

Class Grammar:

```
self.start: str = None          # starting nonterminal

self.terminals: list[str] = []  # list of terminals

self.nonterminals: list[str] = [] # list of nonterminals

self productions: dict[str, list[list[str]]] = {} # dictionary of nonterminal -> list of productions
```

```
def _load(self, filepath: str) -> None:
```

```
    """ Loads grammar from `filepath`. """
```

```
def is_cfg(self) -> bool:
```

```
    """ Checks if grammar is context-free. """
```

class Action:

```
    """ Parent class for all 3 types of actions: shift, reduce and accept """
```

```
    Pass
```

class Accept(Action):

```
    def apply(self, config: Configuration) -> None:
```

```
        """ Performs accept action on LR(0) configuration. """
```

class Reduce(Action):

```
    def __init__(self, nonterminal: str, production: list[str], production_no: int):
```

```
        """ Initializes a reduce action for nonterminal and its production. """
```

```
    def apply(self, config: Configuration, table: dict[State, Action]) -> None:
```

```
        """ Performs reduce action on LR(0) configuration. """
```

class Shift(Action):

```
    def __init__(self):
```

```
        self.goto: dict[str, State] = {} //the dictionary for goto
```

```
    def apply(self, config: Configuration) -> None:
```

```
        """ Performs shift action on LR(0) configuration. """
```

class Configuration:

```
    """ A configuration of LR(0) parser. """
```

class Item:

```

""" An item of the LR(0) parser. """

nonterminal: str      # the nonterminal on the left-hand side production

production: list[str]  # the right-hand side of the production

current_symbol_idx: int # the symbol of the production that is currently processed (it represents
the symbol right after the dot)

class Parser:

    """ LR(0) parser. """

    def __init__(self, grammar_path: str | Path):

        """ Initializes the parser with the given grammar.

        Args:

            grammar_path (str | Path): path to grammar description
        """

    def parse(self, pif: ProgramInternalForm) -> ParserOutput:

        """ Parses the PIF using the syntax described in `syntax_path`.

        The LR(0) parser is used.
        """

    def _augment_grammar(cls, grammar: Grammar) -> Grammar:

        """ Returns an augmented grammar of `grammar`.

        Args:

            grammar (Grammar): grammar of the language

        Returns: an augmented grammar of `grammar`
        """

    def _get_closure(self, initial_item: Item) -> set[Item]:

        """ Computes the closure of `initial_item`.

        Args:

            initial_item (Item): the item on which the closure is computed

        Returns: the closure as a list of items
        """

    def _goto(self, state: State, symbol: str) -> None | State:

```

""" Performs the goto action on `state` using `symbol`.

Args:

state (State): the state on which the goto is performed

symbol (str): the symbol used in goto

Returns: the new State if one was found, otherwise None

"""

def _get_canonical_collection(self) -> set[State]:

""" Computes the canonical collection of the grammar.

Returns: the canonical collection as a list of states

"""

def _generate_parsing_table(self, canonical_collection: set[State]) -> dict[State, Action]:

""" Generates the LR(0) parsing table based on the canonical collection.

Args:

canonical_collection (set[State]): the canonical collection of the augmented grammar

Returns: the parsing table as a mapping from a State to an Action

"""

def _find_next_state(_item: Item) -> State | None:

""" Finds the next state based on an item by checking what state contains that item. """

class ParserOutput:

""" Represents the output of parsing """

class Entry:

""" Entry in parsing table. """

symbol: str

parent: int

right_sibling: int

def __init__(self, grammar: Grammar, output_stack: Stack):

""" Initializes and constructs the parsing table given the grammar and the string of production found in `output_stack`.

Args:

grammar (Grammar): grammar of the language

output_stack (Stack): output stack of LR(0) parser (contains string of productions)

"""

def _construct(self, grammar: Grammar, output_stack: Stack) -> list["ParserOutput.Entry"]:

""" Constructs the parsing table given the grammar and the string of production found in
`output_stack`.

Args:

grammar (Grammar): grammar of the language

output_stack (Stack): output stack of LR(0) parser (contains string of productions)

Returns: the parsing table represented as a list of `ParserOutput.Entry` instances

"""

def dump(self, filepath: str | Path) -> None:

""" Dumps the parsing table to a file.

Args:

filepath (str | Path): path to file

"""

class State:

""" A state of the LR(0) parser. """