UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

NLP: Lab 3

Definite Clause Grammars

Alejandro Hernandez, Javier Fernandez | Natural Language Processing, MAI | January 15th 2014

Table of Contents

Introduction

The present report describes in detail the procedure of defining and implementing definite clause grammars (DCG) to match dates. The previous work was focused on designing finite state automata (FSA) to match the dates. In this work it will be presented how DCG allow to parse the desired data with a more rich and simpler vocabulary, and perform considerably faster.

Running the project

The code was developed in Prolog logic language and tested using SWI-Prolog. To run the project is required to have SWI-Prolog installed.

From the command line the file can be loaded as following (starting on the root folder):

$$swipl\ -s\ dateparser.pl$$

Or otherwise starting SWI-Prolog and loading the file directly:

$$swipl$$

$$[dateparser].$$

Notice that depending on the operative system where the program is being runned the command to start SWI-Prolog might be different.

To execute the parser on the *examples.txt* file do:

$$main.$$

This will read the *examples.txt* file, and apply the *fmatchparser DCG* to determine if each word of the file matches with a valid date. Additionally all the words that were not matched by the parser are printed in the screen. The last three lines contain: the total number of examples, the number of matched words and the percentage of the words being correctly matched as a date.

Parts of the DCG

DCG's in Prolog allow to match specific range of characters of the words to the different elements of the definite clauses. The low level implementation of DCG's in Prolog is based on difference, which makes this possible.

In particular it is known that a definite clause such as:

```
sentence --> noun_phrase, verb_phrase.
```

Is translated into an expression of the form:

```
sentence(S1,S3) :- noun_phrase(S1,S2), verb_phrase(S2,S3).
```

Where each argument is a difference list.

According to this representation the different lines in the text (referenced as *words* in this report) are translated into a list where each element is the ASCII character contained in the line. Given this representation sequences of letters (specific information to match) are defined as:

```
c --> "example"
```

Where the sequence of characters are matched as:

```
[101,120,97,109,112,108,101]
```

This conforms the background information required to understand the proposed representation of grammar to solve this problem.

Regular Expressions

Similarly as done before with the implementation of FSA for matching dates, a set of operators used commonly when defining regular expressions were defined directly as DCG in order to easy the representation of the more complex grammars designed to match each type of date.

In particular the defined regular expression operators are:

(+) One or more occurrences

```
+(E) --> E.

+(E) --> E,+(E).
```

(?) One or none occurrences

```
?(_) --> [].

?(E) --> E.
```

(+) None or many occurrences

```
*(_) --> [].

*(E) --> E, *(E).
```

(+) Exactly one occurrence

```
.(E) --> E.
```

For using the operators directly without the parenthesis the built in definition of this operators for Prolog where overwritten as:

```
:- op(100,xf,+).

:- op(100, xf, ?).

:- op(100, xf, *).

:- op(100, xf, .).
```

Basic Support Grammars

To ease the representation of the whole grammar for matching dates, a set of clauses where defined that allow to match inner parts of the complete grammar. This clauses are used as building blocks for building the final grammar.

Birth date

```
birthdate --> ("birth_date" | "birth date").
```

Months

```
month_letters --> ("January" | "january" | "February" | "february" | "March" | "march" |

"April" | "april" | "May" | "may" | "June" | "june" | "July" | "july" | "August" | "august" |
"September" | "september" | "October" | "october" | "November" | "november" | "December" |
"december").

month_letters_short --> ("Jan" | "Feb" | "Mar" | "Apr" | "May" | "Jun" | "Jul" | "Aug" | "Sep" |
```

"Oct" | "Nov" | "Dec").

month_num --> digit.

month_num --> "0",digit_non0;"10";"11";"12".

So the full representation of possible months is:

month --> month_num;month_letters;month_letters_short.

Digit

A simple representation of digits was implemented to ease the match with digit transformed characters.

digit_non0 --> "1";"2";"3";"4";"5";"6";"7";"8";"9".

digit --> "0";digit_non0.

Day

day --> digit.

day --> "0",digit;"1",digit;"2",digit;"30";"31".

week_day --> "Monday";"monday";"Tuesday";"tuesday";"Wednesday";"wednesday";"Thursday";"thursday";"Friday";"friday";"Saturday";"saturday";"Sunday";"sunday".

Notice that *week_day* is not matched directly with as *day*. This is because the occurrences of *day* are considerable more often than *week_day* so a clause as *day;week_day* would produce a too wide matching parser (and thus a not so good one).

Spacing Delimiters

space --> " ".

tab --> "\t".

comma --> ",".

Special Characters

o_brackets --> "[[".

c_brackets --> "]]".

bar --> "|".

end_trash --> space;".".

dash --> "-".

comma -->",".

dot --> ".".

cyears --> "c.".

unknown --> "unknown";"Unknown".

ortext --> "or".

day_termination --> "th";"st";"nd";"rd".

trash --> letters;dash;"<";"!";">";"(";")";"|";dot;comma;space;digit.

pretext --> "about";"Date?";"born";cyears;"ca".

Date Matching Grammar

Following the representation proposed with FSA previous project, the set of the four most common patterns were firstly defined to reach an accuracy near-identical that such of the FSA ~94%.

For this clauses some interesting generalizations are performed by including the previously defined regular-expression-like operators on certain elements.

The name of the procedure used for matching the dates is: *matchphrase*

Matching Month, Day, Year

Matches phrases such as:

- birth_date [[March 25]], [[1968]]
- birth_date March 25, [[1968]]
- birth_date March 25, 1968

matchphrase --> birthdate,tab,?(o_brackets),month,space,day,?(c_brackets),?(comma),?(space),?(o_brackets),year,?(c_brackets).

Matching Day, Month, Year

Matches phrases such as:

- birth_date 18 May, 1959
- birth_date [[18 May]], [[1959]]
- birth_date 18 May, [[1959]]

matchphrase --> birthdate,tab,?(o_brackets),day,space,month,?(c_brackets),?(comma),?(space),?(o_brackets),year,?(c_brackets).

Matching Year

Matches phrases such as:

- birth_date [[1943]]
- birth_date 1221
- birth_date c. [[350]]

matchphrase --> birthdate,tab,?(pretext),?(space),?(o_brackets),year,?(c_brackets).

Matching Year, Month, Day

Matches phrases such as:

- birth_date 1942|6|10
- birth_date [[1942|6|10]]

matchphrase --> birthdate,tab,?(o_brackets),year,bar,month,bar,day,?(c_brackets),?(end_trash).

Special Clauses

For achieving a higher accuracy and taking advantage of the power and ease of representation provided by DCG's a set of additional clauses were added that matches specific and less common cases of the *examples.txt.* file.

Special Case 1

Matches phrases such as:

- birth_date [[October]] [[1977]]

matchphrase --> birthdate,tab,o_brackets,month,c_brackets,space,o_brackets,year,c_brackets.

Special Case 2

Matches phrases such as:

- birth_date [[1901-09-27]]
- birth_date [[1901-Sep-27]]
- birth_date [[1901-September-27]]

matchphrase --> birthdate,tab,o_brackets,year,dash,month,dash,day,c_brackets.

Special Case 3

Matches phrases such as:

- birth_date [[November 21]], [[1968]]
- birth_date [[November 21]], [[1968]]
- birth_date [[November 21]],[[1968]]
- birth_date [[Nov 21]], [[1968]]
- birth_date [[February 4th]], [[1923]]
- birth_date [[13 November]], [[1943]]

matchphrase --> birthdate,tab,o_brackets,month,space,?(space),day,?(day_termination),?(space),c_brackets,comma,?(space),o_brackets,year,c_brackets.

matchphrase --> birthdate,tab,o_brackets,day,?(day_termination),space,?(space),month,?(space),c_brackets,comma,?(space),o_brackets,year,c_brackets.

Special Case 4

Matches phrases such as:

- birth_date Monday, [[December 19]], [[1955]]
- birth_date Monday, [[19 December]], [[1955]]

matchphrase --> birthdate,tab,?(o_brackets),week_day,?(c_brackets),?(comma),space,o_brackets,month,space,?(space),day,?(space),c_brackets,?(comma),?(space),o_brackets,year,c_brackets.

matchphrase --> birthdate,tab,?(o_brackets),week_day,?(c_brackets),?(comma),space,o_brackets,day,space,?(space),month,?(space),c_brackets,?(comma),?(space),o_brackets,year,c_brackets.

Special Case 5

Matches phrases such as:

- birth_date Unknown
- birth_date unknown

matchphrase --> birthdate,tab,unknown.

Special Case 6

Matches phrases such as:

- birth_date 1302 or 1303
- birth_date [[626]] or [[627]]
- birth_date [[February 2]], [[1650]] or [[1651]]

matchphrase --> birthdate,tab,?(o_brackets),year,?(c_brackets),space,ortext,space,?(o_brackets),year,?(c_brackets).

matchphrase --> birthdate,tab,?(o_brackets),?(month),?(space),?(day),?(c_brackets),?(o_brackets),year,?(c_brackets),space,ortext,space,?(o_brackets),year,?(c_brackets).

Special Case 7

Matches phrases such as:

- birth_date July, [[1980]]

matchphrase --> birthdate,tab,month,comma,space,o_brackets,year,c_brackets.

Special Case 8

Matches phrases such as:

- birth_date January [[1968]]

matchphrase --> birthdate,tab,month,space,o_brackets,year,c_brackets.

Special Case 9

Matches phrases such as:

- birth_date 26th April 1951
- birth_date 26th April, 1951

matchphrase --> birthdate,tab,day,day_termination,space,month,?(comma),space,year.

Special Case 10

Matches phrases such as:

- birth_date [[13 September]] , [[1973]]
- birth_date [[13 September]], [[1973]]
- birth_date [[13 September]] [[1973]]

matchphrase --> birthdate,tab,o_brackets,day,space,month,?(space),c_brackets,?(space),?(comma),?(space), o_brackets,year,c_brackets.

Special Case 11

Matches phrases such as:

- birth_date [[June]], [[1763]]
- birth_date June, 1763

matchphrase --> birthdate,tab,?(o_brackets),month,?(space),?(c_brackets),?(space),?(comma),?(space),?(o_brackets),year,?(c_brackets).

Special Case 12

Matches phrases such as:

- birth_date [[August 8]]
- birth_date October 1

matchphrase --> birthdate,tab,?(o_brackets),month,space,day,?(c_brackets).

Special Case 13

Matches phrases such as:

- birth_date [[April 14]],
- birth_date [[April 14]]

matchphrase --> birthdate,tab,?(o_brackets),month,space,day,?(c_brackets),?(comma).

Special Case 14

Matches phrases such as:

- birth_date [[july 18]], [[1949]]
- birth_date [[May 13]]. [[1928]]

matchphrase --> birthdate,tab,?(o_brackets),month,space,day,?(c_brackets),?(comma | dot),?(space),?(o_brackets),year,?(c_brackets).

Special Case 15

Matches phrases such as:

- birth_date [[September]] [[11]], [[1981]

```
matchphrase --> birthdate,tab,?(o_brackets),month,?(c_brackets),?(comma | dot),?(space),?(o_brackets),day,?(c_brackets),?(comma | dot),?(space),?(o_brackets),year,?(c_brackets).
```

Special Case 16

Matches phrases such as:

- birth_date 1942|6|10
- birth_date [[1942|6|10]]

```
matchphrase --> birthdate,tab,?(o_brackets),year,b
```

After date matching

Some date examples contain a valid date that matches one of the previous patterns but concatenated with a series of variable characters. To ensure a higher accuracy and avoid not matching the first part corresponding to a valid date, the *matchphrase* procedure was extended as:

```
fmatchphrase --> matchphrase,*(trash).
```

Where:

```
trash --> letters;dash;"<";"!";">";"(";")";"|";dot;comma;space;digit.
```

File parsing

The pio library was used to generate a line by line file parser:

```
:- use_module(library(pio)).

lines([]) --> call(eos),!.

lines([Line|Lines]) --> line(Line), lines(Lines).
```

```
eos([], []).


line([])  --> ( "\n" ; call(eos) ), !.

line([L|Ls]) --> [L], line(Ls).
```

So, the *main* rule creates a list with every line of the file and calls *match_dates* rules to attempt matching a date in each line.

```
main :- phrase_from_file(lines(Ls), './examples.txt'),
        match_dates(Ls,0,0).
```

match_dates allows to traverse the list by applying *parse* to each line (which internally tries to match the phrase defined by the line and the *fmatchphrase* clause) and accumulates the count of correctly matched dates and the total number of lines.

```
match_dates([H|T],Total,Accum):-
parse(H,Total,Accum,X,NewTotal),match_dates(T,NewTotal,X).

match_dates([],Total, X):- Y is X/Total, write(Total),nl, write(X),nl,write(Y)

parse(H,Total, Accum,X, NewTotal):-
        phrase(fmatchphrase,H), X is Accum+1, NewTotal is Total+1;
        \+ phrase(fmatchphrase,H), X is Accum, NewTotal is Total+1, writef(H), nl.
```

Results and Conclusions

The resulting grammar allowed to match the 97.40% of the words in *example.txt* file beating the FSA accuracy of 94.95%. Through the use of DFC the definition of a grammar and the most common patterns is a considerably shorter and efficient in performance task that using FSA for this purpose. Moreover, the addition of clauses to match very specific pattern is a straightforward procedure. These new clauses did not produce any considerable impact in the performance of the total matching which is fast.

From the 12.298 words in the test file, 11.979 could be correctly matched as dates giving just a rest of 319 patterns that require very specific clauses to be matched.

Although the test file does not contain examples that do not correspond to dates directly (with exception of some few) the defined grammar took is constrained to match specifically the date patterns and avoids to contain very general expressions that would match words belonging to patterns different than dates.