13 Logic and Computation

13.1 Minimisation of Deterministic Finite-State Automata (10 units)

This project requires an understanding of the Part II course Automata and Formal Languages.

1 The DFA Minimisation Algorithm

A *deterministic finite-state automaton* (DFA) is one of the simplest computing machines, consisting of a finite number of states and bounded read/write memory. Such machines are very limited in the things they can compute. However, the tasks they do compute are performed extremely fast, like integer reductions mod m.

Associated to each DFA is a unique minimal state DFA; the unique one which accepts the same language and has the smallest possible number of states. Thus, every regular language can be represented by a unique minimal state DFA. From hereon, an (n, k)-DFA is one with n states and an alphabet of size k.

There are many well-known algorithms that, on input of any DFA, produce the minimal DFA associated to it. We suggest that in this project you use *Hopcroft's table-filling algorithm*, as detailed in [1] or in the lecture notes. The purpose of this project is to count all the minimal (n, k)-DFAs for sufficiently small (n, k), and to investigate which of these define finite languages.

There are many symmetry properties of DFAs you can appeal to when going through this project. These might make your programs run faster and more efficiently (and in some cases, remove the need for unnecessary additional code). Moreover, you may find that you can verify *some* of the small cases of what you are asked to compute, by hand, which might be useful to do to check that your programs are running correctly.

2 Conventions

Without loss of generality, we will assume that our states are always labelled by integers, and that any DFA with n states has these labelled by $\{1, \ldots, n\}$. We will also always assume that the state labelled 1 is the start state. Finally, we will assume that any DFA on k letters has alphabet $\{0, \ldots, k-1\}$. We will keep all these conventions throughout the project.

An (n, k)-transition table is a transition table for an (n, k)-DFA (recalling all the conventions set out above).

Before you begin this project, you will need to establish a way to store transition tables. As we have adopted the convention that state 1 is the start state, this does not need to be reflected in the table. One way to store such tables could be as a matrix (indeed, MATLAB uses matrices as one of its primary data structures). You can use an $n \times (k+1)$ matrix to represent an (n, k)-transition table, with the final column consisting of 1's and 0's to denote which states are accepting or non-accepting.

3 Counting DFAs

The number of (n, k)-DFAs can be vast, even for relatively small k and n. However, the number of unique *languages* defined by these is much less than the total number of possible DFAs.

Question 1 Give a closed-form expression for the number of DFAs with n states and alphabet of size k. Present this as a 6×4 table of values (in exponent form, to 2 significant figures) for $1 \leqslant n \leqslant 6$ and $1 \leqslant k \leqslant 4$.

4 Accessible states

A state in a DFA is said to be *accessible* if it can be reached by starting at the start state and following a path in the transition diagram labelled by some word w. Otherwise, it is *inaccessible*. Clearly, a minimal DFA can have no inaccessible states.

In all subsequent questions, an *arithmetic operation* can be taken to be the addition, subtraction, multiplication or division of two integers; or the reading or re-writing of a matrix entry.

Question 2 Write a program to determine the set of accessible states from any arbitrary transition table. Run your program on the DFAs given by `Table1, Table2, Table3` and `Table4` in the file `DataTables` on the CATAM website. What is the complexity of your algorithm for an arbitrary (n, k)-transition table, in terms of n and k? Compute the number of (n, k)-DFAs with no inaccessible states, for $k = 2$ and $n = 1, 2, 3, 4$.

5 The Table-Filling Algorithm

Two states p, q of a DFA are said to be *equivalent* if, for every word w, if we follow the two (unique) paths in the transition diagram from p and from q labelled by w, then either both end at accepting states or both end at non-accepting states. Hopcroft's table-filling algorithm is a quick and efficient way to determine which states of a DFA are equivalent.

Question 3 Write a program to implement the table-filling algorithm on any arbitrary transition table and then produce a transition table for a minimal DFA defining the same language. At some point you may need to use part of your program from Question 2 to remove inaccessible states. Run your program on the DFAs given by `Table1, Table2` and `Table3` in the file `DataTables` on the CATAM website. What is the complexity of your algorithm for an arbitrary (n, k)-transition table, in terms of n and k?

6 Finding all languages describable by an (n, k)-DFA.

It is often useful in mathematics to compute and analyse some special cases of a mathematical object, to gain some intuition for how the object behaves as a whole. In this project, we will try and understand properties of minimal DFAs and the language they define, by looking at *all* the minimal (n, k)-DFAs for small n and k. This section will require you to use the programs developed in the previous sections as sub-routines.

Question 4 Compute the number of distinct regular languages definable by a minimal $(n, 2)$-DFA, for each of $n = 1, 2, 3, 4$. Remember that there may be several transition tables which define equivalent DFAs (and thus define the same language), so you will need to account for this.

7 Finite languages

It is immediate that every finite language is regular. We will now investigate which of the minimal DFAs computed in previous questions define finite languages.

Question 5 Let D be a DFA with n states. Prove that $\mathcal{L}(D)$ is infinite if and only if $\mathcal{L}(D)$ contains a word w of length $n \leqslant |w| \leqslant 2n-1$.

Question 6 Write a program to determine if an arbitrary transition table defines a finite language, and if so, outputs the size of the language it defines. Run your program on the DFAs given by `Table1, Table2` and `Table3` in the file `DataTables` on the CATAM website. It may help to use your program from Question 3 to minimise your DFAs first, or you may find your algorithm takes a long time to run.

From Question 4 you should have a list (possibly with some repetition) of all the minimal $(n,2)$-transition tables, for $n = 1,2,3,4$. You should make use of this list in the remaining questions.

Question 7 Let $f(n,k,s)$ be the number of regular languages of size s definable by a minimal (n,k)-DFA. For $n = 1,2,3,4$, compute all values of $f(n,2,s)$ for $0 \leqslant s \leqslant 2^n - 1$. What about when $s \geqslant 2^n$?

Question 8 For $n = 1,2,3,4$, write out all minimal $(n,2)$-transition tables (only *one* for each equivalence class) which give the finite languages of size 1 and of size $2^{n-1}-1$ (if any exist). Write out the language each of these defines.

Question 9 For $n \geqslant 1$, give closed-form expressions in terms of n for $f(n,2,1)$ and $f(n,2,2^{n-1}-1)$. Prove your expressions hold. What about $f(n,2,s)$ when $2^{n-1} \leqslant s \leqslant 2^n - 1$?

References

[1] J.E. Hopcroft, R. Motwani and J.D. Ullman, *Introduction to automata theory, languages and computation* (Chapters 2–4), 2nd edn, Addison-Wesley, 2001.

