Lab 7 Andrei Candet and Radu Ceaca

Link to git:

https://github.com/cinnamonbreakfast/flcd/tree/main/lab5_final_%40raduceaca Assignment for a team of 2 students!

+Run example on last page

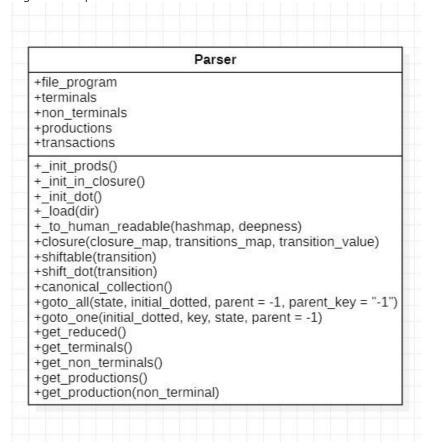
Statement: Implement a parser algorithm (cont)

PART 2: Deliverables

- 1. Algorithm corresponding to parsing tables (if needed) and parsing strategy
- 2. Class ParserOutput DS and operations corresponding to choice 2.a/2.b/2.c (<u>lab 5</u>) (required operations: transform parsing tree into representation; print DS to screen and to file)

Remark:

- if the table contains conflicts, you will be helped to solve them. It is important to print message containing row (symbol in LL(1), respectively state in LR(0)) and column (symbol) where the conflict appears. For LL(1) values ($\alpha\alpha$,i) might also help



+ parse string(string)

	action			Goto	
	а	b	\$	Α	S
0	S ₃	S ₄	-	2	1
1			acc		
2	S ₃	S ₄	3	6	
3	S ₃	S ₄		6	
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

```
_init_prods():
        Initialize the production directory
_init_in_closure()
        Initialeze the closure map
_init_dot ()
        Initialization method for closure
_load(dir)
Load
         data
                 from
                           file
                                   closure(closure_map,
transitions_map, transition_value)
                                         Compute the
closure_map
shiftable(transition)
        Check if the dot can be shifted
shift_dot(transition)
        Call the shiftable function and shift the dot
canonical_collection()
        Populate the canonical collection
goto_all(state, initial_dotted, parent=-1, parent_key="-1")
        Goes through every state
goto_one(initial_dotted, key, state, parent=-1)
        Goes to a single state get_reduced()
```

```
Returns the reduced map
get_terminals()
         Returns terminals
get_non_terminals()
         Returns non terminals
get_produtions()
         Returns productions
get_production(non_terminal)
         Returns the production of a non terminal
parse_string(string)
        pre: string to parse
         post: parsing table
                                             Run Example
Grammar:
a b c
S a S b S
S a S
Sequence to be parsed: aac
state 0
S':[['.', 'S']]
S:[['.', 'a', 'S', 'b', 'S'], ['.', 'a', 'S'], ['.', 'c']]
state 1
S': [['S', '.']]
S:[['a', '.', 'S', 'b', 'S'], ['.', 'a', 'S', 'b', 'S'], ['.', 'a', 'S'], ['.', 'c']]
state 3
S:[['a', '.', 'S'], ['.', 'a', 'S', 'b', 'S'], ['.', 'a', 'S'], ['.', 'c']]
```

state 4

S:[['c', '.']]

state 5

S:[['a', 'S', '.', 'b', 'S']]

state 6

S:[['a', 'S', '.']]

state 7

S: [['a', 'S', 'b', '.', 'S'], ['.', 'a', 'S', 'b', 'S'], ['.', 'a', 'S'], ['.', 'c']]

state 8

S:[['a', 'S', 'b', 'S', '.']]

S0: {'S': 1, 'a': 3, 'c': 4}

S1: {'\$': 'accept'}

S2: {'a': 3, 'c': 4, 'S': 5}

S3: {'a': 3, 'c': 4, 'S': 6}

\$4: {'a': 'r3', 'b': 'r3', 'c': 'r3', '\$': 'r3'}

S5: {'b': 7}

S6: {'a': 'r2', 'b': 'r2', 'c': 'r2', '\$': 'r2'}

S7: {'a': 3, 'c': 4, 'S': 8}

S8: {'a': 'r1', 'b': 'r1', 'c': 'r1', '\$': 'r1'}

Work Stack	Input Stack	Output band	
\$0	aac\$	Empty	
\$0a3	ac\$	Empty	
\$0a3a3	c\$	Empty	
\$0a3a3c4	\$	Empty	
\$0a3a3S6	\$	3	
\$0a3S6	\$	2,3	
\$0\$1	\$	2,2,3	
accepted	\$	2,2,3	

Canonical Table

State	Action a	Action b	Action c	Action \$	Go to S	Go to A
S 0	2				1	
S 1				accept		
S2		4	5			3
S 3	r1	r1	r1	r1		
S4		4	5			6
S5	r3	r3	r3	r3		
S 6	r2	r2	r2	r2		