Projects for CS3050 Theory of Computation

Aug-Nov 2024

1 Summary

Last updated on October 31, 2024

You are free to choose any of the following. A successful completion of any **one** of these activities is sufficient to claim the bonus points.

- Programming Project
- Presentation Project
- Problem Set 42

There is a separate sign up needed for the presentation project. For the other two, no sign up is needed. Please check the respective sections for more details.

2 Programming Project

There are two programming projects. First one is to build a tool similar to `grep` and second one is to build a Turing machine simulator.

2.1 Programming Project 1 - A `grep` like tool

The aim of this project is to build a tool similar to `grep`.

You are given a regular expression r and an input string and the task is to check if the input string belongs to $L(r)$.

This is done using the techniques that we learned in the course which we outline below.

- Stage 1: This stage involves building the necessary parts. There is no dependency between any of these sub stages.
  - Stage 1a: Takes a string as input and outputs an NFA accepting the string and rejecting everything else.
  - Stage 1b: Take a regular expression as input and obtain an equivalent DFA/NFA.
  - Stage 1c: Take an NFA and convert it to an equivalent DFA using subset construction
  - Stage 1d: Take a DFA as input and obtain a minimal DFA using the DFA minimization algorithm
- Stage 2: Using stage 1a and 1c, obtain a DFA accepting the string. Using stage 1b and 1c, obtain a DFA accepting the *complement* of the language corresponding to the regular expression. Use stage 1d to minimize the resulting DFAs.
- Stage 3. Obtain a DFA for intersection of the two minimal DFAs from stage 2 using product construction.
- Stage 4: Using stage 3, finally check if the input string belongs to the regular expression.

The above steps are solving the more general problem of finding a common string in the intersection of two regular expressions.

Note that the actual `grep` program is *not* implemented this way.

2.1.1 Choice of Programming language

- You are allowed to use low level languages like Rust or Functional programming languages like OCaml or Haskell.
- Your code will be benchmarked against a set of test cases.
- You are only allowed to use the standard libraries that comes with these languages. Since your code will be tested against large test cases, language like Python will be inadequate.

A good place to start reading about OCaml is here.

Test files and specification of regular expressions are given in the next section.

2.1.2 Specifications of test file

The first line of the input test file will specify the number of test cases. Following this, the regular expression and the string will be given in separate lines (in that order).

We now describe how the regular expression is specified. The supported operations in the regular expression are `union` ($+$), `concat` ($\cdot$) and `star` ($*$). Suppose that the regular expression is $(a+b)^* \cdot c$ and we want to check membership for *abc*. Then this will be specified in the test file as,

```
concat(star(union(symbol(a),symbol(b))),symbol(c))
abc
```

A sample test file with 5 enties is given below

```
5
star(symbol(a))
aaaa
concat(star(symbol(b)),symbol(a))
bba
concat(star(symbol(a)),union(symbol(b),symbol(c)))
aab
concat(star(union(symbol(a),union(symbol(b),symbol(c)))),symbol(d))
dabcd
concat(concat(symbol(0),symbol(1)),star(union(symbol(0),symbol(1))))
1011
```

Expected output on running your program on the above test file is:

```
Yes
Yes
Yes
No
No
```

Note that for stage 1b, you may also want to implement, given two DFAs how to obtain a DFA accepting (1) union and (2) concatenation of the languages of the two DFAs.

2.1.3 Deliverables

All the stages must be implemented fully and must pass all your test cases to be considered for evaluation. **Submission is open for all. There is no separate sign-up required**.

Submission link: Click here to open.

Final submission made in course moodle page before December 08 (Sunday, 11.59 PM) alone will be considered. No further extension on this deadline is possible.

You are expected to write a `Makefile` to build your program and build it using `make` tool.

Your submission to moodle should be a single zip file with all the relevant source files, `Makefile` and a `README` file explaining the purpose of each file. **Submissions deviating from these guidelines may be discarded**. Any instance of academic dishonesty will have severe consequences.

2.2 Programming Project 2 - Turing machine simulator

This question aims to build a simulator for Turing machines. Assume that the input alphabet set is `0,1` and work alphabet set, in addition to the input alphabets, contains `$` to denote the blank symbol and `|` to denote the left end marker.

You are given a deterministic Turing machine in a file in the following format:

- first line contains the list of states separated by space
- second line contains the start state
- third line contains the accept state
- fourth line contains the reject state
- subsequent lines contains transition rules of the form `(p,a) -> (q,b,D)` where,
  - `p` and `q` are any states appearing in the first line.
  - `a,b` are tape symbols
  - `D` can be `L` for head going one step left, `R` for head going one step right and `S` for head staying without any change. The entry means that: "on reaching state `p`, and the current head position is `a`, change state to `q` , write `b` on the current head position and move according to the direction `D`".

Following is a sample Turing machine file that checks if the input is a `0` or `1`.

```
q0 qyes qno
q0
qyes
qno
(q0, |) -> (q0, |, R)
(q0, 1) -> (qno, 1, S)
(q0, 0) -> (qyes, 0, S)
```

Note that it is suffices to specify only the necessary transitions. For instance, we have not given the further transitions for `qyes` and `qno` or the transition for `q0` on `$`.

2.2.1 Choice of Programming language

- You are allowed to use low level languages like Rust or Functional programming languages like OCaml or Haskell.
- Your code will be benchmarked against a set of test cases.
- You are only allowed to use the standard libraries that comes with these languages. Since your code will be tested against large test cases, language like Python will be inadequate.

A good place to start reading about OCaml is here.

2.2.2 Specifications

Your simulator should take two arguments:

- First argument is the name of the file containing the turing machine description
- Second argument is an input string to the Turing machine.

Your task is to simulate the Turing machine on the given input as follows:

- in each step, if the transition for the current state and the symbol is well defined, the Turing machine must output the configuration and make the move.
- if the transition for a state and a symbol is not defined, then say `aborting` and stop the simulation.
- and finally output whether the Turing machine `accepted` the input string or `rejected` the input string and the stop the simulation.

The configuration of the Turing machine needs to printed as follows with the state, tape content and head position separated by `#` symbol

```
state # tape content # head position
```

For example, the configurations of the above Turing machine on the input `0` is displayed as follows, starting with the start configuration.

```
q0#|0$#0
q0#|0$#1
qyes#|0$#1
accepted
```

One application of the transition function is called as *one step* of the Turing machine. If the Turing machine exceeds 10^8 steps, stop the simulation and report that `time exceeded`.

For processing the Turing machine file, you can choose to build a parsing mechanism from scratch or use a lexer and a parser. You can use parsers that are supported by your programming language.

- Rust: See here for parser generator programs and see here for lexer generators
- OCaml: You can use, ocamllex and ocamlyacc
- Haskell: You can use, Happy and Alex.

2.2.3 Deliverables

Your program must confirm to the input specifications. You can assume that all the Turing machine files are written correctly according to the format specified above. **Submission is open for all. There is no separate sign-up required**. Submission link: Click here to open.

Final submission made in course moodle page before December 08 (Sunday, 11.59 PM) alone will be considered. No further extension on this deadline is possible.

You are expected to write a `Makefile` to build your program and build it using `make` tool.

Your submission to moodle should be a single zip file with all the relevant source files, `Makefile` and a `README` file explaining the purpose of each file. **Submissions deviating from these guidelines may be discarded**. Any instance of academic dishonesty will have severe consequences.

3 Reading Project

The area of Theory of Computation is rich in terms of ideas. One of the goals of reading project is to get a taste of it by reading up selected topics. The key aim is to broaden your understanding and also develop a better insight as to how they are connected to each other and to the topics covered in class.

Each project can be done in groups of size at most 2. The final presentations will be scheduled on November 09 (Saturday).

3.1 Topics

The list of topics are given below (more topics will be added if needed):

- *Homomorphisms* -
  - **Curiosity questions**: The language $L_1 = \{0^n 1^n \mid n \geq 0\}$ is not regular and can be argued via pumping lemma. The following language $L_2 = \{0^n \# 1^n \mid n \geq 0\}$ which "looks" similar to L_1 is also not regular. Is it a coincidence or can we formalize this ?
  - **Answer**: (Inverse) Homomorphisms !
  - **Take away**: Using homomorphisms to prove regularity/non-regularity of languages.
  - **Reading materials**: Kozen, Lecture 10.
  - **Expectation**: What are homomorphisms and how can they be used to argue regularity/non-regularity of languages ?
- *String pattern matching algorithm* : Knuth-Morris-Pratt algorithm.
  - **Curiosity questions**: Are there any algorithms that are used in practise which is inspired by automata design ?
  - **An answer**: The Knuth-Morris-Pratt algorithm
  - **Take away**: How can finite automata be helpful in faster pattern search
  - **Reading material**: Original paper describing the algorithm. Also Jeffe Erickson's notes section 7.4, 7.5, 7.6, 7.7 and exercises 4, 5, 7, 8 both of lecture 7.
  - **Expectation**: Completely understand the KMP algorithm and how an automata based thought process helps in string search.
- *Arden's Theorem and Kleene Algebra* -
  - **Curiosity questions**: Arden's Theorem helps find a regular expression for a state in a finite automata. The process looks very similar to how one would solve a system of linear equations. Is there a way to view this as a solution of a system of equations where the variables corresponds to the states ?
  - **Partial answer**: Indeed ! This is explained in Kozen, Supplementary Lecture A.
  - **Take away**: A different way to obtain all regular expressions of a finite automata.
  - **Reading material**: See here for a proof of Arden's Lemma. Also Kozen, Supplementary Lecture A.
  - **Expectation**: Understand the statement and proof of Arden's lemma. Understand what is reflexive transitive closure and its relation to Arden's Theorem (Kozen, Miscellaneous exercises 1 and 23). Present the connection to Kleene Algebra (Kozen, Supplementary Lecture A).
- *Buchi Automata - Automata theory and model checking*:
  - **Curiosity questions**: Automata theory deals with strings of finite length. What about strings of infinite length ? Such strings can be used to model behaviour of systems that runs infinitely like a satellite transponder, a traffic junction signal system or even the run of an operating system process scheduler.
  - **Partial answer**: Buchi automata
  - **Take away**: Natural ideas from automata theory that worked for finite length strings also (mostly) works in the infinite length setting too. Such an extension is also very helpful in practical settings like verification.
  - **Reading material**: Section 2 and 3 of this article by Madhavan Mukund.
  - **Expectation**: Being able to explain what is Buchi automata, closure properties, connection to LTL and how it is used in mode checking.
- *Regular Languages over Unary*:
  - **Curiosity questions**: We know that unary languages like PRIMES are not regular. So primes as a sequence cannot be recognized by a DFA. What kind of sequences will make a unary language

regular ?
  - **Partial answer**: Ultimately periodic sequences
  - **Take away**: A unary language is regular if and only if the lengths of strings in the language are ultimately periodic.
  - **Reading material**: Kozen Lecture 12
  - **Expectation**: Being able to explain the **Take away** and gives a structural characterization of automata accepting unary languages. Also solve the Miscellaneous exercise 45.
- *Finiteness of Alphabet Set - Topological argument*:
  - **Curiosity questions**: What is the reason to assume that the alphabet set is finite in a Turing machine ? Can we make this reasoning rigorous ?
  - **Partial answer**: Topology of the space of alphabets !
  - **Take away**: Using ideas from Topology, Turing sketches a reason in a footnote of his paper that finite alphabet set suffices using ideas from topology. The purpose of this presentation is to make this idea more precise.
  - **Reading material**: See this article. Also check this write up.
  - **Expectation**: Introduce basic definitions, ideas and results from topology and execute the argument Turing gave in the footnote using them.

3.2 Rules

The rules of this game are: pick a topic from the above and understand it well. There will be an interim meeting (this carries 40% weight) with the instructor on November ~~02~~ 05 for a review and for clarifications. You need to give a black board talk to the instructor/TAs on November 09, Saturday. A schedule will be announced soon.

It is strongly recommended to use the blackboard. Nevertheless, if you plan to use slides, please have a word with the instructor and get permission.

Please write your name against the project ~~here if you are interested in doing this. Deadline for registering for this project is October 28, 11.59 PM~~ The deadline is over now. Thank you !

4 Problem Set 42

42 is the answer to ultimate question of life, the universe, and everything.

Feeling inspired, this activity consists of solving a collection of SOLID questions that appear in the practise problem sets along with a few more challenging questions. See here for a link to the questions. More questions will be added as we progress.

What is expected ? Solve these problems and submit your solutions that is handwritten neatly and legibly. You must deposit your answers in the box kept on Table 24, First floor, D03 Dr. APJ Abdul Kalam Building on or before December 05, 5.00 PM (Thursday). The table number is reverse of 42 !

If you are planning to do this, it may be a good idea to start early !