PROGRAMMING LANGUAGES – HW3 REPORT

ALPER YAŞAR – 151044072

Runnig :

swipl -s 151044072_yasar_alper.pl

main.

Generate a tokenizer function by giving a list of regular expressions and a function, which will be applied to captured objects. The regular expression will be applied to the **beginning** of the input string in the listed order.

E.g.:

```
(define-tokenizer pairs ()
  ("\\(([^:-]+),([^,]+)\\)" #'id))
```

This will generate the function `pairs`, which takes, save for the input, no argument, and returns, applied to e.g. "mammal(horse).", since `#'id` is given as the function to be applied to the results, (("mammal" ("horse")) ()).

```
(define-tokenizer calcs ()
  ("add\\s*\\((\\d+),(\\d+)\\)"
   #'(lambda (res)
       (apply #'+ (mapcar #'parse-integer res))))
  ("sub\\s*\\((\\d+),(\\d+)\\)"
   #'(lambda (res)
       (apply #'- (mapcar #'parse-integer res)))))
```

Generating a parser

<program> ::= <clause list> <query> | <query>
<clause list> ::= <clause> | <clause list> <clause>
<clause> ::= <predicate> . | <predicate> :- <predicate list>.
<predicate list> ::= <predicate> | <predicate list> , <predicate>
<predicate> ::= <atom> | <atom> (<term list>)
<term list> ::= <term> | <term list> , <term>
<term> ::= <numeral> | <atom> | <variable>
<query> ::= ?- <predicate list>.
<atom> ::= <small atom> | "<string>"
<small atom> ::= <lowercase letter> | <small atom> <character>
<variable> ::= <uppercase letter> | <variable> <character>
<lowercase letter> ::= a | b | c | ... | x | y | z
<uppercase letter> ::= A | B | C | ... | X | Y | Z
<numeral> ::= ... integer numbers ...
<character> ::= ... all characters ...
<string> ::= <character> | <string> <character>

)

PROLOG

The code in will run a script from a given plain-text file using the predicate. For example, `run_program('input.txt', legs(X,2):-mammal(X), arms(X,2). , Result).` will attempt to execute the `main` function using the single argument, `2`, from the file called `input.txt`. Extending this example, suppose `input.txt` contained the following line of text:

(“legs” (“X” 2)) ((“mammal” (“X”)) (“arms” (“X” 2))))

First, the code in `tokenizer` reads this as whitespace-delimited text and create a list of tokens.

Tokenizes whitespace delimited ASCII strings from a file

“tokenize_file(FileName, Token_List)” :

Takes in a filename, returns the file contents as a list of tokens

“read_tokens_from_stream(Stream, [Token|Tokens_From_Stream])” :

Tokens are read from a stream and added to a list

“get_next_token(Stream, Token):

Reads characters from stream and unifies as a single token

“**get_next_char(Stream, [Ascii_Char|String], Ascii_Char)**

Retrieves the next character from the stream unless it has an ASCII code

“clean_token_list([], [])” :

Removes empty tokens generated by consecutive whitespace delimiters

Second, the code in `lexer` uses the token list to create a lexed list of identified token types.

Assign labels to all of the tokens in the TokenList

Takes in a list of tokens and returns a list of lexemes (i.e., abstract units defining the meaning of the token)

Third, the code in `parser` uses the lexed list and formats it into a structured list based on the definite clause grammar predicates in `grammar`.

parse_list(LexedList, StructuredList):

Creates the StructuredList from the LexedList

clean_parsed_list(TokenList, StructuredList, ParsedList):

Simplifies the call for the clean_parsed_list/4 predicate which uses an extra variable for handling the token list

parse_token_list(TokenList, ParsedList)

Simplifies creating the parsed list by wrapping other predicates

program(FunctionList) :

xecutes functions from the symbol table based on the integer arguments provided

“functionListCollection(FunctionListCollection)” :

Checks whether the arguments provided match the expected input for the function and verifies that each value in the Arguments list has an integer value that is of the same type (bool or int) as the corresponding parameter type from the function data and adds it to the symbol table

function([TypeID, '(', TypeIDList, ')', '=', Expression])

Retrieves the function from the symbol table, executes it, and returns a result