COMS W3261
Computer Science Theory
Lecture 4
Equivalence of Regular Expressions and Finite Automata

Alexander Roth

2014-09-15

Outline

- McNaughton-Yamada-Thompson algorithm: RE to ε-NFA
- Kleene's algorithm: DFA to RE
- Converting a DFA to an equivalent RE by eliminating states
- A detailed proof that a DFA defines a given language

1 Review

- We say that two finite automata are equivalent if they define the same language.
- In the last two lectures we showed the subset construction could be used to convert either an NFA or an ε-NFA into an equivalent DFA.
- We say that a regular expression E and a finite automaton A are *equivalent* if $L(E) = L(A)$.
- In this lecture we first show how to construct an equivalent ε-NFA from a regular expression. Since we know that we can then use the subset construction to convert the ε-NFA into an equivalent DFA, we will then have shown that regular expressions are no more powerful in defining languages as a DFA.
- We then show that we can construct a regular expression that defines the same languages as a DFA.

- These results allow us to conclude that DFA's, NFA's, ε-NFA's, and regular expressions are all equivalent in definitional power – they all define precisely the regular languages.

Class Notes

- What form of finite automata is this? $\delta : Q \times \Sigma \to Q$, where $Q \times \Sigma$ is the Domain, and Q is the range. This is a DFA.
- $Q = \{q_0, q_1, q_2\}$, $P(Q) = 8sets$.
- If there are only a finite set of state in Q, then there is an exponential finite state in the power set of Q.
- NFA: $\delta : Q \times \Sigma \to P(Q)$.
- ϵ-NFA: $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to P(Q)$
- Regular expressions are an equivalent way of defining the regular languages.
- Check out Warthol's algorithm and Floyd's algorithm. They are examples of dynamic programming.
- We can prove that a regular expression recognizes the same strings as a finite automata.

2 McNaughton-Yamada-Thompson Algorithm: From an RE to an equivalent ε-NFA

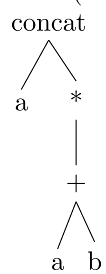
Theorem 1. *Let R be a regular expression R. Then we can construct an ε-NFA N such that $L(N) = L(R)$.*

Proof. See HMU, Sect. 3.2.3, pp. 102 – 107 □

- The proof is in the form of an algorithm that takes as input a regular expression R of length n and recursively constructs from it an equivalent ε-NFA that has
  - exactly one start state and one final state,
  - at most $2n$ states,
  - no arcs coming into its start state,
  - no arcs leaving its final state,
  - at most two arcs leaving any nonfinal state
- This algorithm was discovered by McNaughton and Yamada and then independently by Ken Thompson who used it in the string-matching program `grep` on Unix. On an input string of length m, an n-state MYT ε-NFA can be efficiently simulated in time $O(mn)$ using a two-stack algorithm.

Class Notes

Base cases for a regular expression: a, ϵ, and $\emptyset$. Here is an example: Let $R = a(a+b)^+$



Two Stack Algorithm

If there are n states, in the machine there can be n states in the stack. Sometimes this is called the Two Stack algorithm.

3 Kleene's Algorithm: From a DFA to an equivalent regular expression

- Given a DFA A, Kleene's algorithm constructs a regular expression R from A such that $L(R) = L(A)$.
- Suppose the states of A are numbered $1, 2, \ldots, n$.
- Kleene's algorithm is a dynamic programming algorithm that constructs a regular expressions $R[i, j, k]$ that denotes all paths from state i to state j with no intermediate node in the path numbered higher than k as follows:

```
for (i = 1; i <= n; i++)
  for (j= 1; j <= n; j++)
    if (i != j)
      if (there are transitions from state i to
          state j labeled a1, a2, ..., ak)
        R[i,j,0] = a1 + a2 + ... + ak;
      else
        R[i,j,0] = (emptyset);
    else if (i == j)
      if (there are transitions from state i to
          state i labeled a1, a2, ..., ak)
        R[i,i,0] = (emptystring) + a1 + a2 + ... + ak;
      else
        R[i,i,0] = (emptystring)
for (k = 1; k <= n; k++)
  for (i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
      R[i,j,k] = R[i,j,k-1] + R[i,k,k-1](R[k,k,k-1])*R[k,j,k-1];
```

- Assuming the start state is 1, the regular expression for the DFA A is then the sum (union) of all expressions `R[1,j,n]` where j is a final state.

- Note that Kleene's algorithm for constructing a regular expression from a DFA reduces to Warshall's transitive closure algorithm and to Floyd's all-pairs shortest paths algorithm for directed graphs.

- Example 3.5, HMU, pp. 95 – 97.

4 Converting a DFA to an equivalent RE by Eliminating States

- Kleene's algorithm gives us a mechanical way to construct a regular expression from a DFA, or for that matter, from any NFA or ε-NFA.

- Another approach that avoids duplicating work is to eliminate states, one at a time, from the DFA using the procedure outline in Section 3.2.2 of HMU.

- Example 3.6, HMU, pp. 101 – 102.

- You can see an expanded treatment of state elimination with more examples in pp. 583 – 588 of Chapter 10 of *Aho and Ullman, Foundations of Computer Science*.

5 A detailed proof that a DFA defines a given language

- To prove that a DFA D defines a given language L, we need to show that every string in $L(D)$ is in L and that every string in L is in $L(D)$. Here is a detailed example of how this can be done using induction for both parts.

- Consider the DFA $D = (\{\mathtt{A}, \mathtt{B}\}, \{0, 1\}, \delta, \mathtt{A}, \{\mathtt{A}\})$ in Example 1 from Lecture 2, Sep 8, 2014 where the transition function δ has the transition table:

State	Input Symbol	
	0	1
A	B	A
B	A	B

- Let L be the language consisting of all strings of 0's and 1's with an even number of zeros. We shall prove that $L(D) = L$.

- To begin, we shall show that every string accepted by D is in L, by proving the following inductive hypothesis by induction on n, the number of moves made by D accepting a string w, for $n \geq 0$:
  Inductive Hypothesis IH1:

  (a) If $\delta^n(A, w) = A$, then w has an even numbers of 0's. Here, δ^n means n moves by D.

  (b) If $\delta^n(A, w) = B$, then w has an odd number of 0's.

  - Basis. $n = 0$ which implies that $w = \epsilon$ and hence that w has an even number of 0's
  - Induction. Assume IH1 is true for 0, 1, 2, ..., n moves.
    - ∗ Suppose D makes $n+1$ moves on a string $w = x0$ and enters state A after reading x and then enters state B after reading the final 0. From the inductive hypothesis we know that x must have an even number of 0's. Therefore, $x0$ has an odd number of 0's.
    - ∗ Suppose D makes $n+1$ moves on a string $w = x0$ and enters state B after reading x and then enters state A after reading the final 0. From the inductive hypothesis we know that x must have an odd number of 0's. Therefore, $x0$ has an even number of 0's.
    - ∗ Suppose D makes $n+1$ moves on a string $w = x1$ and enters state A after reading x and then state A after reading the final 1. From the inductive hypothesis we know that x must have an even number of 0's. Therefore, $x1$ also has an even number of 0's.

- * Suppose D makes $n+1$ moves on a string $w = x1$ and enters state `B` after reading x and then state `A` after reading the final 1. From the inductive hypothesis we know that x must have an odd number of 0's. Therefore, $x1$ also has an odd number of 0's.

- We have now shown that IH1 is true for all sequences of n moves, where $n \geq 0$.

- We now need to show that every string in L is accepted by D. To do this, we shall prove the following inductive hypothesis by induction on n, the length of a string w, for $n \geq 0$:
  Inductive Hypothesis IH2:

  (a) If w has an even number of 0's, then $\delta^*(A, w) = A$.

  (b) If w has an odd number of 0's, then $\delta^*(A, w) = B$

  - Suppose w is now a string of length $n+1$ and $w = x0$ and x has an even number of 0's. From IH2, $\delta^*(A, x) = A$. Since $\delta(A, 0) = B$, $\delta^*(A, w) = B$.
  - Suppose $w = x0$ and x has an odd number of 0's. From IH2, $\delta^*(A, x) = B$. Since $\delta(B, 0) = A$, $\delta^*(A, w) = A$.
  - Suppose $w = x1$ and x has an even number of 0's. From IH2, $\delta^*(A, x) = A$. Since $\delta(A, 1) = A$, $\delta^*(A, w) = A$.
  - Suppose $w = x1$ and x has an odd number of 0's. From IH2, $\delta^*(A, x) = B$. Since $\delta(B, 1) = B$, $\delta^*(A, w) = B$.

- We have now shown that IH2 is true for all strings of length n, $n \geq 0$.

- From the two inductive hypotheses, we can conclude that w is accepted by D iff w is in L. In other words, $L(D) = L$.

6 Practice Problems

1. Use the MYT algorithm to construct an equivalent ε-NFA for the regular expression $a + b * a$.

2. Show the behavior of your ε-NFA on the input string `bba`.

3. Use the subset construction to convert your ε-NFA into a DFA.

4. Show the behavior of your DFA on the input string `bba`.

5. Consider the DFA D with:

   1. $Q = \{$`1, 2, 3`$\}$
   2. $\Sigma = \{$`a, b`$\}$

3. δ:

State	Input Symbol	
	a	b
1	2	1
2	3	1
3	3	2

4. Start state: 1
5. $F = \{3\}$

   a) Use Kleene's algorithm to construct a regular expression for $L(D)$. Simplify your expressions as much as possible at each stage.

   b) Construct a regular expression for $L(D)$ by eliminating state 2.

7 Reading Assignment

- HMU: Ch 3