https://github.com/Aurelian-Iancu/FLCD-Parser

Documentation

Grammar:

- Terminals: the set of terminals
- Nonterminals: the set of nonterminals
- Start: the starting symbol
- Productions: the set of productions
- Special terminals: a map which contains the special terminals and their ,,translation”
- parseTerminalsAndNonTerminals(string input): Method used to parse the terminals and the nonterminals from the file
- parseProduction(string production): Method used to parse the productions from the file
- Constructor(string file): which reads from the file then creates and stores the sets of terminals, nonterminals and productions
- displayTerminals(): method that prints the terminals on the screen
- displayNonterminals(): method that prints the nonterminals to the screen
- displayProductions(): method that prints the productions to the screen
- displayProductions(string nonterminal): method that prints the productions of a particular nonterminal

Item:

- represents an LR(0) item
- lhs: the left-hand-side of an item
- rhs: the right-hand-side of an item
- dotPosition: the position of the dot in the rhs of the item

State:

- Action: state action for the parsing table
- ClosureItems: the items for which the closure is computed in the state
- Closure: the items in the state
- hasSameClosure(const vector<Item>& otherClosure): return true if 2 states have the same closure items
- getAtomAfterDot(): returns the atoms that are situated after the . in the Item

- allDotsNotAtEnd(): returns true if not all the dots are at the end of each Item of the closure
- allDotsAtEnd(): returns true if all the dots are at the end of each item of the closure
- getAction(string startingAtom): returns the action in which a state is: Accept if the state has the dot at the end of the enriched grammar production, Reduce if the dot is at the end and there is only 1 item, shit if the dot is not at the end of any item, reduce-reduce conflict if there are multiple items with the dot at the end and shift-reduce conflict otherwise

Parser:

- grammar: the grammar of the parser
- connections: map used to represent the connections of the gotos and the relations from states
- cannonicalCollection: the set of states
- parsingTable: the parsing table with go to and action for each state. This is done using an additional class named TableRow
- closure(items): compute the closure of the given items and returns a state which contains the computed items
- goto(state,atom): compute the closure of all the items from the state that contain the dot before the given symbol, with the dot shifted one position;
- createCannonicalCollection(): compute the canonical collection of states for the parser grammar; go through each state and compute new states for all the symbols that have dot in front of them
- createParsingTable(): go through each state from the canonical collection and depending on the type of the action for each state, add the corresponding information: for accept-> only action, for reduce-> action and the number of the production to be used, for shift-> action and the gotos from that state; raise shift-reduce and reduce-reduce conflicts
- getStateIndex(): returns the index of a state in the canonicalCollection
- parse(atoms): parse the given sequence until the sequence is accepted or there is an error; accept occurs when the state on top of the stack is the accept state; when the action is shift, the top of the input stack is shifter and put at the top of the work stack, the new state is computed; when the action is reduce, the production with the corresponding number is used to reduce from the top of the work stack and prepended to the output band;

ParserOutput:

- Node: has an id, the value of the atom, father and sibling fields;
- Grammar: the grammar for the parser;
- Output: the vector in which the output is built;

- parsingTree: list of Nodes;
- createParsingTree(): transform the output, starting from the grammar's initial symbol and going through each production from the output; for each node compute the father and right sibling, if it exists;
- display(): function to print the parsing tree on the screen;
- saveToFile(): function to save the result in output.txt file;