Overview

• Regular Expressions
• State Machines
Objective

• Understand how pattern matching can be performed with regular expressions
• Learn how state machines can be used to implement regular expressions

Regular Expressions

• Method to describe patterns of text
  • Character-by-character processing
  • Special operators
    • | (alternatives)
    • . (arbitrary character)
    • * (zero or more repetitions)
    • + (one or more repetitions)
    • ( ) (precedence)
    • …
Regular Expressions

• Examples
  • abcd
    • abcd matches; aabcd does not match
  • a*bcd
    • aabcd matches; bcd matches; cd does not match
  • (ab|bb)cd
    abcd matches; bbcd matches; abbbcd does not match
  • (ab|bb)*cd
    abbbcd matches; bbabcd matches; cd matches; ababcd matches; abbb does not match

Use of Regular Expressions

• Compiler
  • Interpreting characters in program
  • Regular expressions for numbers, keywords, etc.
  • Example tool: flex

• Networking
  • Checking network traffic for attacks
  • Regular expressions for attack patterns
  • Example tool: snort database
More Examples for Regular Expressions

- Examining command lines
- Parsing user input
- Parsing various text files
- Examining web server logs
- Examining test results
- Finding text in emails
- Reading configuration files

More Examples for Regular Expressions

- `^[a-zA-Z0-9]+[0-9]$

- `^[a-zA-Z]+[0-9]$

- `^[a-zA-Z0-9]+$`
Limits of Regular Expressions

- Regular expression match patterns from “regular language”
- Regular expression cannot describe patterns from more complex language
  - What can you not describe with a regular expression?

Limits of Regular Expressions

- Regular expression match patterns from “regular language”
- Regular expression cannot describe patterns from more complex language
  - Context-free grammars
    - Equal number of opening and closing parentheses
  - Context-sensitive grammars
    - Grammatically correct English language
State Machine

- Regular expression can be matched with a state machine (or finite automaton)
- State machine is special case of directed graph
  - Nodes represent state
  - Edges represent transitions (based on input)
- State machines can be constructed for any kind of regular expression

State Machine Examples

- Example 1: ac+|bd

- Example 2: a(b*|c)d
Deterministic vs Non-Deterministic

• What is the problem with $(ab)^*ac$?

• Non-deterministic transition on a

• Non-deterministic state machines
  • A bit more complex to implement
  • We do not consider them here
  • There exist algorithms to convert from NFA to DFA
Deterministic vs Non-Deterministic

• list/lost/lust

![Diagram](image)

Deterministic vs Non-Deterministic

• list/lost/lust

![Diagram](image)
Deterministic vs Non-Deterministic

- list/lost/lust

![Diagram](image)

Deterministic vs Non-Deterministic

- list/lost/lust

![Diagram](image)
Implementing a State Machine

- Vertex with multiple outgoing edges
  - Need class to represent edge
  - Need linked list to store edges
- Matching operation
  - Start at start node
  - Follow edge that matches character
  - At end, check if accepting state
  - If no edge or no accepting state, then no match

Edge Class

class Edge:
    def __init__(self, c, dest):
        self.destination = dest
        self.character = c
**Vertex Class**

class Vertex:
    def __init__(self, n):
        self.number = n
        self.edgeList = []
        self.isAcceptingState = None

    def setAcceptingState(self):
        self.isAcceptingState = True

    def addEdge(self, e):
        self.edgeList.append(e)

    def followEdge(self, c):
        for i in self.edgeList:
            if i.character == c:
                return i.destination
        return None

**Matching Method**

class DFA:
    def __init__(self, s):
        self.start = s

    def match(self, s):
        self.characters = list(s)
        self.current = self.start
        print("trying to match "+s+": ")

        for i in self.characters:
            if self.current == None:
                print("no match")
                return
            print(self.current.number + " ")
            self.current = self.current.followEdge(i)

        if self.current == None:
            print("no match")
            return

        print(self.current.number);
        if self.current.isAcceptingState:
            print("match")
        else:
            print("no match")
        return
Creating DFA and Pattern Matching

```python
v1 = Vertex(1)
v2 = Vertex(2)
v3 = Vertex(3)
v4 = Vertex(4)
v5 = Vertex(5)
v6 = Vertex(6)
v4.setAcceptingState()
v6.setAcceptingState()
v1.addEdge(Edge("a", v2))
v1.addEdge(Edge("b", v1))
v2.addEdge(Edge("b", v3))
v2.addEdge(Edge("a", v5))
v3.addEdge(Edge("c", v4))
v5.addEdge(Edge("b", v6))
v6.addEdge(Edge("c", v6))
v4.addEdge(Edge("d", v1))

stateMachine = DFA(v1)
stateMachine.match("abc")
stateMachine.match("bbabc")
stateMachine.match("baab")
stateMachine.match("baabcc")
stateMachine.match("abcdcbbaacc")
stateMachine.match("abcd")
stateMachine.match("e")
```

- **Graph:**
- **Matching:**
  - abc
  - bbabc
  - baab
  - baabcc
  - abcdcbbaacc
  - abcd
  - e
Next Steps

- Next lecture and on Thursday
- Project 2 due on 11/11