Gebze Technical University
Department of Computer Engineering

CSE 341 – Homework 5 Report

(MIDTERM)

DİLARA KARAKAŞ

171044010

In the first part of the assignment, I encountered a problem such as reading from the file and listing it. The string-to-list function was used to solve this problem. It is assumed that the input file will come in the form of a list. It looks the same in the example. Therefore, an error may occur when another kind of input file is given.

```
(defun string-to-list (str)
        (if (not (streamp str))
           (string-to-list (make-string-input-stream str))
           (if (listen str)
              (cons (read str) (string-to-list str))
              nil)))
```

Another trim-list function has been written to help read input file. While reading, it makes correction by checking the "(" and ")" characters.

```
(defun trim-list (list)
    (setq newList '())
    (dolist (line list)
        (setq resultString "")
        (loop for ch across line
            do
                (cond
                    ((equal (string ch) "(") (setq resultString (concatenate 'string resultString (string "( "))))
                    ((equal (string ch) ")") (setq resultString (concatenate 'string resultString (string " )"))))
                    (t (setq resultString (concatenate 'string resultString (string ch))))
                )
        )
        (setq newList (append newList (list resultString)))
    )
    newList
)
```

Then the "input.txt" file in the same directory is started to be read. Reading is in the form of a string. Reading has been done using the open function. Line by line reading is done. However, if the list in the file consists of one line, no problem will occur. It doesn't matter how many lines the file consists of. If the size of the read line is greater than 2, action is taken. This is to exclude those in the first or last line of the sample input file such as "(".

```
(setq line2 "")
(setq full_ '())
(let ((in (open "input.txt" :if-does-not-exist nil)))
    (when in
        (loop for line = (read-line in nil)
            while line do
            (progn
                (setq line2 line)
                (setq list_ '())
                (setq list_ (append list_ (list line)))
                (setq read_string (nth 0 (trim-list list_)))
                (cond
                    ((> (length read_string) 2)
                        (progn
                            (setq list_1 (string-to-list read_string))
                            (setf full_ (append full_ list_1))
                        )
                    )
                )

            )
        )
        ;;(print full_)
        (close in)
    )
)
```

Three new lists are created after the file has been successfully read. These lists are created for fact, horn clause and quyery. There is a loop here. Loops as many as the number of elements of the list read. This number shows how much they have already been found. When the example in the PDF is examined, it is understood that if the first element of the list is empty, it is a query, and if the second element is empty, it is a fact and otherwise a horn clause. The shredding process was made according to these features. Then these three lists are printed on the terminal in order.

```
(setq horn_clause_list '())
(setq fact_list '())
(setq query_list '())
(setq control 0)
(setq new_full '())
(setq new_full full_)

(print "READ FILE : ")
(print new_full)
(terpri)
(terpri)
(dolist (i new_full)
    (if (>= (length i) 2)
        (progn
            (if (equal (nth 1 i) nil)
                (progn
                    (setq fact_list (append fact_list (list (nth 0 i))))
                    (setq control 1)
                )
            )
            (if (equal (nth 0 i) nil)
                (progn
                    (setq query_list (append query_list (list (nth 1 i))))
                    (setq control 1)
                )
            )
            (if (equal control 0)
                (progn
                    (setq horn_clause_list (append horn_clause_list (list i)))
                )
            )
            (setq control 0)
        )
    )
)
```

The go_horn_clause function is the function that finds horn clauses. This function takes a horn clause as a parameter. Using the first element of this horn clause, a comparison is made in the horn_clause list. If the first element of the relevant horn clause is in the list, a comparison is made with other values (facts). If they are the same, type check is done this time. For example, "X" or "horse" should not cause a problem because they are the same type. This check has been done here. In addition, two lists were kept. These lists are first_list and second_list. The purpose of their use is that one list holds "X", while the other list keeps the value of this variable, if any, so the other list's element in the same index is "horse". Then, according to these lists, the value of the variable, if any, is replaced. So instead of "X", "horse" is written. Of course, if these values are really met, this is done.

```
(defun go_horn_clause(query)
    (setq return_list '())
    (setq first_list '())
    (setq second_list '())
    (dolist (element horn_clause_list)
       (if (equal (nth 0 (nth 0 element)) (nth 0 query))
           (progn
               (setq new_list (nth 1 (nth 0 element)))
               (if (equal (length new_list) (length (nth 1 query)))
                   (progn
                       (dotimes (i (length new_list))
                           (if (equal (type-of (nth i new_list)) (type-of (nth i (nth 1 query))))
                               (progn
                                   (if (not (equal (type-of (nth i new_list)) (type-of (string "ABC"))))
                                       (progn
                                           (if (equal (nth i new_list) (nth i (nth 1 query)))
                                               (progn
                                                   (setq return_list (append return_list (list element)))
                                               )
                                           )
                                       )
                                       (progn
                                           (setq first_list (append first_list (list (nth i new_list))))
                                           (setq second_list (append second_list (list (nth i (nth 1 query)))))
                                       )
                                   )
                               )
                           )
                       )
                   )
               )
           )
       )
    )

    (setf first_list (remove-duplicates first_list :test #'equal))
    (setf second_list (delete-duplicates second_list :test #'equal))
```

```
(setf first_list (remove-duplicates first_list :test #'equal))
(setf second_list (delete-duplicates second_list :test #'equal))

(dolist (list_x return_list)
    (setf predic_list (nth 1 list_x))
    (dolist (small_list predic_list)
        (setf new_small_list (nth 1 small_list))
        (dotimes (i (length new_small_list))
            (if (not (equal (position (nth i new_small_list) first_list :test #'equal) nil))
                (progn
                    (setq index (position (nth i new_small_list) first_list :test #'equal))
                    (if (not (equal (nth index second_list) nil))
                        (setf (nth i new_small_list) (nth index second_list))
                    )
                )
            )
        )
    )
)
return_list
```

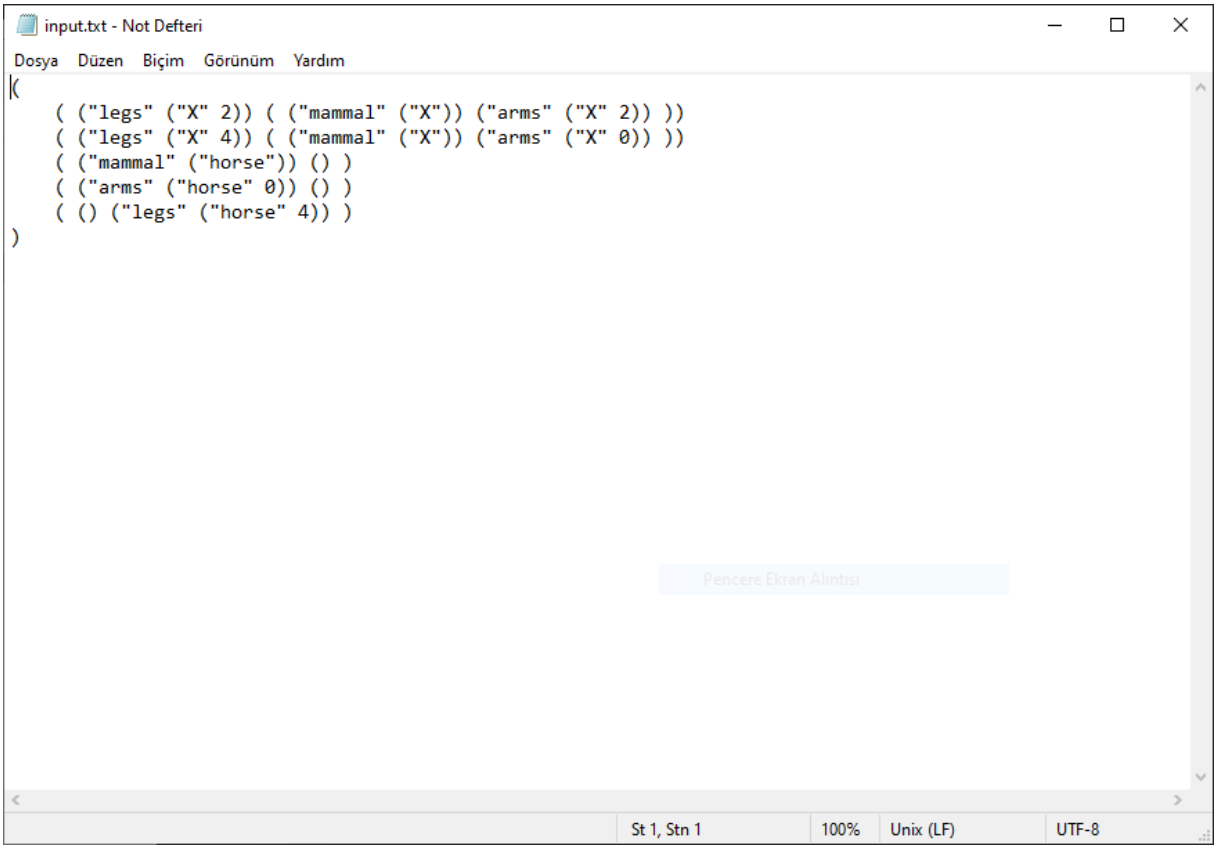
Then there is a running loop for each query. The more queries there are, the more they return. And for each query, go_horn_clause function is called. And the value returned from here is the result of either the nil or the horn clause. The results returned are in the form of a list. And it is checked whether the elements in this list are in the fact list. If they have all, this query is satisfied. If there isn't even one, it's not met. That's why nil is written to the file. If it is met, true is written to the file. Writing to the file takes place according to the query order.

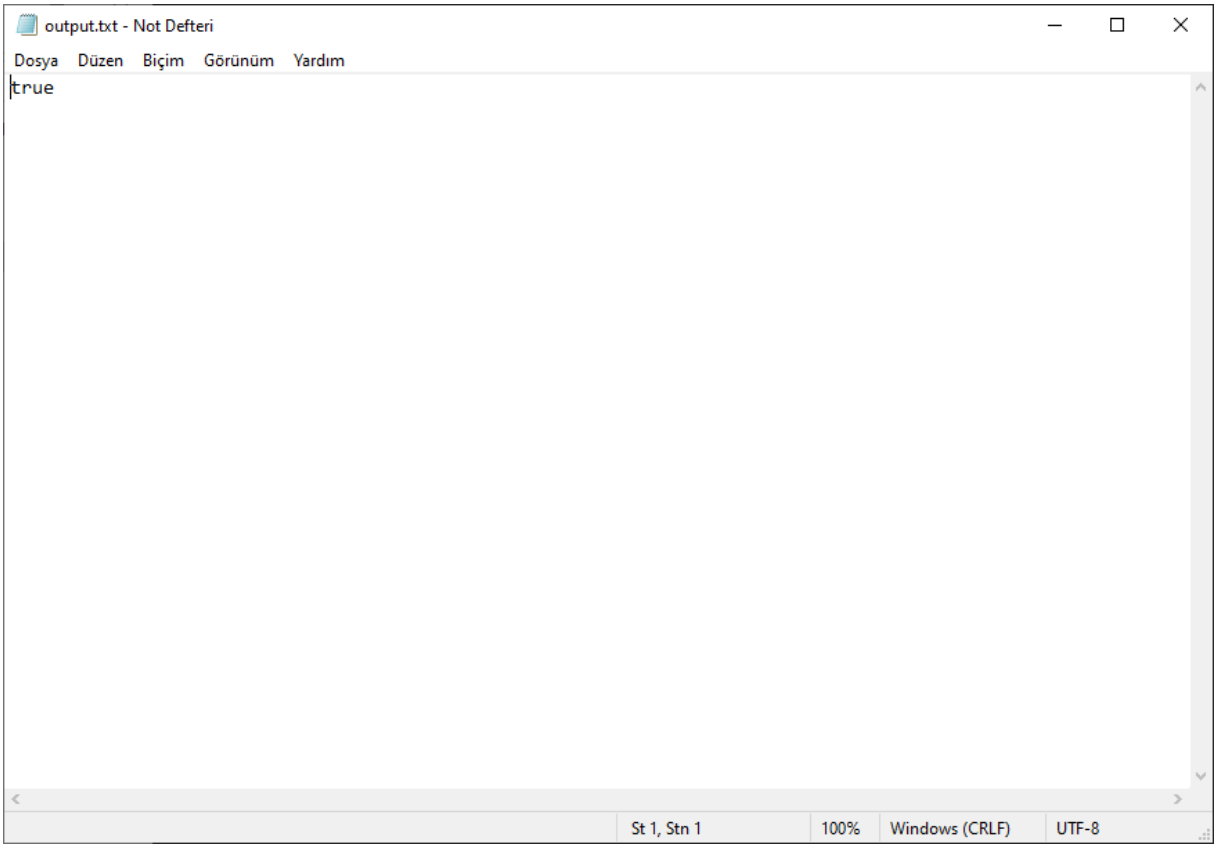
```
(with-open-file (stream "output.txt" :direction :output )
    (dolist (querys query_list)
        (setq resultList (go_horn_clause querys))
        (setq ifTheNewQueryList (nth 1 (nth 0 resultList)))

        (setq controlIFTrue 0)
        (dolist (first_element ifTheNewQueryList)
            (dolist (second_element fact_list)
                (if (equal first_element second_element)
                    (setq controlIFTrue (+ controlIFTrue 1))
                )
            )
        )

        (setq result_query "nil")
        (if (equal controlIFTrue (length ifTheNewQueryList))
            (setq result_query "true")
        )

        (format stream result_query)
        (terpri stream)
    )
    (close stream)
)
```


input.txt - Not Defteri
Dosya Düzen Biçim Görünüm Yardım
(
(("legs" ("X" 2)) (("mammal" ("X")) ("arms" ("X" 2))))
(("legs" ("X" 4)) (("mammal" ("X")) ("arms" ("X" 0))))
(("mammal" ("horse")) ())
(("arms" ("horse" 0)) ())
(() ("legs" ("horse" 4)))
)
St 1, Stn 1
100%
Unix (LF)
UTF-8


output.txt - Not Defteri
Dosya Düzen Biçim Görünüm Yardım
true
St 1, Stn 1
100%
Windows (CRLF)
UTF-8