CSCI 4430 | Programming Languages 2024 | PA3

Team Creation Due: Nov 22, 2024| Due Date: Dec 04, 2024

---

1 Parsing SQL-like Natural Language Commands

The first part of this assignment involves parsing SQL-like natural language commands using a Definite Clause Grammar (DCG). You will take two input files: one containing SQL-like commands written in natural language and the other representing a knowledge base. You are provided with a grammar that defines the SQL-like commands. Logical operators include an operation that requires logical reasoning over the knowledge base. Note that you will not receive any test cases that deviate from this grammar.

Grammar[1]

<**command**>::= Get <**table_column_info**> <**command_operation**>.

<**table_column_info**>::= <**table_column_detail**> | (<**table_column_detail**> and <**table_column_info**>)

<**table_column_detail**>::= (all from <**table**>) | (<**columns**> from <**table**>)

<**command_operation**>::= [<**join_operation**> | <**match_operation**> | <**where_operation**>]

<**join_operation**> ::= (linking | connecting) <**table**> by their <**col**>

<**match_operation**> ::= such that <**match_condition**>

<**match_condition**>::= its values are either <**values**>

<**match_condition**>::= <**col**> matches values within the <**col**> in <**table**> <**where_operation**>

<**where_operation**>::= where <**or_condition**> {and <**or_condition**>}

<**or_condition**>::= <**condition**> | (either <**condition**> or <**condition**>)

<**condition**>::= <**col**> <**equality**> <**val**>

<**equality**>::= (is less than) | (is greater than) | equals

<**table**>::= <**atom**>[2]

<**columns**>::= <**col**>{ (, | and) <**col**> }

<**col**>::= <**atom**>[3]

<**values**>::= <**val**>{ (, | or) <**val**> }

<**val**>::= <**atom**>[4]

---

[1] This grammar utilizes Extended Backus–Naur notations. The Wikipedia page for **E-BNF** enlists all the necessary notation. Specially, these notation: (),{}, [] and | .

[2] These values are directly from the *facts.pl*. It is something that binds to the `TableName` in `table(TableName,[_])` from the *facts.pl*.

[3] Same as 1. However, this includes individual elements within the list of column headers that bind to `Column_i` in `table(TableName,[Column_1,Column_2,...,Column_N])`.

[4] Same as 1 and 2. However, this includes individual cell values within the list of row facts that bind to `CellValue_i` in `row(TableName,[CellValue_1,CellValue_2,...,CellValue_N])`.

Objective:

- Interpret the Command: For each natural language command, you will utilize DCG to parse the command and return the parsed command.
  - `nlp_parse(LineSplit, Query)`: Write a rule to parse the command `LineSplit`, a list of strings split into different words and punctuation, into a specific list of queries. You will later need to implement the `Query` to execute the logical reasoning and constraint satisfaction on the facts provided in *facts.pl*.
- After parsing all of the provided commands, you will need to use `writeln(Query)` to display the parsed text[5]. They need to be in the following format based on the provided grammar:

  [**command**,variables for the <**table_column_info**>,variables for the <**command_operation**>]

  Variables for <**table_column_info**> must be in the following order:
  - [[**all**, `TableName`]]
  - [[variables for < **columns** >, `TableName`], ...]
  - [[variables for < **columns** >, `TableName1`],[variables for < **columns** >, `TableName2`],...]
  - [[**all**, `TableName`], [variables for < **columns** >, `TableName`]]

  Variables for <**columns**> must be in the following order:
  - [< **column** >]
  - [< **column** >, < **column** >, ...]

  Variables for <**command_operation**> must be in the following order:
  - [ ]
  - [**join**, `TableName`, `ColumnName`]
  - [**matches**, [< **value** >, < **value** >, < **value** >, ...]]
  - [**matches**, `ColumnName`, [**command**, [[`ColumnName`, `TableName`]], [**where**, [variables for < **where_operation** >]]]]
  - [**where**, [variables for < **where_operation** >]]

  Variables for <**where_operation**> must be in the following order:
  - [variables for <**condition**>]
  - [**and**, [variables for <**where_operation**>], [variables for <**where_operation**>]][6]
  - [**or**, [variables for <**condition**>], [variables for <**condition**>]]

  Variables for <**condition**> must be in the following order:
  - [**condition**, `ColumnName`, <**equality**>, `Value`]

  <**equality**> must be one of the following:
  - <, =, or, >

[5] It has been included within the starter code.

[6] Parse it as a right associativity for multiple **and**. That is, for `... where c1 and c2 and c3 and c4.`, it should be: [**and** `c1` [**and** `c2` [**and** `c3` `c4`]]]

Expected Output:

This determines how well you parse the file. You need to print out the parsed text document by printing different variables required to solve Part 2. **For consistency, you must have the exact ordering match for the output.**

The grade for part one will be determined by your rules' ability to parse the *example.txt* file.

Example of commands within *example.txt*:

1. Get all from employee.
2. Get ProductName from product and Price from product.
3. Get CustomerID from order linking customer by their CustomerID.
4. Get LastName from customer such that its values are either MariaAnders, Fuller or Suyama.
5. Get OrderDate from order such that CustomerID matches values within the CustomerID in customer where Country equals UK.
6. Get all from employee and FirstName from employee where BirthDate is less than 12-16-1965.
7. Get ProductName, Price, CategoryID and SupplierID from product where Price is greater than 25 and CategoryID is greater than 4.
8. Get ProductName, Price, CategoryID and SupplierID from product where Price is greater than 25 and either CategoryID is greater than 4 or SupplierID is less than 3.
9. Get OrderID from order where either OrderDate is greater than 11-22-1996 or ShipperID equals 2.

Expected Output[7]:

1. [**command**, [[**all**, `employee`]], []]
2. [**command**, [[[`ProductName`], `product`], [[`Price`], `product`]], []]
3. [**command**, [[[`CustomerID`], `order`]], [**join**, `customer`, `CustomerID`]]
4. [**command**, [[[`LastName`], `customer`]], [**matches**, [`MariaAnders`, `Fuller`, `Suyama`]]]
5. [**command**, [[[`OrderDate`], `order`]], [**matches**, `CustomerID`, [**command**, [[`CustomerID`, `customer`]], [**where**, [**condition**, `Country`, =, `UK`]]]]]
6. [**command**, [[**all**, `employee`], [[`FirstName`], `employee`]], [**where**, [**condition**, `BirthDate`, <, `12-16-1965`]]]
7. [**command**, [[[`ProductName`, `Price`, `CategoryID`, `SupplierID`], `product`]], [**where**, [**and**, [**condition**, `Price`, > , 25], [**condition**, `CategoryID`, >, 4]]]]
8. [**command**, [[[`ProductName`, `Price`, `CategoryID`, `SupplierID`], `product`]], [**where**, [**and**, [**condition**, `Price`, > , 25], [**or**, [**condition**, `CategoryID`, >, 4], [**condition**, `SupplierID`, <, 3]]]]]
9. [**command**, [[[`OrderID`], `order`]], [**where**, [**or**, [**condition**, `OrderDate`, >, `11-22-1996`], [**condition**, `ShipperID`, =, 2]]]]

[7] The exact ordering must match. Use `writeln()` instead of `print()`.

2 Logical and Constraint Reasoning with Logical Operators

In this part, you will focus on logical reasoning using SQL-like operators parsed from Part 1. The goal is to evaluate queries that involve solving commands. The output from the part 1 should result in two parts: columns required for output and conditions that filter the commands. You are expected to output individual tables with all the columns within <**table_column_info**> from the provided *facts.pl* in that particular order. However, if there are any conditions within <**command_operation**>, you are expected to return the subset based on the operation.

Objective:

- `evaluate_logical(Query, FilteredTable)`.: Executes logical reasoning based on the parsed query. The queries are going to be the < **where_operation** >, as mentioned in part 1. Given that

  `Query` is `[command, TableColumnInfo,Conditions]`,

  a parsed individual command from Part 1. You are expected to bind `FilteredTable` with the list of filtered tables for each table information within `TableColumnInfo`.

  `FilteredTable` is [[`TableName`, `TableColumnHeader`, `FilteredRows`], ...],

  where `TableName` is the name of the table, `TableColumnHeader` is a list of column headers, `FilteredRows` is a list of filtered row values that are associated based on the selected `TableColumnHeader`.

  Since, `FilteredTable` is dependent upon the `TableColumnHeader`(<**table_column_info**>) within the `Query`. We will expect the `FilteredTable` to be in the following format:

  - [**all**, `TableName`]: returns a list with the table name, a list of all the column headers of the table, and a nested list of all filtered rows within the `TableName` corresponding to the list of columns.

    [[`TableName`, `AllColumnHeader`, `FilteredRows`]]

  - [`ColumnNames`, `TableName`]: returns a list with the table name, a list with the column names, and a nested list of all filtered rows within the `TableName` corresponding to the columns.

    [[`TableName`, [`ColumnName1`, `ColumnName2`, `ColumnName3`, ..], `FilteredRows`]]

  - [`ColumnNames1`, `TableName1`],[`ColumnNames2`, `TableName2`],...: returns a multiple list corresponding to each table information with the name of the table (`TableName1`,`TableName2`,...), a list of all column headers for each table, and a nested list of all filtered rows within the `TableName` corresponding to the list of columns.

    [[`TableName1`, `ColumnNames1`, `FilteredRows1`], [`TableName2`, `ColumnNames2`, `FilteredRows2`], ...]

  For `Condition` (<**command_operation**>), you will utilize the table information from <**table_column_info**> to filter the information based on the condition type. You will utilize the `TableName` from `TableColumnInfo` and column information from `Condition` to select the subset of rows to filter the data. For comparisons, you are to compare equals with string, number, and date type, and greater than and lesser than with number and date type.

  - [ ] : Do nothing. Return all rows within `FilteredRows`.
  - [**join**, []]|[**matches**, [_]] : Return no rows within `FilteredRows`.
  - [**where**, [conditions]] : Filters based on each of the conditions and bind all satisfied rows within `FilteredRows`.

Notes

1. We are only performing **where_operation** for the reasoning and constraint based filtering.
2. The grammar from part 1 could result in cases where the datatype for command operation will not make sense. Example: `Get ProductName from product where ProductName is less than AniseedSyrup.` You are expected to return an empty list of rows since the condition cannot be satisfied.
3. Data type conversion rule(`is_date(InputString, Date)`) for checking and converting date format into date predicate is provided within *helper.pl*. It converts a string with a specific date-formatted string(MM/M-DD/D-YYYY) into its date predicate data type(`date(Year, Month, Day)`). Do take note of the date type provided within the helper predicate. You may wish to use one or decide not to use it.
4. `print_tables(FilteredTable)` is also provided to display the filtered table. We will be utilizing this rule to display the result for grading. A single tab separates the tables; hence, it could appear uneven.

Example:[8]

1. Get all from employee.
   `FilteredTable=`

   ```
   [[employee,[EmployeeID,LastName,FirstName,BirthDate,Photo],[[1,Davolio,Nancy,
   12-8-1968,EmpID1.pic],[2,Fuller,Andrew,2-19-1952,EmpID2.pic],
   [3,Leverling,Janet,8-30-1963,EmpID3.pic],[4,Peacock,Margaret,9-19-1958,
   EmpID4.pic],[5,Buchanan,Steven,3-4-1955,EmpID5.pic],[6,Suyama,Michael,7-2-1963,
   EmpID6.pic],[7,King,Robert,5-29-1960,EmpID7.pic],[8,Callahan,Laura,1-9-1958,
   EmpID8.pic],[9,Dodsworth,Anne,7-2-1969,EmpID9.pic],[10,West,Adam,9-19-1928,EmpID10.pic]]]]
   ```

   Expected Output:

   ```
   Get all from employee.

               employee
   --------------------------------
   EmployeeID      LastName        FirstName       BirthDate       Photo
   --------------------------------
   1       Davolio Nancy   12-8-1968       EmpID1.pic
   2       Fuller  Andrew  2-19-1952       EmpID2.pic
   3       Leverling       Janet   8-30-1963       EmpID3.pic
   4       Peacock Margaret        9-19-1958       EmpID4.pic
   5       Buchanan        Steven  3-4-1955        EmpID5.pic
   6       Suyama  Michael 7-2-1963        EmpID6.pic
   7       King    Robert  5-29-1960       EmpID7.pic
   8       Callahan        Laura   1-9-1958        EmpID8.pic
   9       Dodsworth       Anne    7-2-1969        EmpID9.pic
   10      West    Adam    9-19-1928       EmpID10.pic
   --------------------------------
   ```

[8] More examples are provided with the starter file.

2. Get CustomerID from order linking customer by their CustomerID.
   FilteredTable=

   ```
   [[order,[CustomerID],[]]]
   ```

   Expected Output:

   ```
   Get CustomerID from order linking customer by their CustomerID.
              order
   --------------------------------
   CustomerID
   --------------------------------
   --------------------------------
   ```

3. Get all from employee and FirstName from employee where BirthDate is less than 12-16-1965.
   FilteredTable=

   ```
   [[employee,[EmployeeID,LastName,FirstName,BirthDate,Photo],[[2,Fuller,Andrew,
   2-19-1952,EmpID2.pic],[3,Leverling,Janet,8-30-1963,EmpID3.pic],[4,Peacock,
   Margaret,9-19-1958,EmpID4.pic],[5,Buchanan,Steven,3-4-1955,EmpID5.pic],[6,
   Suyama,Michael,7-2-1963,EmpID6.pic],[7,King,Robert,5-29-1960,EmpID7.pic],[8,
   Callahan,Laura,1-9-1958,EmpID8.pic],[10,West,Adam,9-19-1928,EmpID10.pic]]],
   [employee,[FirstName],[[Andrew],[Janet],[Margaret],[Steven],[Michael],
   [Robert],[Laura],[Adam]]]]
   ```

   Expected Output:

   ```
   Get all from employee and FirstName from employee where BirthDate is less than 12-16-1965.
              employee
   --------------------------------
   EmployeeID      LastName        FirstName       BirthDate       Photo
   --------------------------------
   2       Fuller  Andrew  2-19-1952       EmpID2.pic
   3       Leverling       Janet   8-30-1963       EmpID3.pic
   4       Peacock Margaret        9-19-1958       EmpID4.pic
   5       Buchanan        Steven  3-4-1955        EmpID5.pic
   6       Suyama  Michael 7-2-1963        EmpID6.pic
   7       King    Robert  5-29-1960       EmpID7.pic
   8       Callahan        Laura   1-9-1958        EmpID8.pic
   10      West    Adam    9-19-1928       EmpID10.pic
   --------------------------------
              employee
   --------------------------------
   FirstName
   --------------------------------
   Andrew
   Janet
   Margaret
   Steven
   Michael
   Robert
   Laura
   Adam
   --------------------------------
   ```

4. Get ProductName, Price, CategoryID and SupplierID from product where Price is greater than 25 and either CategoryID is greater than 4 or SupplierID is less than 3.
   FilteredTable=

   ```
   [[product,[ProductName,Price,CategoryID,SupplierID],[[UncleBobOrganicDriedPears,
   30,7,3],[MishiKobeKiku,97,6,4],[Ikura,31,8,4],[AliceMutton,39,6,7],[FooBar,26,2,1]]]]
   ```

   Expected Output:

   ```
   Get ProductName, Price, CategoryID and SupplierID from product where Price is
   greater than 25 and either CategoryID is greater than 4 or SupplierID is less than 3.
               product
   --------------------------------
   ProductName Price CategoryID SupplierID
   --------------------------------
   UncleBobOrganicDriedPears 30 7 3
   MishiKobeKiku 97 6 4
   Ikura 31 8 4
   AliceMutton 39 6 7
   FooBar 26 2 1
   --------------------------------
   ```

Provided Files:

- *main.pl*: The main Prolog file that you include your code. It also includes import for the *helper.pl* and *facts.pl* file. It has been set such that if you correctly write `nlp_parse(LineSplit,Query)` and `evaluate_logical(Query,FilteredTable)`, you would get the expected output as provided.
- *helper.pl*: It contains some starter helper predicates for parsing the input file. It also includes the date data type conversion helper predicates as well as `print_tables(FilteredTable)`, which prints the list of `FilteredTable` into a tabular display. **Do not make any changes to the helper predicates; any modification will be ignored. You may choose not to use the provided predicates, but include any new predicates within *main.pl*.**
- *example.txt*: A text file that contains SQL-like natural language commands which are passed as an argument.
- *expectedoutput_part1.txt*: Expected output with `part1` argument after parsing *example.txt* file.
- *expectedoutput_part2_FilteredTable.txt*: It contains nested list of table information within `FilteredTable`, which is required as an argument for `print_tables(FilteredTable)`. **Note that this is not an expected output. It is provided to you to understand the required format for an actual output.**
- *expectedoutput_part2.txt*: Actual expected output with `part2` argument after performing constraint reasoning with the *example.txt* file.
- *facts.pl*[9]: Represents the table data in Prolog facts format. Note that to avoid any discrepancy while parsing the table and commands, we have no spaces in between any strings in the *facts.pl*.

How to Run in Prolog

A file, `main.pl`, is provided that will read in a file with **definitions** and **constraints** until encountering the end of the file (EOF). Note that `main.pl` imports the file `helper.pl`, which is also provided in the PA3 web directory. To test this file, run the following:

```
swipl -s main.pl -t main --quiet -- example.txt part1 > expectedoutput_part1.txt
swipl -s main.pl -t main --quiet -- example.txt part2 > expectedoutput_part2.txt
```

The command contains two arguments file name that contains the command and print option argument. The print option argument(`part1` or `part2`) is only differentiated by the output statement for testing purposes. **Note:** You are allowed to modify the provided starter file. But make sure all the changes are within the main.pl such that your assignment compiles correctly.

What To Submit & Grading

Please ensure you have registered for a team by **Nov 22, 2024**.

Please submit your completed version of `main.pl`. Do **not** submit any other provided files (`helper.pl`, `facts.pl`.

You **must** also submit a `readme.txt` file with your name (or names if you have a partner). If you have any comments, reports of bugs, self-assessments, or anything, put them in your `readme`.

[9]Hidden tests will contain a different facts.pl file. Hence, do not **hardcode** any of the facts.

Warning About Posting Solutions Publicly & Inter-Team Code Sharing

Do not post any code you submit for this assignment onto any public website or network share. Also, your code from your team (which can be at most two people) should be wholly yours; Do not look at or use code made by other students and/or teams as your own. If a violation of either of these rules is detected, your grade will be heavily affected.

Links & Documentation for (SWI) Prolog

- Quickstart documentation:
  - `https://www.swi-prolog.org/pldoc/man?section=quickstart`
- Installers for version 8.4 (Windows/MAC):
  - `https://www.swi-prolog.org/download/stable` (for Linux, check your package manager for `swi-prolog`)
- Definite Clause Grammar documentation:
  - `https://www.swi-prolog.org/pldoc/man?section=DCG`