Slide 1:

Good morning, everyone. Thank you for coming to my presentation today. My name is Yang, and I am going to talk about my project “Automata and logic for regular functions”. When it comes to regular functions, it is not a well-known concept for people. And there is a similar concept called regular languages which is widely used in the theory of computation and is more familiar. So, let’s start from regular languages.

Slide 2:

A language is a set of [strings](https://brilliant.org/wiki/strings/) which are made up of characters. A regular language is a language that can be expressed with a regular expression or a deterministic finite automaton (DFA for short). Here is an example DFA that accepts all strings beginning with 0. Regular languages share a number of properties. It not only can be described using several equivalent models such as [non-deterministic finite automaton](https://en.wikipedia.org/wiki/Nondeterministic_finite_automaton) and [two-way finite automaton](https://en.wikipedia.org/wiki/Two-way_finite_automaton) but also is closed under various operations such as [union](https://en.wikipedia.org/wiki/Union_(set_theory)), [concatenation](https://en.wikipedia.org/wiki/Concatenation), and [Kleene star](https://en.wikipedia.org/wiki/Kleene_star). Given two DFA *A* and *B*, it is decidable whether they accept the same language. Therefore, we can analyse algorithms that are related to questions such as equivalence and emptiness. With so many desirable properties, regular languages are widely used in areas such as algorithmic verification and text processing. However, instead of the concept of sets accepting and classification, sometimes we prefer to use functions.

Slide 3:

Here is an example list-processing program that deletes some given data in the list and outputs the new list. You do not need to look into details of this program. We simply use it to preform functions such as delete all occurrence of 2s in an integer list. In this way, we obtain a simple function from input to output integer list. If we want to use finite-state verification algorithms to analyse the program result, we can only find the problem is not decidable using regular languages because we cannot find a set of [strings](https://brilliant.org/wiki/strings/) that can define this simple function. Therefore, functions are needed to perform such tasks.

Slide 4:

Regular functions are the analog of regularity of regular languages for defining functions. Current known applications of regular functions are few. It is only used for verification of list-processing programs and document transformation tasks such as shifting or switching given context in some files.

Now let’s define regular functions. A regular function is a function that can be performed by a deterministic two-way transducer, which is very similar to DFA but with string output at each transition step and I will talk about it later. Regular functions also have similar closure and decidability properties. It can also be defined with different transducer models. And its definition causes a linear O(n) upper bound for the length of the output string.

Slide 5:

Although regular functions have a number of appealing theoretical properties, there exists few simulation tools related to transducer models that define regular functions and people who are not familiar with this area will not be able to see through the difference between models intuitively. Therefore, this project is focused on the implementation of transducer models that express regular functions and translation algorithms between different models. These listed models and translation algorithms are chosen to be implemented. The first model 2DFT is a classic model that was studied a half century ago and was introduced by Gisburg and Rose. The second model SST is from recent research of Alur and Cerný. We will discuss about these two models shortly. And the last model monadic second-order logic transducer is more like a graph transaction and is more complicated. It is introduced by Engelfriet and Hoogeboom in 2001. Translation algorithms are the focus of this project. Algorithms between 2DFT and SST described by Dartois, Jecker and Reynier was implemented. We will discuss more technical details about the last model and the translation algorithms in the final report.

Now let’s have a close look at the first two models that define regular functions.

Slide 6:

The first transducer model is the deterministic two-way finite state transducer. As is shown in the state diagram of the example 2DFT, it is consist of sets of states, input and output alphabet, initial and final states and a transition function represented by arrows (1 represents right and -1 represents left in the transition function). Let’s have an example run on this 2DFT. Assume we have an input string of abb, the 2DFT will surround it with the left and right end-markers and set its reading head to the first symbol a in our case. We start in the initial state q0, read symbol a and follows the transition function to stay in state q0, output symbol a and move the reading head to the right. Then we read b in state q0 and again follow the transition function to stay in state q0, output symbol b and move the reading head to the right. If we keep running this transducer, we will get the output as the original string concatenated with its reverse. The difference between a 2DFT and a DFA is that the 2DFT contains output symbols and the moving direction of the reading head in its transition function. This model is close under sequential composition. If we have two 2DFTs and we give the output of the first 2DFT to the second one as input and get the result, the whole process can be simulated by a single 2DFT. This is proved by Chytil and Jákl.

Slide 7:

The second transducer model is the deterministic streaming string transducers. You can think of it as a one-way automaton with registers. Comparing to 2DFT, SST has a variable set and has no final state. Instead, it uses a partial output function to output strings that are stored in variables. The transition function is also divided into two parts: one function for state updates and another one for variable updates. As is shown in the state diagram, the variable set of the example SST contains x and y. Assume we have an input string of abb, the SST will go through the string from left to right. We start in the initial state q0 and read symbol a, we stay in state q0 and append a to x. Then in next step, we read b in state q0, we move to state q1 and append b to both x and y. In the last step, we read b in state q1, we stay in state q1 and append b to both x and y. The transducer has now processed the whole input string and use the partial output function of the last state q1 in our case to output sting bb stored in variable y. This is a complete run of the SST. If each variable is only used once in these functions (so we don’t have functions such as update x with double x), then we can call the SST copyless. And such copyless SSTs have the same expressiveness as regular functions. The ability of storing variables seems to occupy more memory space than other models, but actually this structure has advantages in type checking and equivalence checking problems as is stated by Alur and Cerný. As we can see, through comparison of these two models, each different model has its own advantages. Therefore, translations between models may help us to address the priority problems of the same task under different situations.

Slide 8:

In order to implement the simulator for regular functions, the following tools are chosen. Firstly, Java is chosen as the programming language because of its object-oriented nature which may help classify different transducers. Its extensive libraries which may help in later implementation are also taken into consideration. Secondly, git and GitHub are used as the version control tool and the cloud-based backup storage. Lastly, LaTex and TikZ library are also used to draw state diagrams of output transducers to provide a graphical output which help users to have a better understanding of the resulting transducers of the translation algorithms.

Slide 9:

When it comes to project management, this project use a plan-based methodology. Initial plan was made in the project specification and was slightly adjusted and detailed in the progress report. These plans were followed throughout the whole project. One special thing about this project is that it started early in the summer vacation. Considering that there will be some busy days during the terms, almost one month was spent on this project in order to release pressure and achieve better results on both sides. It turns out to be a wise decision because it definitely helps me to focus on the both sides and produce the best as I can rather than work in haste to meet the deadlines. The most challenging event happened due to my carelessness. A small detail inside the algorithm from 2DFT to SST was not given enough thoughts before coding and the problem was noticed when implementation was almost completed. It causes the resulting SST to be copyful and express strickly more than regular functions. The implementation was planned to complete in two weeks. Although actions were taken immediately, understanding and building a new algorithm consumed almost all remaining buffer time in term 1 and delayed the project plan for almost a week. Thanks to the fast progress in other parts and the planned buffer time, the remaining plan was almost unaffected. Challenges are not always bad to have. In order to prevent such situation from happening again, pseudo code was created before the actual implementation in the later development.

Slide 10:

The program was tested using white-box testing method. Transducer encodings that perform 6 different transduction functions were created to go through all the branches in model and translation algorithm simulation. The program passed all these tests and the system achieved most of the goals set upon initialising the project. It can be concluded that in this project, a number of new concepts and algorithms were learned and put into practice. Only the abandoned algorithm we mentioned before was not completed due to the time constrains. As the result, we cannot analyse and compare different translation algorithms for two transducer models.

Slide 11:

One way to fix the abandoned algorithm is to introduce another transducer model called heap-base transducer. With the help of this intermediate transducer, the algorithm can achieve the translation by combining two translations. Therefore, more transducer models and translation algorithms are needed to enrich this project. A graphical user interface could also be a useful extension for this project. Users can input by drawing state diagrams of transducers. It also can be fully interactive when users run the transducer on given inputs. In this way, users can have a more straightforward understanding of how transducers work.

Slide 12:

Now let me run some examples to demonstrate my work.

Demo

Slide 13:

This summarises the presentation. Thank you for listening. Any questions?

Slide 14:

Q & A