(Finite) State Machines

Lecture 22

Several Models of Computation

- Automata/Machines, Algebras/Calculi, Grammars, ...
- A few examples we shall see:
  - (Finite) State Automata
  - (Context Free) Grammars
  - Circuits
  - You already saw (implicitly): "Random Access Machine"
- Today: States (and automata)

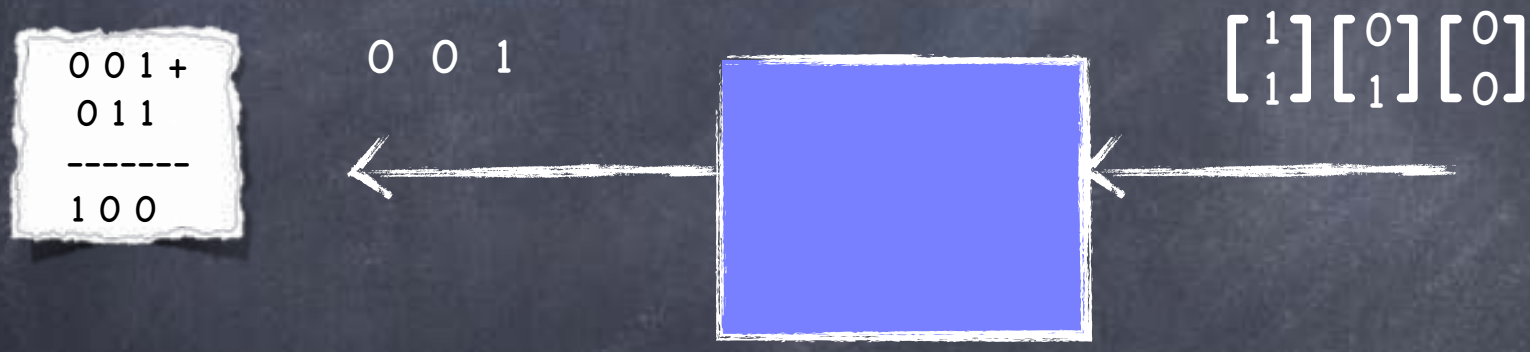
State

- Consider a (discrete) system which takes a stream of inputs and produces a stream of outputs (a "transducer")

- The system's output at any moment depends not only on the "current" input but also on what the system "remembers" about the past
  - State of the system: what is in the system's memory
- The number of possible states could be finite or infinite (for e.g. if the system remembers the sequence of inputs seen so far, or even just the number of inputs so far)

State Diagram

- A graph with nodes as the states and arcs from a state to another if the system can make that transition in one step
- e.g. A system in which the inputs are pairs of binary digits (Least Significant Bit first) and the outputs are the digits of their sum



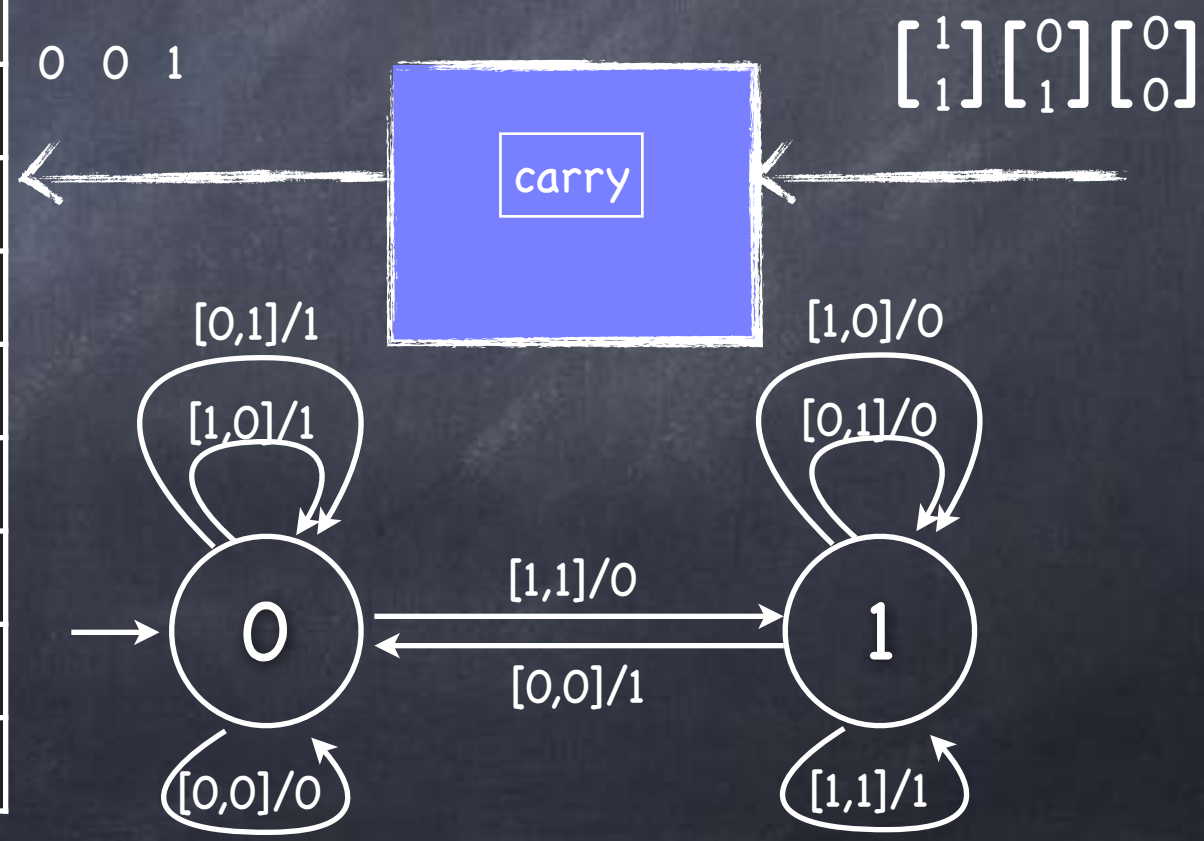
- What should the system remember?
  - The "carry": a single bit
  - State diagram has two nodes

State Diagram

- Initially carry is 0
- If carry is 0, and input is [0,0], then output is 0
  - And carry remains 0
- If carry is 0, and input is [1,1], then output is 0, but new carry is 1

...

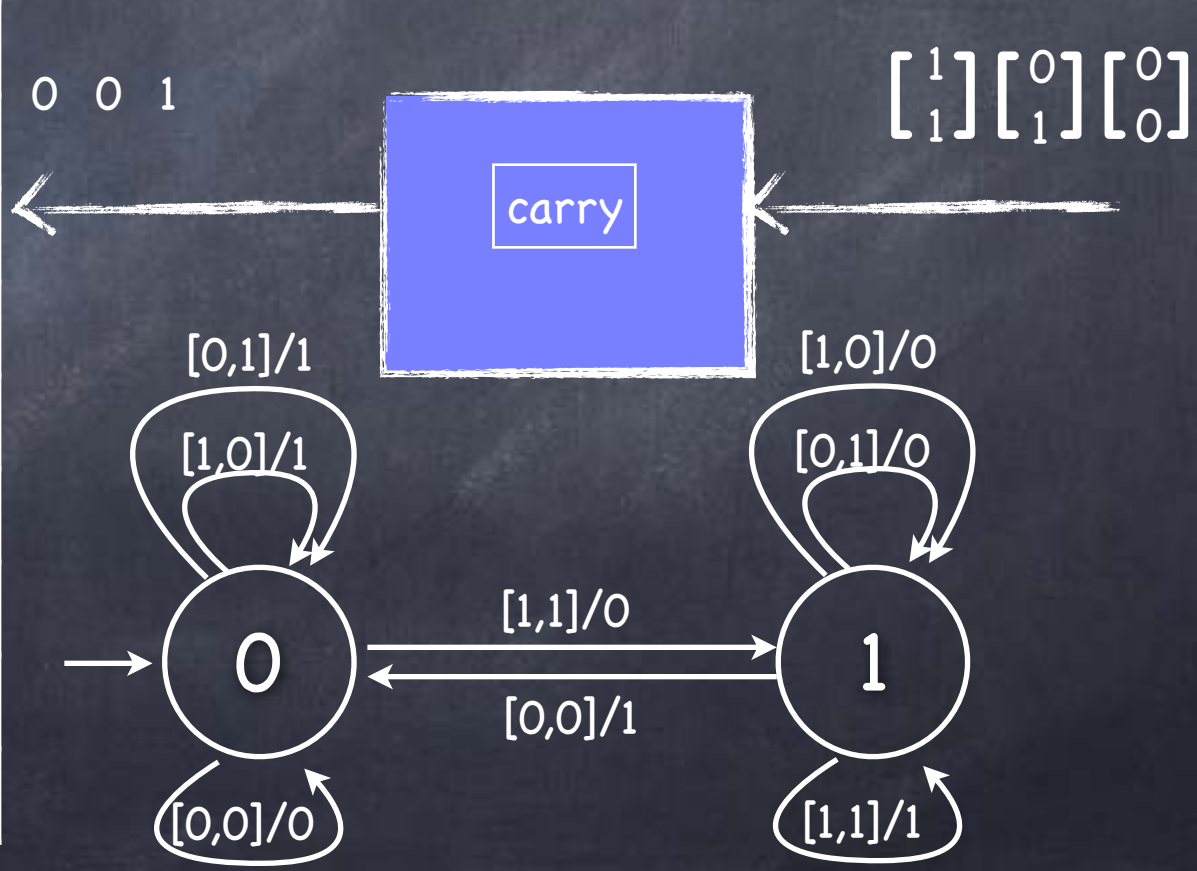
carry	input	output	new carry
0	[0,0]	0	0
0	[0,1]	1	0
0	[1,0]	1	0
0	[1,1]	0	1
1	[0,0]	1	0
1	[0,1]	0	1
1	[1,0]	0	1
1	[1,1]	1	1



State Diagram

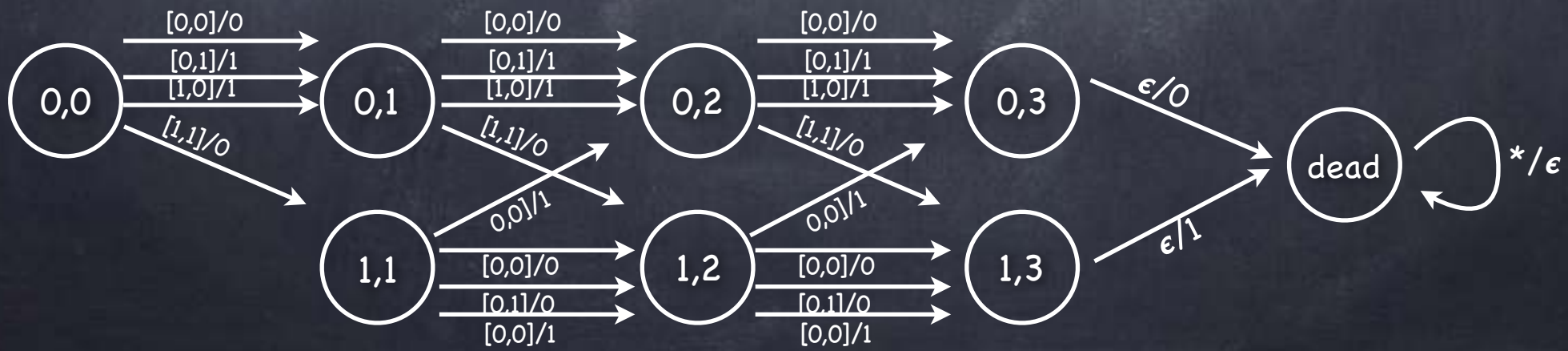
- Transition function: maps (state,input) pairs to (state,output) pairs
  - $\delta_{deterministic}$: $S \times \Sigma_{in} \longrightarrow S \times \Sigma_{out}$ (S: state space, Σ: "alphabet")
  - Deterministic: given a state and an input, the system's behavior on next input is completely determined

carry	input	output	new carry
0	[0,0]	0	0
0	[0,1]	1	0
0	[1,0]	1	0
0	[1,1]	0	1
1	[0,0]	1	0
1	[0,1]	0	1
1	[1,0]	0	1
1	[1,1]	1	1



Another Example

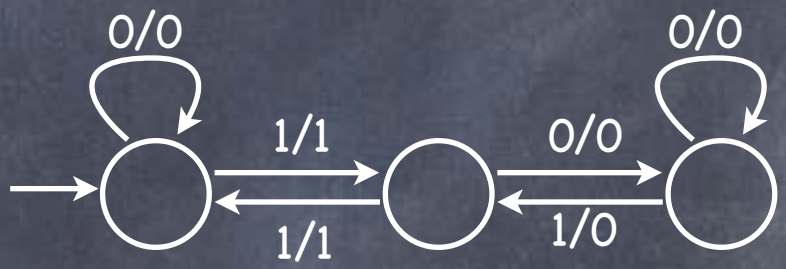
- Binary addition for 3 bit numbers
  - In the previous example, the answer is complete only if carry is 0 (can enforce by feeding [0,0] as a last input)
  - Here, accepts only up to 3 bits for each number, and produces a 4 bit output
  - State space?
    - Need to remember carry, and number of inputs seen so far



Question

RFYQ

- On giving which of the following strings as input does this transducer give a different string as output



A. 100
B. 0100
C. 0011010
D. 1110110
E. 1100011

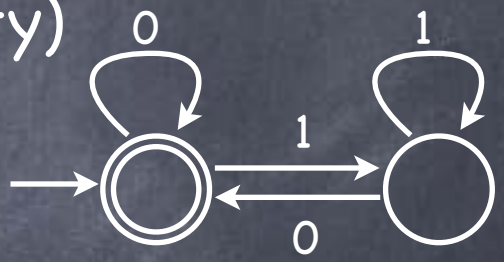
(0*11)* **10** 0* **1** (0|1)*

Acceptors

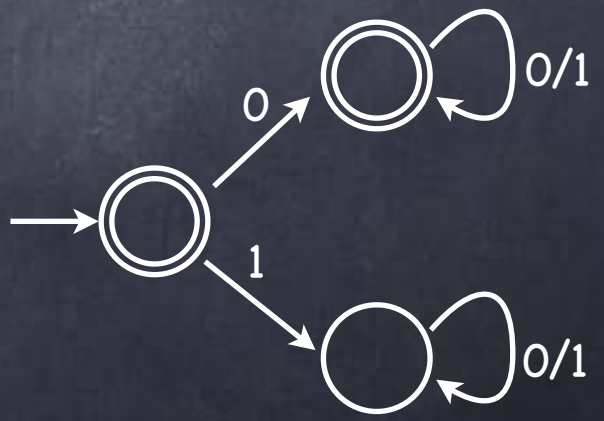
- The machines we saw are deterministic transducers
  - Converts an input stream to an output stream
- Acceptors don't produce an output stream
  - At the end of input, either "accepts" or "rejects" the input. Indicated by the state it is in at that point.
  - Accepting states are called final states
  - Transition function: $\delta_{\text{det-acceptor}} : S \times \Sigma \rightarrow S$

An Example

- Input: a number given as binary digits, MSB first. Accept iff the number is even (or empty)
  - Just remember the last digit seen



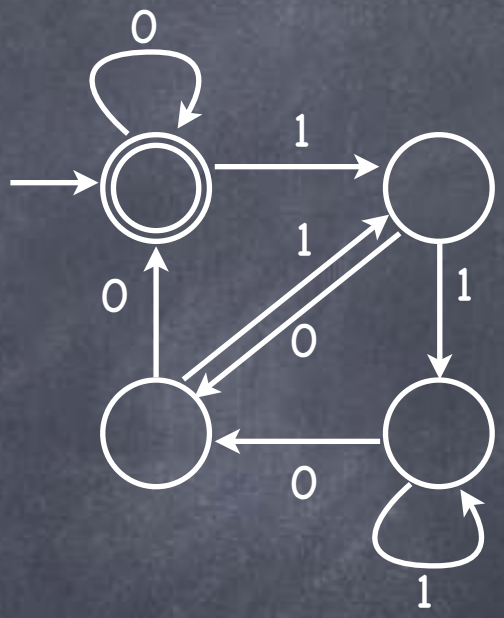
- What if input is given LSB first?
  - Remember the first digit seen



LQXQ

Question

- Which of the following strings does this acceptor accept?

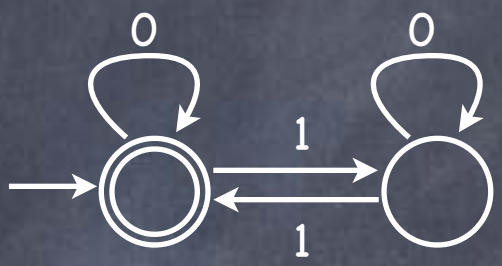


A. 0101
B. 1001
C. 1010
D. 1100
E. None of the above

BPKQ

Question

- Which of the following strings is <u>not accepted</u> by this acceptor:

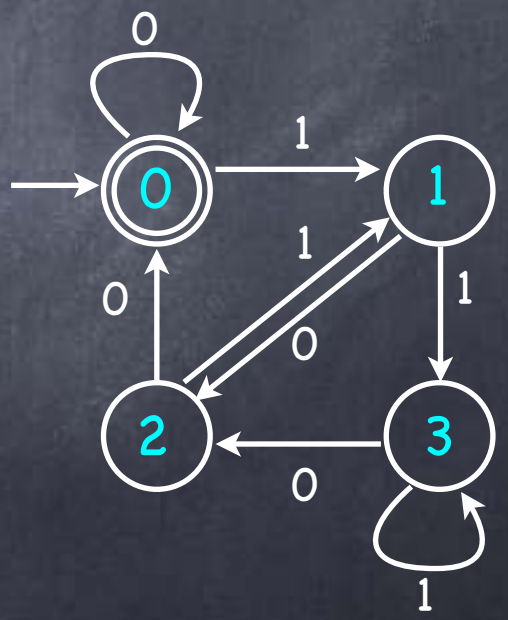


A. ϵ (empty string)
B. 101
C. 001000110
D. 1011001
E. 10000001

Odd number of 1s

An Example

- Input: a number given as binary digits, MSB first. Accept iff the number is divisible by d (or empty)
  - Just remember remember x (mod d), where x is the number seen so far.
  - Next number x′ is 2x or 2x+1 depending on the current input bit.
    - x′ (mod d) is determined by x (mod d)



A Variant

- Input: a number given as binary digits, **LSB first.**
  Accept iff the number is divisible by d (or empty)
  - To remember x (mod d), where x is the number seen so far.
  - Next number x' = ?
  - $x' = x + b.2^n$, where n bits seen so far, and $b \in \{0,1\}$ is the next bit
    - But we can't "remember" n.
    - Enough to remember 2^n mod d (along with x mod d)

Counting Number of States: An Example

- Game of Nim:
  - 2 piles of matchsticks, with T matchsticks each.
  - Each round a player removes one or more matchsticks from one pile.
  - Alice makes the first move.
- What are the states?
  - (|pile$_1$|, |pile$_2$|, next-player)
- Number of such states? $2(T+1)^2$
- Number of <u>reachable</u> states? $2(T+1)^2 - 4$

(T,T,Bob)
(T,T-1,Alice)
(T-1,T,Alice)
(T-1,T-1,Bob)
are unreachable

Finite-State Machines

- Many sets of strings have finite-state acceptors
  - e.g., numbers divisible by d, LSB first, or MSB first; strings matching a "pattern" like 0*10*10* (strings with exactly two 1s)
  - Can run on arbitrarily long inputs without needing more memory
- Many interesting sets of strings do not have finite-state acceptors
  - e.g., strings with equal number of 0s and 1s, palindromes, strings representing prime numbers, ...
  - How do we know they don't have finite-state acceptors?
    - If only finite memory, can come up with two input sequences which result in same state, but one to be accepted and one to be rejected
    - Later (in CS 310)

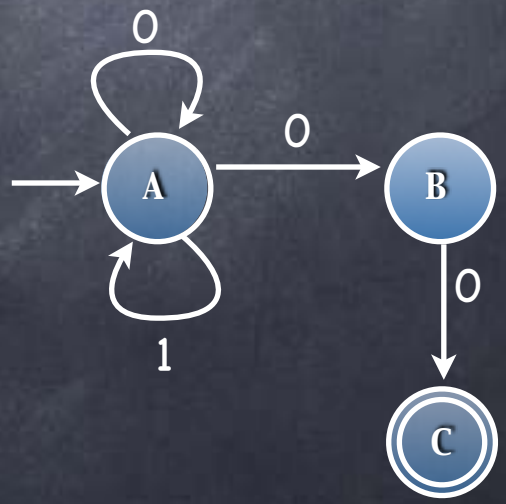
Non-determinism

- At a state, on an input, the system could make zero, one or more different transitions
- $\delta_{\text{nondet-acceptor}} : S \times \Sigma \rightarrow \mathbb{P}(S)$
  - δ (s,a): At a state s, on input a, what is the set of all the states to which the system can transition
- System's behavior not necessarily fixed by its state and input
- Sometimes probabilistic machine: Non-deterministic machine + probabilities associated with the multiple transitions

An Example

- At a state, on an input, the system could make zero, one or more different transitions
- $\delta_{\text{nondet-acceptor}} : S \times \Sigma \rightarrow \mathbb{P}(S)$
- Accept only strings which end in 00
- Example string: 0100
  - Note: δ (B,1) = $\varnothing$ (no where to go!)

0100



Representing a Finite-State Machine

- If your program uses only a constant amount of memory (irrespective of how large the input (stream) is) then it is a finite state machine
- But often useful to explicitly design a finite state machine (identifying all its states/transitions), and then implement it
  - To represent the transition function of a deterministic acceptor, a look-up table mapping (state,input) pair to a state
  - But if sparse - i.e., for many states, many inputs lead to a "crash state" (which is left implicit) - it is more space-efficient to simply list valid (state, input, next state) tuples
    - This would slow down look-up
    - An appropriate data structure (sometimes a "hash table") can give (almost) the best of both worlds

Or, in the case of non-detereministic machines, ∅

Infinite-State Systems

- If we consider an infinite set of possible inputs (all possible strings), many systems are best modeled as infinite-state systems
  - e.g., a counter that keeps track of the number of inputs so far
  - In practice, your machine has only a finite memory, but it is not very useful to model it as a finite-state machine if the number of states is huge
    - e.g., if a program stores 100 bits of input in memory, already the number of possible states it can have is more than the age of the universe in pico seconds
    - In general infeasible to explicitly describe the state diagram
- An infinite-state system can still be a "finite-control" system
  - i.e., system's behaviour defined by a fixed "program"
  - This is what we consider computation

Infinite-State Systems

- Even a few simple rules can lead to complex behavioural patterns (or rather, "non-patterns")
  - Popular examples
    - Game of Life
    - Cellular automata
    - Aperiodic tilings/Quasicrystals
  - A simple model for computation
    - Turing Machines
    - Later...