACK_STATE = 'b'

ERROR_STATE = 'e'

FINAL_STATE = 'f'

NORMAL_STATE = 'q'

CHAPTER TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX