Finite Automata

https://courses.missouristate.edu/anthonyclark/333/

Some examples adapted from Professor Wayne Goddard at Clemson university
Outline

**Topics and Learning Objectives**
- Discuss finite automata theory

**Assessments**
- Finite automata
Chomsky Hierarchy

- Type-0: Turing machine
- Type-1: Linear bounded automaton
- Type-2: Pushdown automaton
- Type-3: Finite state automaton
Automata Theory

- An automaton consists of states and transitions
  - Some states are referred to as accept states
- An automaton consumes an input sequence of symbols
  - An automaton only accepts an input string if it ends in an accept state
- Transitions to different states depend on the current state and the current input symbol
- Finite automata have finite memory
- An automaton is a device that recognizes a language
- We are only going to scratch the surface
Finite Automata Procedure

1. Go to the start state
2. Try to read a symbol from the input
   I. If we could not read any more symbols (reached the end) then go to step (3)
   II. Transition to the next state based on the current state and the input
   III. Repeat step (2)
3. If we are in any accept state then return “accept” otherwise return “reject”
Example

Possible input symbols: \{0, 1\}

What is the final state if you are given the string: 101001?
What is the final state if you are given the string: 11101?
skip any initial white space (spaces, tabs, and newlines)
if cur_char ∈ {‘(’, ‘)’, ‘+’, ‘-’, ‘*’}
    return the corresponding single-character token
if cur_char = ‘:’
    read the next character
    if it is ‘=’ then return assign else announce an error
if cur_char = ‘/’
    peek at the next character
    if it is ‘*’ or ‘/’
        read additional characters until “*/” or newline is seen, respectively
        jump back to top of code
    else return div
if cur_char = ‘.’
    read the next character
    if it is a digit
        read any additional digits
        return number
    else announce an error
if cur_char is a digit
    read any additional digits and at most one decimal point
    return number
if cur_char is a letter
    read any additional letters and digits
    check to see whether the resulting string is read or write
    if so then return the corresponding token
    else return id
else announce an error
Building FAs

- Start with what is obvious
- For example: “accept all strings that have two consecutive zeros”
Building FAs

- Start with what is obvious
- For example: “accept all strings that have two consecutive zeros”
- Now handle all other cases
Examples (work in groups on back of quiz)

1. Create an automaton that accepts all strings of 0’s and 1’s with an odd number of 1’s
2. Create an automaton that accepts all strings that start with 00
3. Create an automaton that accepts all strings that end with 00
4. Create an automaton that accepts alternating 0’s and 1’s
Odd 1s
Start with 00
End with 00
Alternating
1. Create an automaton that accepts all strings of 0’s and 1’s with an odd number of 1’s

2. Create an automaton that accepts all strings that start with 00

3. Create an automaton that accepts all strings that end with 00
Alternating 0’s and 1’s
How to FAs relate to REs?

• There is a one-to-one mapping between finite automata and regular expressions

• Write a regular expression that recognizes all binary strings with at least 2 digits that begin and end in with the same symbol.

• Draw an FA that accepts the same strings
Accept all binary strings with at least 2 digits that begin and end in with the same symbol.
Nondeterministic Finite Automata (NFA)

• NFAs are similar to what we’ve already discussed, but you can have zero or more transitions from a state based on a given input symbol

• NFAs can also have empty transitions (ε-transitions), which allow an NFA to change state without consuming an input symbol

• NFAs have multiple branches, and a branch dies only if it cannot transition given the next symbol

• An NFA accepts the input string if any branch ends in an accept state
What makes this non-deterministic?

Possible input symbols: \{0, 1\}

What does this NFA accept?

What is the final state if you are given the string: 1010011?
What is the final state if you are given the string: 11101?
What does this NFA accept?

Consider the input input 10100
Another example NFA

What makes this an NFA?

What does this NFA accept?
Write a Regular Expression Matching the NFA

Looks like an OR operation (alternation)

Looks like “zero or more” (Kleene Star)

Looks like an AND operation (concatenation)
DFA vs NFA

- Transition from a state to a single next state for each input symbol
- No empty transitions
- An input string is accepted if the DFA ends in an accepting state

- Transition to multiple next states for each input symbol
- Empty transitions allowed
- An input string is accepted if any branch of the NFA ends in any of its accepting states
Formal definition of a Finite Automaton

An FA can be denoted as \((Q, \Sigma, q_0, T, \delta)\)

- \(Q\) is the set of states
- \(\Sigma\) is the alphabet of input symbols
- \(q_0\) is the start state
- \(T\) is the set of accept states (subset of \(Q\))
- \(\delta\) is the transition function
Counting/Matching

Write a regular expression or draw a FA (DFA or NFA) to do the following:
• Accept all strings that have balanced parenthesis

) (  
(  )  
( ) ( ( ) ) ( )  
( ) ( ) ( ( ) ) ( )  
( ) ( ) ( ( ) ) ( ) ( )
A Key Point

• An automaton cannot **count**

• Try to create an automaton that matches strings with some number of 0’s followed by the same number of 1’s (01, 0011, 000111)

• Your **finite** state automata would need an **infinite** number of

• An important example to remember why this is important: an FA cannot even perform bracket matching

• So, we’ll need something more powerful for **parsing**
Bottom Line

• Why are we discussing finite automata?
• They can be used to recognize regular languages

• So, your scanner implements a finite automaton (FA)
• But, your parser will not

• Parsers require, at minimum, a pushdown automaton (PDA)