COMP 102: Computers and Computing

Lecture 4: Finite State Machines and Memory

Instructor: Kaleem Siddiqi (siddiqi@cim.mcgill.ca)

Class web page: www.cim.mcgill.ca/~siddiqi/102.html

What have we seen so far?

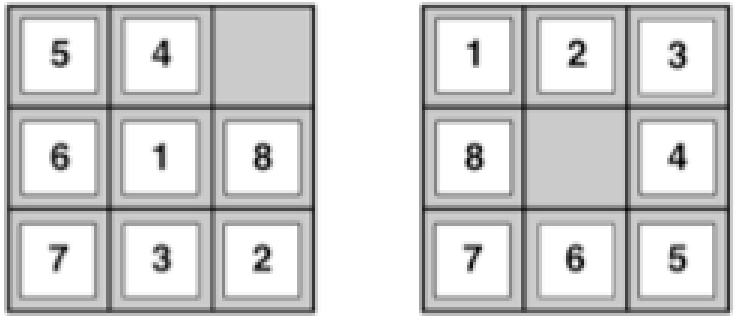
- Representing many types of data (text, numbers, images, sound) using binary representations.
- Solving problems using binary variables, logical expressions, truth tables, and logic gates.

Any limitations to this?

- So far we have assumed that the state of the logical variables stay constant over time. E.g. Reset the computer between each round of Rock-Paper-Scissors.

Today: How can we represent any configuration in memory?

The eight-puzzle



Start state Goal state

- Need to define sequences of moves to go from start configuration to goal configuration.
- Need to represent any configuration in memory.
- BUT! Need a way to describe how the configuration (a.k.a. the state) changes over time.

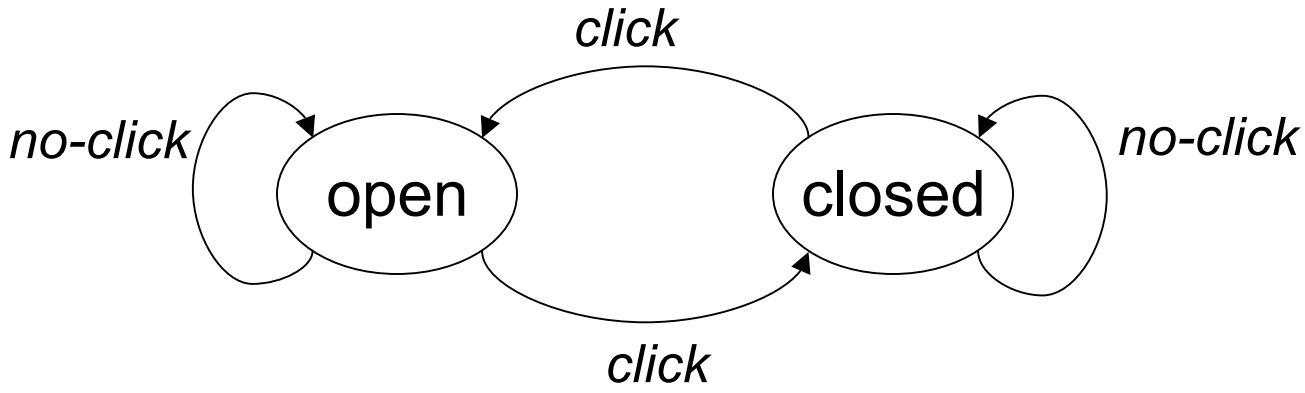
Other examples

- Retractable ballpoint pen
- Automatic door
- Traffic-light control
- Combination lock
- Elevator control system
- Language generation

What do these things have in common?

- The state changes over time.
- Need memory to capture the state of the system at any point in time.

Ballpoint pen



Can you think of a finite state machine for a sequential parity checker, i.e., a finite state machine that determines whether the total number of binary digits in a sequence, where the bits are received sequentially, is odd or even?

Eight-puzzle

- How many possible states are there?
- What would the inputs (or observations) be?
- What would the actions (or transitions) be?

Finite State Machine

- The Finite State Machine combines a look-up table (constructed with binary logic) with a memory device (to store the state).

- Components of the finite state machine:
  - Set of states: $S = \{s_1, s_2, \ldots, s_n\}$
  - Set of observations: $O = \{o_1, o_2, \ldots, o_n\}$
  - A set of transitions which describe the movement from state to state in response to the observation or observations seen.

Transition function

- For each state and observation, need to know what is the next state.
- Denote this as: ***T(s,o) = s'***

where ***s*** is the current state of the device

o is the most recent observation (or observations)

s' is the next state of the device

Combination Lock

- **State** = Summary of the sequence of numbers.
- **Observation** = Number entered.
- **Memory** = Not all numbers ever dialed need to be stored, but need to remember enough about recent numbers to know if the sequence opens the lock.

Traffic-light controller

- **State** = 2-way car traffic light
  - S = {red, yellow, green}$_{\text{direction 1}}$ x {red, yellow, green}$_{\text{direction 2}}$
  - S = {red-red, red-yellow, yellow-red, red-green, green-red}
  - Not all configurations of lights are included (e.g. green-green) at least one light must be red.

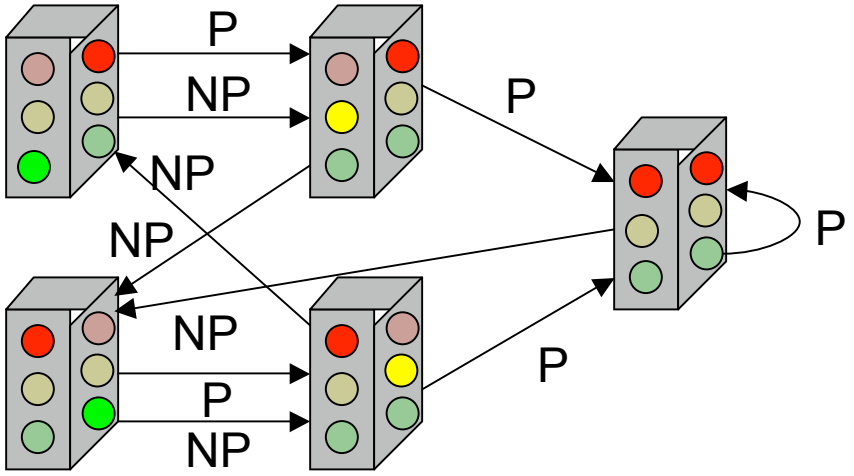
- **Observation** = pedestrian light request
  - O = {pressed, not pressed}

- **Memory** = Current state of the light in both directions.

Traffic-light controller: Transition function

- **State** = 2-way car traffic light
  - S = {red-red, red-yellow, yellow-red, red-green, green-red}
- **Observation** = pedestrian light request for both streets
  - O = {pressed, not pressed}
- **Transition function**: T(current state, observation) = next state

Here is one way to show the transition function as a graph.

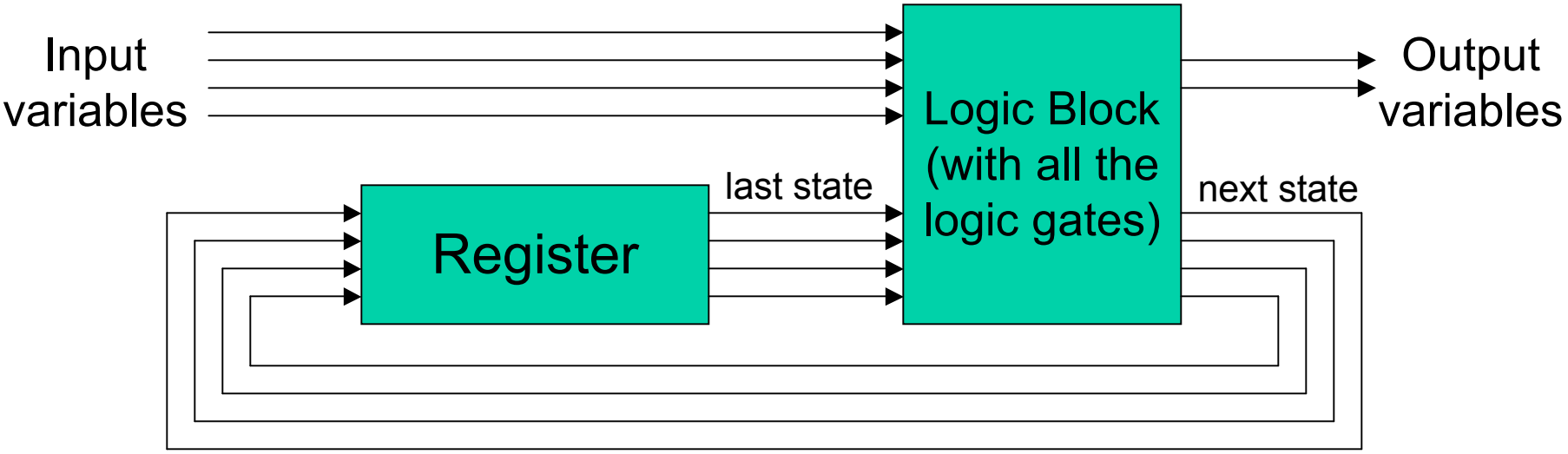


Traffic-light controller: Transition table

State	Observation	Next state
Green-Red	Pressed	Yellow-Red
Green-Red	Not-Pressed	Yellow-Red
Yellow-Red	Pressed	Red-Red
Yellow-Red	Not-Pressed	Red-Green
Red-Green	Pressed	Red-Yellow
Red-Green	Not-Pressed	Red-Yellow
Red-Yellow	Pressed	Red-Red
Red-Yellow	Not-Pressed	Green-Red
Red-Red	Pressed	Red-Red
Red-Red	Not-Pressed	Red-Green

- This can be seen as just a standard truth table.
- Simply need to pick logical variables for the states and observations.
- Only subtlety: Need to use same set of variables for the "State" and "Next state" variables.
- Also need to store those variables between time steps.

Storing the state of a finite-state machine



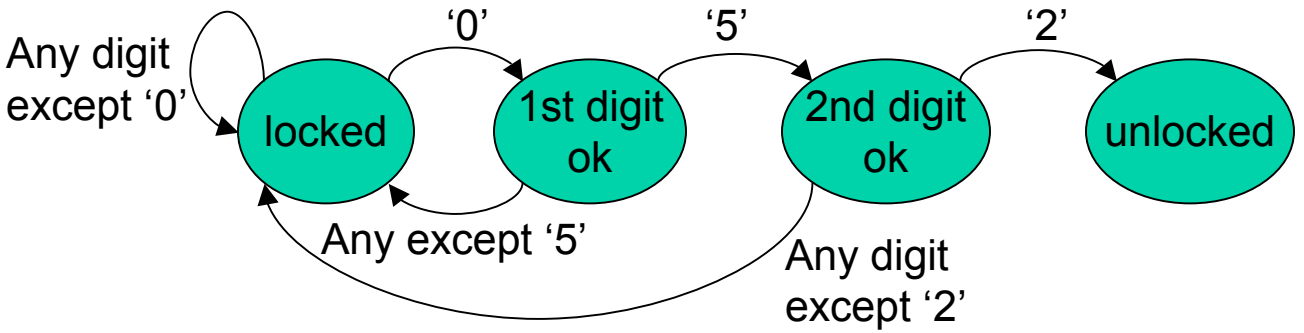
- Observations are set through inputs.
- Register is used to store bits. It has an additional timing input that tells it when to change state (*think of a clock that 'ticks' every second*).
- The logic block implements the transition function.

To implement a finite-state machine (FSM)

1. Select the set of states and the set of observations.
2. Choose a different pattern of bits to each state
   - In traffic light example: need 3 bits for state, 1 bit for observation.
3. Generate transition table or transition graph.
4. Implement transition table using logic gates.
   - Input = bits representing the observation and the last state.
   - Output = bits representing the next state.

Recognizing sequences with a FSM

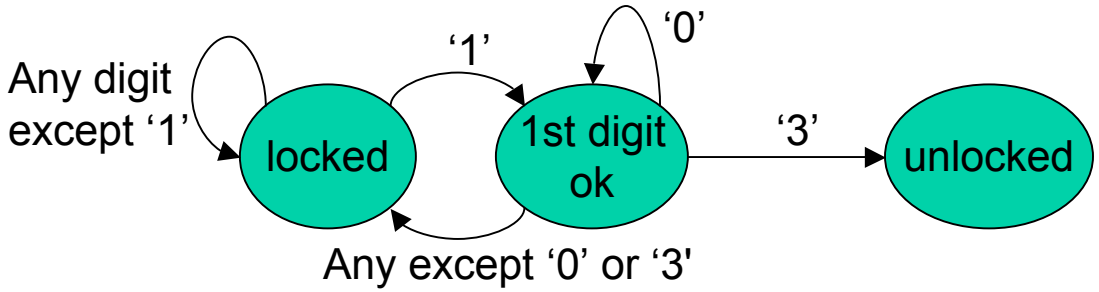
- Recall: Combination Lock
  - Assume the lock opens only when it sees sequence **0 - 5 - 2**.
- States: Number of digits in the sequence that have been recognized already = {0, 1, 2, 3, done}
- Observations: Digits that can be selected = {0, 1, …, 9}
- Transition function:



Recognizing sequences with patterns

- Consider a lock which recognizes sequences that start with “1”, have any number of “0”’s, and end with “3”.

- Transition function:



Which sequences can be recognized?

- Can recognize any pre-specified sequences of numbers or letters of a finite length.
  - E.g. Misspelled word within a stream of text.

- Cannot recognize all types of patterns.
  - E.g. Cannot build a finite-state machine that unlocks a lock whenever you enter any palindrome: 3-2-1-1-2-3
  - Why? Palindromes can be of any length, and to recognize the 2nd half, you need to remember every character in the first half.

    Because there are infinitely many possible first halves, this would require a machine with an infinite number of states.

Other tasks we cannot do with an FSM

- Problems with non-deterministic transitions, e.g. backgammon.
- Problems where we don't know the set of state/observations in advances.
- Problems where the transitions change over time.

Take-home message

- Finite-state machines let us reason about functions that change over time.
- Understand the components of finite state machines (state, observation, transition) and the implementation steps (transition table, logic encoding through gates, use of registers)
- Understand what tasks can and cannot be done with a FSM.

Final comments

- Some material from these slides was taken from:
  - *http://www.cs.rutgers.edu/~mlittman/courses/cs442-06/*