Projects for CS3050/CS5017 Theory of Computation

Aug-Nov 2022

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 presentations are scheduled on November 26 (Saturday).

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

- *Arden's Theorem and Kleene Algebra* -
  - Group: Ujjwal Verma and Milan Chakraborty
  - **Curiosity questions**: Why do we need the rules 9.12 and 9.13 as part of Kleena algebra axioms ? Why care about Kleene Algebra ?
  - **Partial answer**: Finding $\leq$ least solution using Arden's Theorem. The rules 9.12 and 9.13 precisely capture this !
  - **Take away**: A different way to obtain regular expressions directly from DFA using Kleene Algebra
  - **Reading material**: For a statement, proof and examples, see here. For connection to Kleene algebra - Kozen, Supplementary Lecture A.
  - **Expectation**: Understand the statement and proof of Arden's lemma and the uniquness of the solution. For uniqueness, see here also. 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.
- *Homomorphisms* -
  - Group: Arun Sankar and Jyothiradithya
  - **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 ?
- *An application of Myhill-Nerode theorem*: 2DFAs accepts regular languages
  - Group: Pratham Ravindra Nagpure and Nideesh N
  - **Curiosity questions**: Suppose a DFA is allowed to re-read the input as many times as it wants. Call them as 2DFAs. Can a 2DFA accept non-regular languages ? Can there be an DFA accepting the same language as that of a 2DFA ?
  - **Answer**: No & Yes
  - **Take away**: Myhill-Nerode theorem as a tool to show limitations of computational models.
  - **Reading materials**: Kozen, Lecture 17 and 18.
  - **Expectation**: Being able to completely explain why 2DFAs are not more powerful that DFAs.
- *The Chomsky-Schutzenberger Theorem*
  - Group: Kavya Syamala and Sneha Bhattacharjee
  - **Curiosity questions**: For regular languages, there is a well defined notion of unique minimal DFA via Myhill-Nerode relation. Is there a unique context free grammar for every context free language ?
  - **Partial answer**: Not quite ! But all context free grammars "looks" the same as a particular kind of context free grammar (called Dyck grammars).
  - **Take away**: The structure of any context free languages are essentially captured by Dyck languages.
  - **Reading material**: Kozen, Lecture 20, Supplementary lecture G. Also see here
  - **Expectation**: Understand the structural characterisation and fully demonstrate it using an example.
- *String pattern matching algorithm* : Knuth-Morris-Pratt algorithm.
  - Group: Karavadi Suchith and Geddam Gowtham
  - **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 have a look at 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.
- *Ogden's Lemma and application to ambiguity of grammars* :
  - Group: Cipriyano Simous and Zahir Khan
  - **Curiosity questions**: To show that the grammar for a language is

ambiguous, it suffices to show two parse trees for the same string. Can there be languages where *all* grammars are ambiguous (aka the CFL is inherently ambiguous) ?
  - **Partial answer**: Yes. This can be checked using Ogden's lemma, a stronger form of Pumping lemma for CFLs.
  - **Take away**: We can check if the language is itself ambiguous using Ogden's Lemma in addition to non-context freeness.
  - **Reading material**: Section 4.3 (complete) and Section 4.4, Theorem 4.4.1 of this pdf
  - **Expectation**: Understand the statement of lemma, proof and its application to ambiguity.
- *Parsing techniques - LL(1) parsers*:
  - Group: Kritika Kashyap and Susmita Behera
  - **Curiosity questions**: How is the theory of CFL helpful in designing compilers for programming languages ?
  - **Partial answer**: Key notion used is parse trees. One of the popular method, called LL(k), builds parse trees in a top-down fashion using left to right scan and using k-look aheads.
  - **Take away**: Use of LL(k) parsers in practise.
  - **Reading material**: Section 5.3 of this pdf.
  - **Expectation**: Understand how the FIRST and FOLLOW sets are computed, proof of correctness and an example.
- *Buchi Automata - Automata theory and model checking*:
  - Group: Deepa Sara John and Ashicka P Saj
  - **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.

The rules of this game are: pick a topic from the above and understand it well. There will be an interim meeting with the instructor on November 19 (a week before the presentation day) for a review and for clarifications. You need to give a black board talk to the instructor/TAs on November 26.

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 sign up here if you are interested in doing this.~~ Thank you ! We are no longer taking new projects.

Programming Project

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

- 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.

You are free to choose any programming language of your choice. ~~Test files and specification of regular expressions will be provided in due course.~~

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
```

For processing the regular expression as a part of the stage 1b, you need not implement a parser. You can use parsers that are supported by your programming language. Most of the programming languages have either libraries (like in Java and Python) or parser generator programs (like `flex` and `bison` in C) that supports parsing.

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.

All the stages must be implemented fully and must pass all the above test cases to be considered for evaluation. Final submission made in course moodle page before December 10 (Saturday) alone will be considered.