**Own Issue Project Proposal**

|  |  |  |  |
| --- | --- | --- | --- |
| Name : | *Callum Hynes* | | |
| Provide an overview of what your program is intended to do. | Compile a regular expression into a deterministic finite automaton that can match a given regular expression so that regex can be used for efficient pattern matching, in O(n) time  DFAs are represented by a [graph](https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)) (specifically a multi-digraph if u want to sound fancy) , which represents all possible paths between states (it’s a state-machine), overall representing a pattern to match. NFAs are a more general form of DFAs, but can have non-deterministic outputs from a state – i.e. one state can have multiple out paths which could both be activated by the same input char. This means that it is not always determinable which path should be traversed in an NFA. Powerset construction can be used to convert an NFA to a DFA, which solves this issue, and allows the pattern-matching to run in linear time (O(n)), at the cost of exponential space (O(exp(n))). | | |
| What are the steps the program needs to complete to be effective?  /  What are the specifications that need to be met? | 1. *Parse the regular expression into an NFA* 2. *Optimise and minimize the NFA, and perform e-closure* 3. *Use powerset construction to convert the NFA into a DFA*   *The program should*   * *Produce a minimal, efficient DFA* * *Be able to test the DFA with an input string to find if the regex is present within the string* * *Evaluate the DFA on given strings to find positional matches (i.e. return indices of matches)*   *It should also*   * *Provide an easily usable and well-thought-out API interface* * *Be very well-structured and extensible, for example to allow future expansion (e.g. compiling efficient machine code from the DFA to make hyper-optimised compiled regex; This is outside the scope of the project but the project should be built in a scalable way to allow these kinds of expansions)* | | |
| Identify the Complex Programming Skills you intend to include and how you intend to use them. | *1. Classes ~~(because im not a psychopath)~~ (to organise code)* | | |
| *2. GUI to view NFA for debugging (else I will never understand all the bugs I have) (Very useful when developing to show the optimisation steps. Could also be useful for end user if they want to see how their regex is being parsed)* | | |
| *3. External libraries (for GUI) (Also for the base library itself, which uses numpy to efficiently store the transition table as a matrix)* | | |
| *4. there isn’t four (I don’t think idk what counts as a complex programming skill imo making DFAs and stuff sounds complex enough for me like look at that fancy name: deterministic finite automaton)* | | |
| What will be the biggest challenges you will face with this project? | Powerset construction, e-closure and minimisation can be quite challenging.  Powerset construction especially can be quite mathmatical and can be hard to detect if two paths along the graph have any overlap, which means that their powerset would be reachable.  I changed my mind powerset construction was relatively easy  The most challenging bit was this:    And wtaf my code was doing to get there like actually what is that its literally just a scribble it looks like a 5yo drew it  Ok I changed my mind again the most challenging part was substring (non-anchored) searches, which require a lot of powerset construction to all go right. The issues during this stage are difficult to diagnose as the graph diagram often becomes overwhelmed by the fail-return-to-start moves and such. | | |
| Status: | Approved Rejected  U wouldn’t do that to me  definately | | |
| Teacher: | Mrs Cropp | Date : |  |