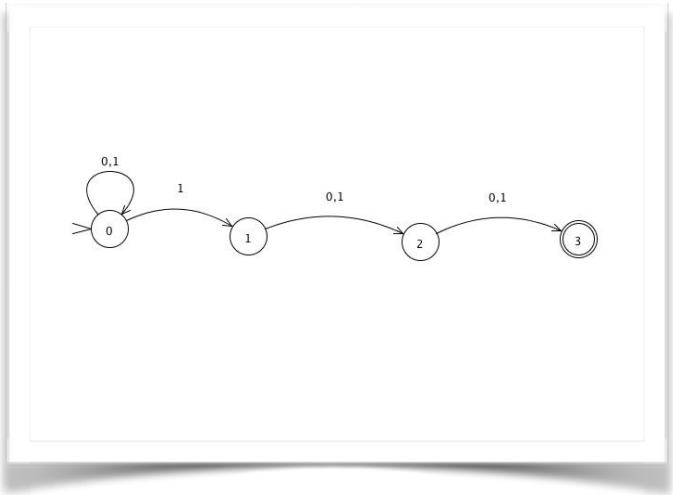
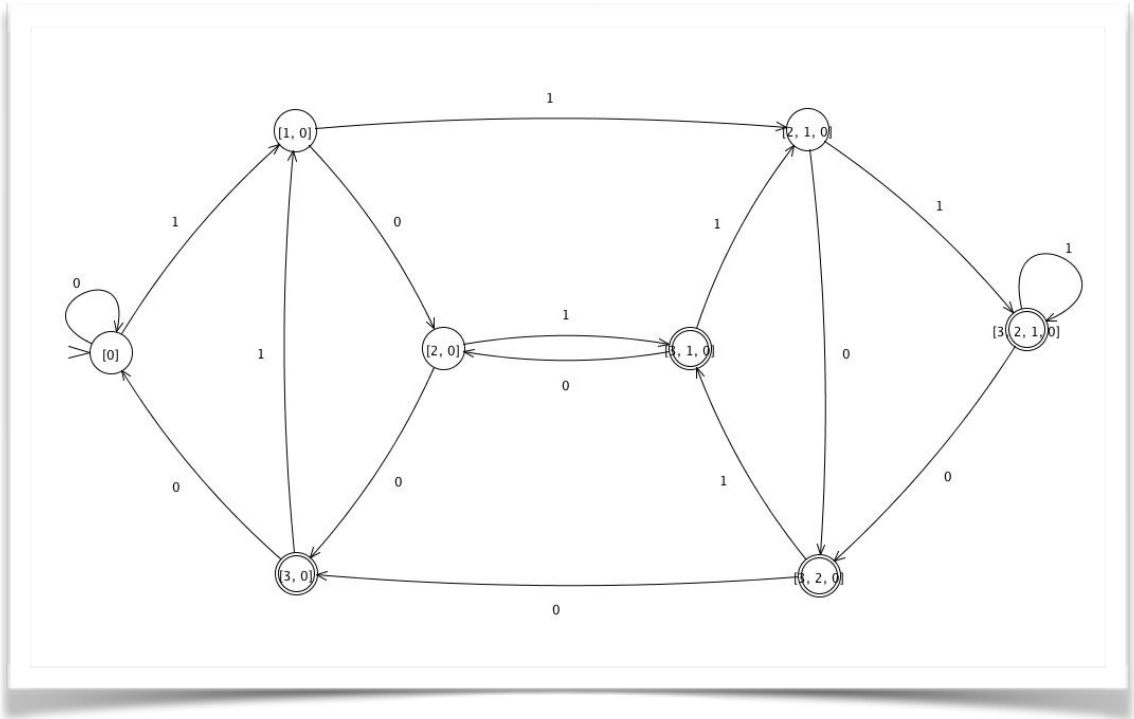
COMPLEXITY PROJECT

Discrete Math
Project 3





Complexity Project

Objective

Use our library and other appropriate sources to explore the limits of computability.

Introduction

Modern computers are deterministic machines. Given an input, an exact chain of events is set in motion. The machine goes through a predetermined path of states and transitions (follows an algorithm). While different from modern computers, a Turing Machine is an abstraction of the processes computer can perform. There is a notion of a Turing machine that is **not deterministic** which would be revolutionary if implemented with a quantum computers.

A. Research and answer the following questions:

1. Define **tractable** and **intractable** (*in regards to computational complexity*)
2. What is the complexity class **P**?
3. What is the complexity class **NP**?
4. Give an example of an **unsolvable** problem.
   note: P = NP is an unsolved (as of the time of this project), not an unsolvable problem.

B. Research Finite State Machines. Answer the following questions:

5. Compare and contrast non-deterministic FSM (**NFSM**) and deterministic FSM (**DFSM**).
   - Do they have the same "power"?
   - Is there a difference in the class of languages these machines decide?
   - What theorem/thesis/important work proves this result?
6. The FSM examples on the cover page receive bit strings. What words belong to the language they decide? (experiment with the examples in the **FSM-CoverExamples.zip** file)

Reminder: The set of possible inputs is the "alphabet".
A sequence of string of input is a "word"
A set of related words is the "language"

C. Form an Opinion

7. Do you think the classes P and NP have equal "power" or do you think one class contains harder problems? Explain the reasoning behind your stance. How does it compare with your answer to #5?

D. Modeling Finite State Machines:

Go to http://www.cburch.com/proj/autosim/download.html and download the AutoSim.jar. We will build some finite state machines. Open the zip folder you downloaded. Run AutoSim.jar.

8. Open the file 'xaaa or aaax'. Create a diagram for a DFSM using only {a,b} as your alphabet that accepts all strings that begin with aaa OR end with aaa.
9. Open the file 'Missing a Letter' Create a diagram for an NFSM that accepts strings over {a, b, c} where at least one letter is missing. *You must use epsilon transitions.*
10. In the remaining files, create a DFSM and an NFSM of your choice. *These machines must not decide the same language, and the NFSM must not be a DFS*