CS 454 Theory Of Computation Spring 2-18

Final Project Proposal: Nathan Kamm, David Tauraso, Brandon Williams

References:

DFA Minimization
https://www.tutorialspoint.com/automata_theory/dfa_minimization.htm
Union/find for DFA Equivalence
drona.csa.iisc.ernet.in/~deepakd/atc-2010/equivalence-checking-seminar.pdf

Problem statement:

Design an efficient algorithm to test the equivalence of DFA's. It is possible to solve this problem using the UNION/FIND data structure that we discussed in CS 415 using the Hopcroft - Karp algorithm that improves the Time Complexity to O(n|sigma|), where n = the union of the states between the two dfas being compared and sigma = the number of input alphabets. If the input DFA's are equal, the last step is to show the smallest possible DFA that can be made from each input DFA.

Basic definitions and example:

We will be using a UNION/FIND data structure to implement an algorithm that test the equivalence of two DFA's to see if they accept the same language.

Example: L(1) = L(2) ?

See Applications and Extensions of the Union -Find algorithm on page 144

Methodology of DFA Equivalence:

We are testing equivalence between two distinct states p and q, we say they are equivalent if for all input strings w, delta(p,w) is a final state if delta(q,w) is a final state. This means either both are accepting or both are non-accepting (drona). The DFA's are equivalent if their start states are equivalent.

Data Structures Used and Hopcroft - Karp Algorithm Implementation:

Union/Find Data structure:

The Union/Find, sometimes called a disjoint set, data structure is a simple and powerful data structure that is taken full advantage of by the Hopcroft - Karp algorithm for testing dfa equivalence. There are two main types of instructions that this data

structure. The first is FIND(i), which returns the name of the set that i belongs to. The second is Union(j,k), which combines the set j and k into one set.

Main steps involved:

1.) Make a set for every state in the union of the two DFA's
2.) Union the first two states into one set and push onto a stack
3.) Pop from stack if not empty and search for the sets which contain the states reached by the transitions from the popped pair; if the sets equal each other do nothing, otherwise union the two different sets into one set.
4.) The dfas are considered equivalent If and only if no set contains both a final and a non-final state

Time Complexity when testing equivalence with our implemented structure is based on the four above mentioned steps. Each step's time complexity is explained in detail below.

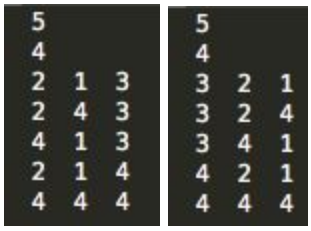
1.) Step one is achieved in O(n) since we have to create a 'parent list' which is the size of the number of states of both of our dfa's combined. The parent list is the beginning of our set data structure that we use the union and find functions on to achieve greater efficiency than other implementations such as we programmed in project 2.
2.) Step two is a simple operation which is achieved in O(1) since all we are doing is taking both the start states of the two dfas, union them together, and push them on to a stack.
3.) Step three is where most of our work is being accomplished in our algorithm. Here we go into a loop while the stack is not empty and pop a pair of states from the stack, for every non-terminal value we use the delta function to find the neighboring states from each state in the pair we popped. We then use our Find function to find the set that these states belong, if the sets are not equal we then union these and push them onto the stack. This step is bounded by the number of input symbols in our non-terminal alphabet times with the sum of the total number of states that equal the union of the states in our first and second dfas that we are testing.
4.) Finally, after we iterate through every state, transition, and comparing the corresponding set that they belong to, we can make a final check that determines the dfas are equivalent if and only if no set contains both a final and a non-final state.

Sample Input/Output:

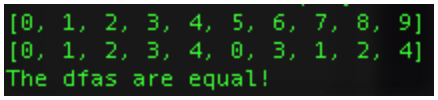
The output is structured so that the first line of output is the original list that represent the states and the sets that they belong to. The index position represents the states, and the value in the index position represents the set that the state belongs to. The second line is the final list/set structure that is obtained after running the Hopcroft - Karp algorithm

Test Case 1:
A string having a substring of length 2, whose characters are the same
Ex: [aa]bc

DFA1 DFA2
(dfa7.txt and dfa8.txt in submit file)

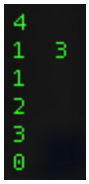


Output:



Test Case 2:
A string has an odd number of A's
(testdfa.txt and testdfa2.txt in submit file)

DFA1 DFA2



Output:

```
[0, 1, 2, 3, 4, 5]
[0, 1, 0, 1, 0, 1]
The dfas are equal!
```

Test case 3:
A string that contains one one
(dfa1.txt and dfa2.txt in submit file)

DFA1

```
2
1
0 1
1 1
```

DFA2

```
11
1 7 8 9 10
2 1
7 7
3 1
4 1
5 1
6 1
0 1
8 8
9 9
10 10
1 1
```

Output:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
[0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1]
The dfas are equal!
```

Problem statement:

Use an algorithm that when given a DFA, will output the minimize DFA.

Basic definitions and example:

Given DFA(large) we will create an equivalent DFA(small)

{JFLAP STUFF #1}

Data Structures and Algorithms Used:

1.) Step 1 uses Breadth First Search (BFS) to mark all reachable nodes in the graph.

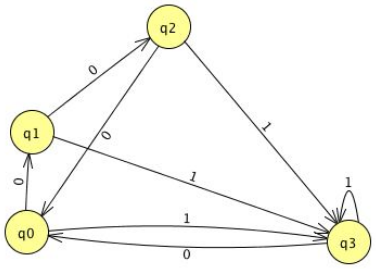
2.) Step 2 Iterates through the graph and deletes the nodes that are not reachable
3.) Step 3 Runs the Table Filling Algorithm (TFS) on the given graph
    a) Make a matrix whose dimensions are the total number of states
    b) for all cells in the lower triangle
        if x or y in (x, y) coordinates is an accepting state
            Mark the cell
    c) for all cells in the lower triangle
        For all transitions of x on the alphabet and transitions of y on the . . alphabet
            If the (transition_x, transition_y) cell is marked and (x, y) cell . is not marked
                Mark the cell (x, y)
4.) Step 4 Once table filling is finished the program iterates through the lower triangle, collecting the coordinates that reference the cell that is not marked(x, y)
5.) Use union find to find all the pairs and group them into disjoint groups.
6.) Group together all of the remaining states that were not previously grouped together by union find.
7.) Take each of the disjoint sets and make a single state whose name is the state names from that set. EX {1, 3, 5} -> {135}
8.) Take the combo set and the other sets and make a list out of them.
9.) Take the DFA that was created before table filling algorithm and convert its structure into that of a NFA. Technically it's still a DFA, however the data structure it’s represented in is that of a NFA.
10.) Take the NFA structure and convert it into a DFA while bringing the list(holding the combo and non combo states with us).
11.) Check the combo next states that are suggested from the NFA at the current input symbol with the set of states that are inside one of the minimized DFA states. If the combo state set is inside the minimized DFA set we use the corresponding minimized DFA state instead of the combo state as the next state during the conversion.
12.) Enumerate through all the states in our newly minimized DFA. We are finished

Conclusion:

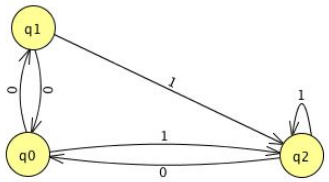
When presented with two DFA's we are now able to minimize both of them using the Myphill-Nerode, and then test the equivalence on the now minimized DFA's using the Hopcroft - Karp. Pictured below in stage 1 are 2 DFA's that are equivalent. You can see the compressed DFA's in stage 2.

Stage 1:

JFLAP 1

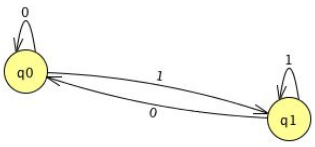


JFLAP 2

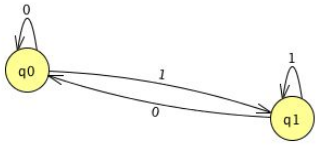


Stage 2:

JFLAP 1



JFLAP 2



Result of Equivalence

True

Problem

The goal of our project is to compare the efficiently compare the equivalency of two Deterministic Finite Automata (DFA), and then to minimize the DFA. Two DFA's are equivalent if they accept the same language.{{ add about minimizing the dfa a bit }

Solution Outline #1 (Equivalency):
For this problem we used the equivalence checking algorithm

There are 4 steps that we follow to test for equivalency

Step 1: For every state in the union of the two dfa's we need to make a set. This is O(n) time.

Step 2: We take the first states of the two DFA's and union them together. We then push that set on the stack. This is constant time.

Step 3: While the stack is not empty we pop the pair from the stack. For every A in the nonTerminal set we find the state where A goes and we need to find the st that that state belongs to for both of the states in the pair. If those sets are not equal, we union those two sets and push them on the stack. We repeat until the stack is empty.

Step 4: If no set contains both a final and a non final state we know that they are equivalent

Problem 2 (Minimization)

Minimization consisted of 4 steps

Step 1: We use Breadth First Search (BFS) and mark all reachable nodes in the graph.

Step 2: We now iterate through through graph and delete nodes that are not reachable.

Step 3: On the current graph we now run the table filling algorithm (TFS)

Table Filling Algorithm

1. We make a table whose length and width is n, we only look at (n^2)/2 nodes

2. We visit all the squares in the lower triangle. We visit all the squares, if one of the items in the square is an accepting state we mark that square. We do this for all squares. We visit all the squares again
3. {{{}}}

Step 4: Table Filling Is Finished - We go through the lower triangle and collect the coordinates that reference the cell that is not marked (x, y).

Step 5: We use union find to find all of these pairs and group them into disjoint groups.
Step 6: We now group together all of the remaining states that were not previously grouped together by union find.
Step 7: We take each of the disjoint sets and make a single state whose name are the state names from that set ex {1, 3, 5} -> {135}
Step 8: We now take the combo set, and the other sets and make a list out of them.
Step 9: We now take the DFA that we created before we ran the table filling algorithm and convert its structure into that of a NFA. Technically it's still a DFA, however the data structure its represented in is that of a NFA.
Step 10: We take this NFA structure and convert it into a DFA while bringing the list(holding the combo and non combo states) with us. We do this by setting it up
Step 11: We check the combo next states that are suggested from the NFA at the current input symbol with the set of states that are inside one of the minimized dfa states. If the combo state set is inside the minimized the dfa set. Then we use the minimized DFA state instead of the combo state as the next state during the conversion process
Step 12: We now make no more modifications to the NFA to DFA algorithm.
Step 13: We now enumerate through all the states in our newly minimized DFA. This converts the combo state strings into numbers.
Step 14: Done