Automata As Input

- Our goal is to make TMs that can simulate other automata, given as input
- TMs can only take strings as input, so we need a way to encode automata as strings
- We'll start with the simplest: DFAs…

DFAs Encoded Using {0,1}

- The DFA's alphabet and strings:
  - Number Σ arbitrarily as $\Sigma = \{\sigma_1, \sigma_2, ...\}$
  - Use the string 1^i to represent symbol σ_i
  - Use 0 as a separator for strings
  - For example, if $\Sigma = \{a, b\}$, let a = σ_1 and b = σ_2; then *abba* is represented by 101101101
- The DFA's states:
  - Number $Q = \{q_1, q_2, ...\}$, making q_1 the start state and numbering the others arbitrarily
  - Use the string 1^i to represent symbol q_i

DFA Encoding, Continued

- The DFA's transition function:
  - Encode each transition $\delta(q_i,\sigma_j) = q_k$ as a string $1^i01^j01^k$
  - Encode the entire transition function as a list of such transitions, in any order, using 0 as a separator
  - For example,

  - Numbering *a* as σ_1 and *b* as σ_2, δ is
    $\delta(q_1,\sigma_1) = q_2$ $\quad$ $\delta(q_1,\sigma_2) = q_1$
    $\delta(q_2,\sigma_1) = q_1$ $\quad$ $\delta(q_2,\sigma_2) = q_2$
  - That is encoded as:
    101011 0 101101 0 110101 0 11011011

DFA Encoding, Continued

- The DFA's set of accepting states:
  - We already encode each state q_i as 1^i
  - Use a list of state codes, separated by 0s
- Finally, the complete DFA:
  - Transition-function string, 00, accepting-state string:
    101011 0 101101 0 110101 0 11011011 00 11

Simulating a DFA

- We have a way to represent a DFA as a string over {0,1}
- Now, we'll show how to construct a TM that simulates any given DFA
  - Given the encoded DFA as input, along with an encoded input string for it
  - Decide whether the given DFA accepts the given string
- We'll use a 3-tape TM…

3-Tape DFA Simulator

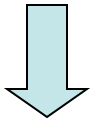
- First tape holds the DFA being simulated
- Second tape holds the DFA's input string
- Third tape hold the DFA's current state q_i, encoded as 1^i as usual

Example:

- Initial configuration, in the start state, on input *abab*:

△

- Each simulated move performs one state transition and erases one encoded input symbol…



First move on *abab*: read *a*, go to state q_2

Strategy

- Step 1: handle termination:
  - If the second tape is not empty, go to step 2
  - If it is empty, the DFA is done; search the list of accepting states (tape 1) for a match with the final state (tape 3)
  - If found, halt and accept; if not, halt and reject
- Step 2: look up move:
  - Search tape 1 for the move $1^i01^j01^k$ that applies now, where 1^i matches the current state (tape 3) and 1^j matches the current input symbol (tape 2)
- Step 3: execute move:
  - Replace the 1^i on the tape 3 with 1^k
  - Write **B** over the 1^j (and any subsequent 0) on tape 2
  - Go to step 1

An Easy Simulation

- That was no challenge for our 3-tape TM
- Used only a fixed, finite portion of each tape
- There is (by Theorem 16.8) a 1-tape TM with the same behavior
- One detail we're skipping: what should the TM do with ill-formed inputs?
  - If we specified behavior for ill-formed inputs, there would have to be an extra initial pass to verify the proper encoding of a DFA and its input

Outline

- 16.1 Turing Machine Basics
- 16.2 Simple TMs
- 16.3 A TM for $\{a^n b^n c^n\}$
- 16.4 The 7-Tuple
- 16.5 The Languages Defined By A TM
- 16.6 To Halt Or Not To Halt
- 16.7 A TM for $\{xcx \mid x \in \{a,b\}^*\}$
- 16.8 Three Tapes
- 16.9 Simulating DFAs
- 16.10 Simulating Other Automata

Simulating Other Automata

- We can use the same 3-tape technique to simulate all our other automata
- Trickier for nondeterministic models (NFAs and stack machines): our deterministic TM must search all sequences of moves
- Relatively straightforward for deterministic automata