Practice lessons of Theory of automata and Formal languages

Prof. Karl Thurnhofer Hemsi
Computer Science and Computer Language Department
University of Málaga

October 12, 2021

Practice 1

LaTeX, Grammars and Regular Expressions

1.1 Set up

1. Open VirtualBox and choose the Virtual Machine (VM) named as 'TALF18-19_Ubuntu18'.

2. Within the VM, surf into the *Campus Virtual* and download the file "Practica1.tar.gz".

3. Decompress it using the command: `tar -xzvf Practica1.tar.gz`. Then, go into the folder "files".

4. Install the LaTeX editor called Gummi: `sudo apt install gummi`. If there are permission errors try to execute: `sudo fuser -vki /var/lib/dpkg/lock`

5. Install Git, the control version software of repositories: `sudo apt install git`. You may need to configure the proxy in the computer's lab of the UMA: `git config --global http.proxy http://proxy.lcc.uma.es:3128`

1.2 Introduction

This practice aims to introduce the grammars and regular expressions shown in theory. For that purpose, we are going to use GNU Octave, which is a high-level language, primarily intended for numerical computations. It provides a convenient command line interface for solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with Matlab. It may also be used as a batch-oriented language.

Octave has extensive tools for solving common numerical linear algebra problems, finding the roots of nonlinear equations, integrating ordinary functions, manipulating polynomials, and integrating ordinary differential and differential-algebraic equations. It is easily extensible and customizable via user-defined functions written in Octave's own language, or using dynamically loaded modules written in C++, C, Fortran, or other languages.

There exists an online version of this software (`https://octave-online.net/`), which allows users to use Octave to execute simple scripts like in the standalone version.

1. Open GNU Octave (which is already installed) from shell: `octave --no-gui`

2. Download or clone the repository by
   `git clone https://bitbucket.org/umafoss/tafluma.git`. Alternatively, you can go here and download the files contained in /software.

3. Open the help of the script unionrelation.m in GNU Octave (already installed) typing `help <script>` and run the script with one of the examples described.

1.3 Grammars in Octave

Here we are going to present a tool for defining and executing grammars using Octave.

1. Firstly, we need to specify how the script is going to read a grammar. Since we have a formal definition, we can use it to generate a plain text file as the example "g1.gra" present in the grammar folder.

2. Open the script leergramatica.m and check the help. Test it using the command `G = leergramatica('g1')` in the Octave console.

3. Now we are able to create strings of the language generated by this grammar by using the script producir.m. What output does it produce?

Related to this, we can also define a special type of grammars called L-systems. They were introduced and developed in 1968 by Aristid Lindenmayer, a Hungarian theoretical biologist and botanist at the University of Utrecht. The recursive nature of the L-system rules leads to self-similarity, so fractal-like forms are easy to describe. Plant models and natural-looking organic forms are easy to define, as by increasing the recursion level the form grows and becomes more complex.

1. Check the lsystems subfolder and open one of the JSON examples. Each of them define a different L-system.

2. Use the script "developlsystem.m" to execute one of the examples and see the output. What do they look like?

1.4 Writing in LaTeX

LaTeX is a document preparation system widely used in academia for the communication and publication of scientific documents in many fields, including mathematics, statistics, computer science, engineering... The writer uses markup tagging conventions to define the general structure of a document, to stylise text throughout a

document, and to add citations and cross-references. A TeX distribution such as TeX Live or MikTeX is used to produce an output file (such as PDF) suitable for printing or digital distribution. Here we are going to use Gummi, a LaTeX editor, and see how we can create and edit a document:

1. Open the file "example.tex" in Gummi.
2. Compile it and read the content of the generated pdf.
3. Open the file /lecturenotes/Latex/Simplified_Latex_Manual.pdf to understand the basic concepts of LaTeX.
4. Which are the minimum requirements (packages or commands) to create a document?

1.5 Intro to Regular Expressions

1. Surf to the webpage `https://regexr.com`.
2. Copy this text in the text panel.

3. Select option 'RegEx Reference' at the left, to visualize the documentation. Read it and see the different commands and examples.

4. Answer the following questions:

    a. How many words does the document contain? (use the regular expression `\w+`).
    b. How many words end with 's'? (use the regular expression `s\b`).
    c. How many words end with 'a' followed by 't' or 'y'? (use the regular expression `a[ty]\b`).
    d. How many words contain the substring 'at', followed by any symbol, and then followed by a 's'? (use the regular expression `at[^\s]s`).

1.6 Regular Expressions in UNIX

We are going to use the command prompt of UNIX to find files, directories and words using regular expressions with the commands `find` and `grep`. Open a console window and complete the following exercises. Use `man <command>` to see the documentation.

The command **find** is oriented for file searching, where the regular expression is commonly used to find specific file names. The structure is as folows:

```
find <directory> [options <value>]
```

Two useful options are **-type**, to determine the type of file to be searched, and **-iname**, that allows the use of a regular expression. For example, if we want to look for all the PDF files within the Documents folder:

```
find ./files/ -type f -iname "*.tex"
```

On the other hand, **grep** is used to filter some words within a file or set of files. The structure is as folows:

```
grep [options] "<regular_expression>" <files>
```

For example, if we want to find all the words "test" within a text file:

```
grep "Sigma" ./*.tex
```

1. Use the command **find** to filter all files with extension"txt" and whose names end with '5'.

2. Find in the "/usr/share/common-licenses/" directory all files starting with 'G'.
3. Create a file "texto.txt" in your working directory with the contents of the GPL-3 file. Note: use `cat` and >.
4. Use `grep` to find the word "license" in "texto.txt", case insensitive.
5. Find all those lines that start with the word "GNU".
6. Find all the lines that end in "and".
7. Find those words that begin with any two letters followed by "cept" (for example, accept).
8. Find all those words containing "too" or "two" (see what they have in common).
9. Find all those phrases that are enclosed in parentheses. For example, "(with or without modification)". Keep in mind the spaces.
10. Find all those lines whose first word begins with a capital letter and the line ends with a point.

More: exercises about regular expressions

Activities

1. Find the power set R^3 of $R = \{(1,1),(1,2),(2,3),(3,4)\}$. Check your answer with the script `powerrelation.m` and write a LaTeX document with the solution step by step.

2. Within the folder "files", find a TEX file in whose content appears the string `\usepackage{amsthm, amsmath}`. Note: use `grep` and escape the special characters with \. Complete the proof and answer the question.

References

- `http://pubs.opengroup.org/onlinepubs/009696899/basedefs/xbd_chap09.html`
- `https://www.johndcook.com/blog/regex-perl-python-emacs/`
- `https://www.gnu.org/software/grep/manual/html_node/Regular-Expressions.html#Regular-Expressions`
- `https://www.gnu.org/software/sed/manual/html_node/Regular-Expressions.html`