CS 561 Project 1 : Exploring Efficiency of FA Models
Due on October 18, 2023 (Wednesday)

1 Project Description

From wikipedia.org

> In computing, a regular expression is a specific pattern that provides a concise and flexible means to "match" (specify and recognize) strings of text, such as particular characters, words, or patterns of characters. Common abbreviations for "regular expression" include RegEx and regexp.
> Regular expressions are used by many text editors, utilities, and programming languages to search and manipulate text based on patterns. Some of these languages, including Perl, Ruby, AWK, and Tcl, integrate regular expressions into the syntax of the core language itself.

Conventionally, in programs, a pattern matching is defined as a regular expression. During program execution, the RegEx is converted to a FA, which is used to decide the matching. At first, the RegEx is converted to an NFA using a recursive decent parser (`http://matt.might.net/articles/parsing-regex-with-recursive-descent/`). After that, the language designers have a choice to either trace the resulting NFA, or convert it to a DFA and trace it deterministically, or even further minimize that DFA. The hope is that each step, on average, improves the runtime of string matching. But it does come at the cost of conversion and minimization algorithms. However, these optimizations can still be insufficient when the size of the string is very large. Researchers came up with a parallelization algorithm [1] to speed up the results, which comes with additional resources' usage.

In this project, you will implement the string tracing algorithms for different types of FAs. You need to implement the determinization and the minimization algorithms as well as the tracing of each implementation NFA, DFA/DFA_{min} and for an extra credit, DFA_{bpa}. You should finish this project by yourself (i.e., this is **not** a team project), and submit through onyx using the following submit command:

```
submit elenasherman cs561 p1
```

2 Design Requirements

Given the encoding of a NFA, your program should instantiate it and then trace it for a given input string s. Using the NFA encoding, your program should have an algorithm to convert it to a DFA, and then minimize that DFA. Given an input string s your program should be able to trace the instantiated NFA as well as DFAs. Note that the techniques for tracing strings in NFA and DFA are different. This is because the former creates a computation tree and the latter a computation

sequence. For an extra credit, you can implement a bulk synchronous parallel automaton DFA_{bsp}, which partitions the input string into a few chunks and traces them in parallel, and then "stitches" valid execution runs together.

The program takes three parameters: the type of simulation, the NFA encoding file and the input string file. The simulation types are `1` for NFA, `2` for DFA, `3` for minimized DFA DFA_{min}, and `4` for distributed DFA DFA_{bpa}.

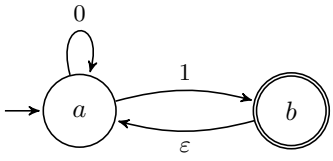
You are given a zip file with NFA encodings and input string files. It contains two folders: `tests` and `evals`. You can use files in tests to check the correctness of your implementations. There are four NFA encoding files: `tc0.txt`, `tc1.txt`, `tc2.txt` and `tc3.txt`; and the rest of files are the input string files, five for each NFA encoding file. Also, there are answer files with yes/no answers for each NFA encoding and its input strings (ordered according to the inputs' ids). The second folder contains the benchmark NFAs and four input strings, which you will use to run experiments. It has three benchmark files `bm1.txt`, `bm2.txt` and `bm3.txt` and four inputs for each of them. The inputs vary in size by an order of magnitude: $\forall i \in \{1, 2, 3, 4\}$ the size of `eval*_`i is 10^{2+i}.

2.1 Input NFA file to `FASimulator`

The input file has the following format:

- The 1st line contains the names/labels of the final states, i.e., F. The names are separated by white space. The first line can be empty if there are no final states.
- The 2nd line contains the name/label of the start state, i.e., q_0.
- The 3rd line contains the rest of the states, i.e., those states that are neither q_0 nor in F. Note, that line 3 can be a blank line.
- The 4th line lists the transitions. Transitions are separated by the white space. Three symbols compose a transition $s_0 s_1 s_2$, where
  - s_0 is the name of the "from" state.
  - s_1 is the symbol from the alphabet, i.e., $s_1 \in \Sigma$ or the empty string ε for which we use 'e' symbol, that is, 'e' $\notin \Sigma$.
  - s_2 is the name of the "to" state.

For example, this NFA



has the following encoding and the set of strings to test:

```
b
a
```

```
a0a a1b bea
EOF
```

Note: The 3rd line is empty because both states are already listed on the 1st and 2nd lines.

2.2 Output Specifications

We will give you three NFA encoding files and five input strings for each of them. If an input string is accepted by the NFA, then your program should print out **yes**; otherwise **no**.

2.3 Project Report

As a part of the project assignment, you will submit a report with empirical evaluations showcasing the size of each computational model and the runtime it took each model to process an input string. The evaluations are done on the benchmark files in `eval` folder: `bm1.txt`, `bm2.txt` and `bm3.txt`. Input strings are ordered in increasing size, i.e., $|eval1\_1| < eval2\_2|$. For each of the NFA files, you can display the data in the following format:

Model	Size	eval_1	eval_2	eval_3	eval_4
bm1					
NFA	$\|Q\|$, $\|\delta\|$	t_1^n	t_2^n	t_3^n	t_4^n
DFA	$\|Q\|$, $\|\delta\|$	t_1^d	t_2^d	t_3^d	t_4^d
DFA_{min}	$\|Q\|$, $\|\delta\|$	t_1^m	t_2^m	t_3^m	t_4^m
DFA_{bps}	# bat, # cpy	t_1^p	t_2^p	t_3^p	t_4^p

In sequential FAs the size is defined by the number of states and transitions, in the parallel model, it is defined by the number of batches (the partitions of the input string) and the total number of DFA copies invoked. The inner cells t_i^k are the averaged runtime of the model on each input. The average should be over at least three executions.

Then document your observations in terms of the model sizes, the corresponding runtimes, and how they change with increased input size.

2.4 Program Specifications

You can implement your program in any language that can be compiled on Onyx. . However, you cannot use any built-in NFA/DFA functions and libraries. Make sure to specify in the README how to compile it on Onyx. The execution file should be name `FASimulator`. For example,

```
[you@onyx]$ java FASimulator 3 tc1.txt in1_1.txt
```

3 Submission

Your project should be submitted through `submit` command on onyx. You must have onyx account to do so. If you don't have one, please let your instructor know.

You should submit:

1. All source files
2. A .txt README file that briefly describes all submitted files and how to compile your project.
3. A .pdf file with report data and your observations.

Note for late submissions: 10% will be deducted per day.

4 Grading

Grading will be based on the following elements:

- (5%) Have you submitted all required files? (including the properly formatted README, which also should contain project implementation reflection))
- (5%) Is the source code easy to understand (i.e., good structure and comments), authors tags?
- (5%) Can the code be compiled on onyx?
- (5%) Does your program take a file as an input, and print out to the standard out (i.e., screen)?
- (25%) Does your program run correctly? We will run test cases (some that you already have) to check for correctness.
- (10%) NFA to DFA implemented.
- (10%) DFA to DFA_{min} is implemented.
- (35%) Report with runtime data and your observations.
- (15% - Extra) Implementing and running DFA_{bpa}

5 Demo

```
[you@onyx p2]$ cat tc0.txt
b
a

a0a a1b bea
[you@onyx p2]$ cat in0_1.txt
0
```

```
[you@onyx p2]$ cat in0_2.txt
1
[you@onyx p2]$ java FASimulator 1 tc0.txt in0_1.txt
no
[you@onyx p2]$ java FASimulator 2 tc0.txt in0_2.txt
yes
```

References

[1] Holub, Jan and Štekr, Stanislav (2009) *On Parallel Implementations of Deterministic Finite Automata*, Conference on Implementation and Application of Automata.