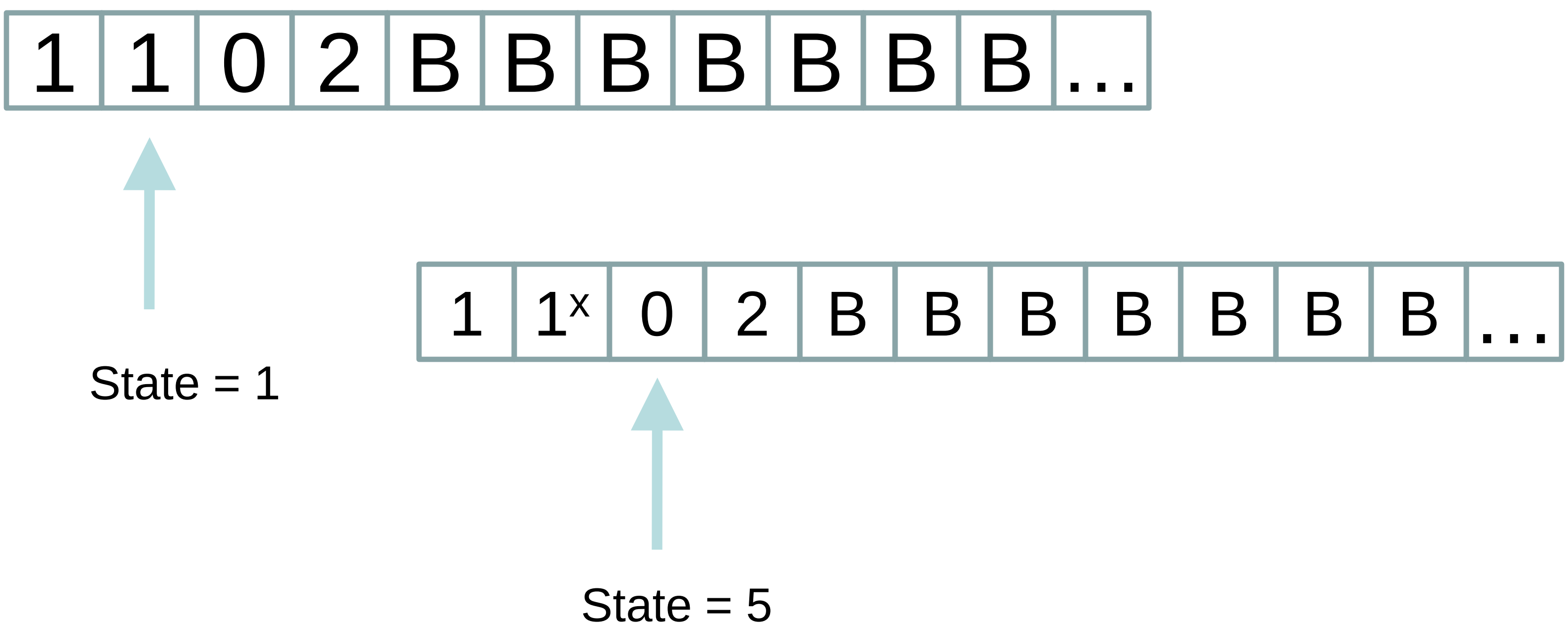
Turing Machine Variants

Sipser pages 148-154

Marking symbols

- It is often convenient to have the Turing machine leave a symbol on the tape but to mark it in some way



An expanded alphabet

- Marking is achieved by expanding the tape alphabet.
- Add a new symbol with a mark for every old symbol in the tape alphabet
- Marking x "expands to two moves"
- $\delta(q,a) \rightarrow (a^x,r,L)$
- $\delta(r,a^x) \rightarrow (a^x,q,R)$

Strategy

- Most Turing machine variants are introduced by showing how a regular Turing machine can simulate the variant.
- Simulation often uses one or more of the following tricks
  - Adding new symbols to the tape alphabet
  - Adding new states to the set of states
  - Adjusting the transition function
  - Placing marks between symbols on the tape

Multiple Tracks

- If you'd like the tape cells to contain not one, but three symbols (perhaps from different alphabets $\Gamma_1, \Gamma_2, \Gamma_3$), then you just use the tape alphabet $\Gamma = \Gamma_1 \times \Gamma_2 \times \Gamma_3$.
- Effectively, the tape now has 3 “tracks”, which we can manipulate independently.
- Note that the blank symbol of Γ is (B_1,B_2,B_3), where B_i is the blank of Γ_i.
- A common application of this idea is to use one track for “real” data, and the second track for one or more “markers” that conveniently mark some positions in the strings.

Example

- Suppose we want a TM for the language of palindromes over {0,1} that contain more 0's than 1's.

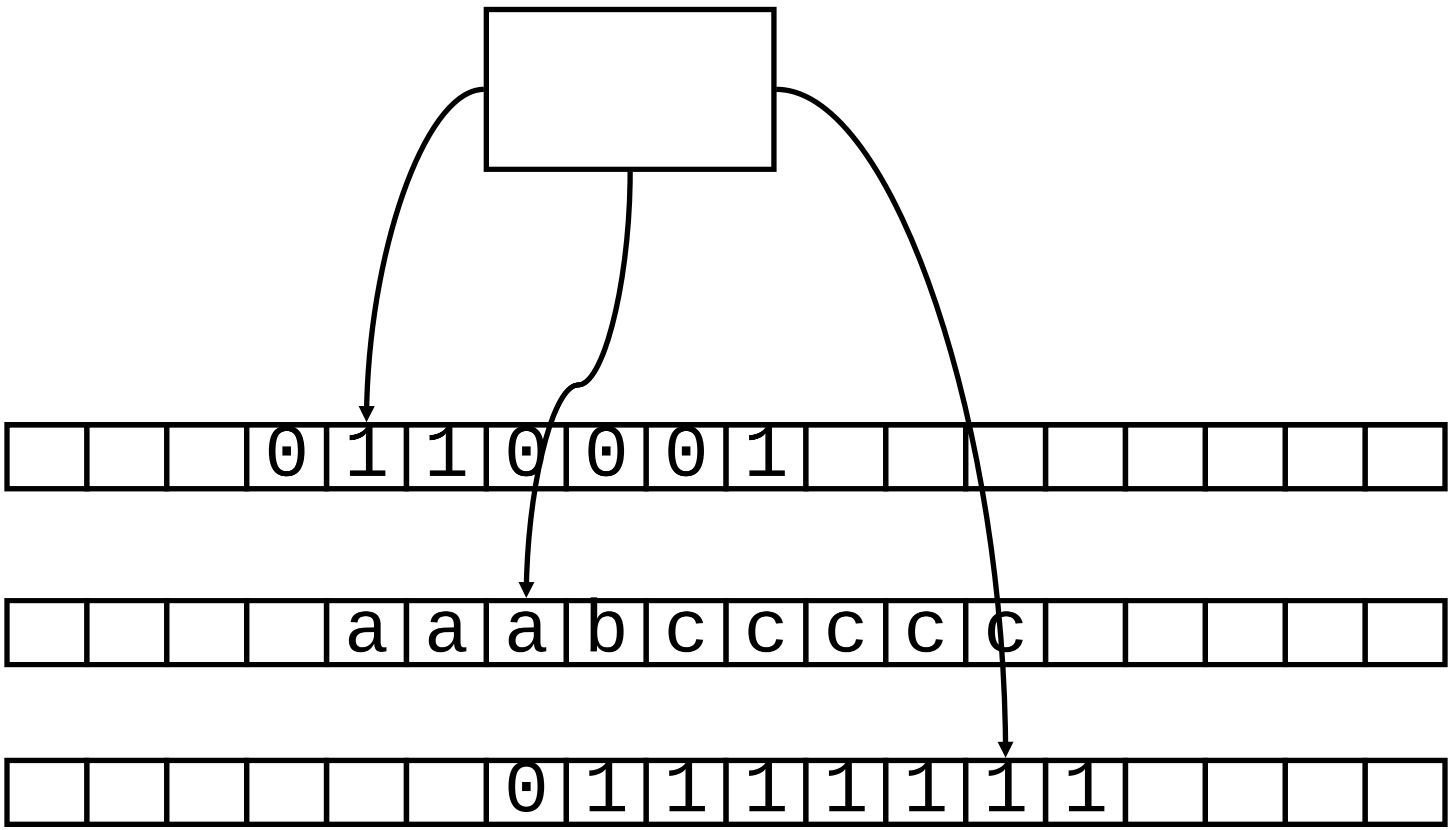
- The natural idea is to first check if the input is a palindrome, then count the 0's and 1's.

- The palindrome TM of the previous example cannot be used because it progressively deletes the input.

- But we can modify it by using the new tape alphabet Γ '= $\Gamma \times$ {*,B}. At the beginning, we put the mark * on the first and the last symbol of the input, then move these two marks one cell closer, as we check that the ``real'' contents of the two cells are equal.

Multi-Tape Turing Machines

- These generalized TM's can use a finite number of independent tapes.

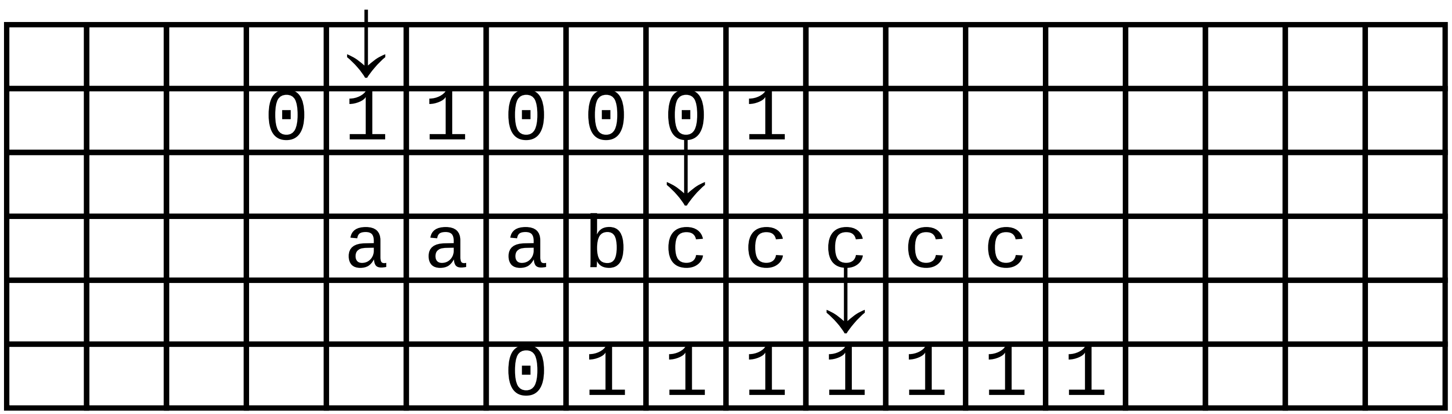


•

- Transitions are determined by the current state and the contents of all scanned cells (one on each tape).
- On a transition, the TM moves to the next state, scanned symbols get overwritten, and each head gets a direction to move (L, R, or S (stationary)).
- Initially, the first tape holds the input. The other tapes are blank.

Simulating Multitape TM's

- To simulate k tapes, use one tape with 2k tracks. One track holds the contents of each tape, another marks the position of the corresponding head.

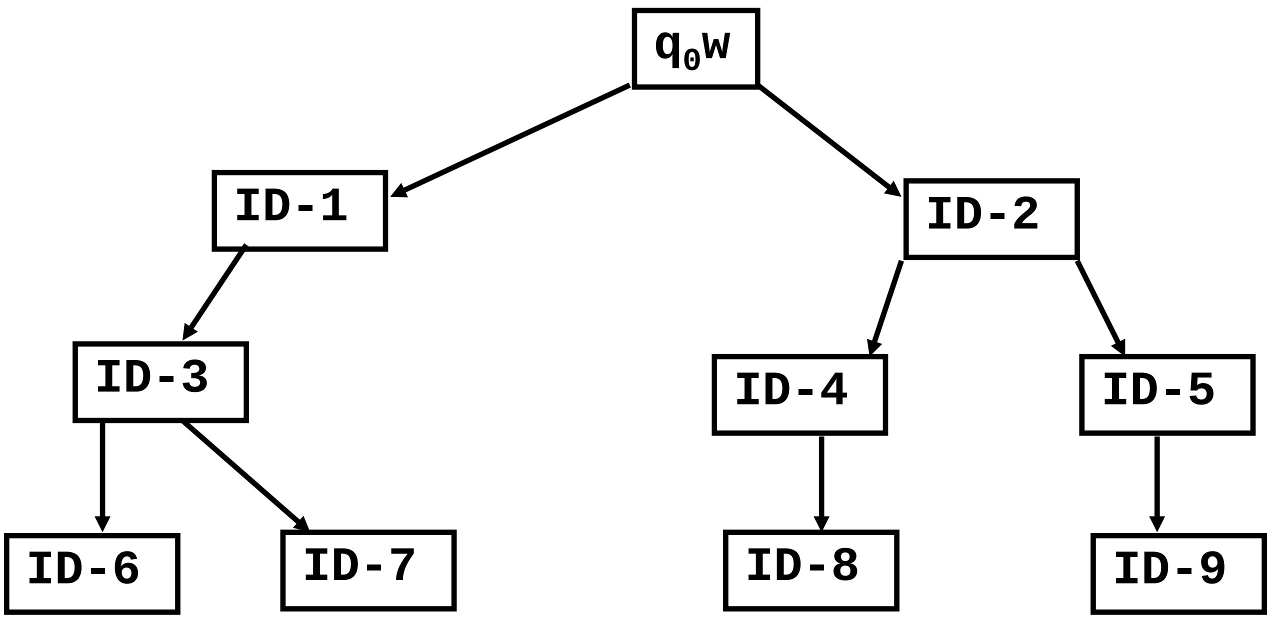


- One move of the multitape TM M is simulated by a sequence of moves of the one tape TM M_1:

1. M_1 moves left, then right, visiting all the $\downarrow$'s to see what each tape head of M is scanning.
2. Based on the scanned symbols of M and the current state of M (that M_1 keeps remembering), M_1 knows the next move of M.
3. With the information about the next move of M available, M_1 visits each $\downarrow$ again, changing the corresponding symbol on one of the tracks, and moving that $\downarrow$ appropriately.

Nondeterministic Turing Machines (NTM)

- The definition of a NTM is the same as the definition of a TM, except that the transition function has the type $\delta : Q \times \Gamma \to P(Q \times \Gamma \times \{L,R\})$
- At each move, an NTM has a finite set of choices.
- The execution of an NTM is naturally represented by a tree whose non-root nodes are all future configurations (we use ID's for instantaneous descriptions, because it is easier to write).

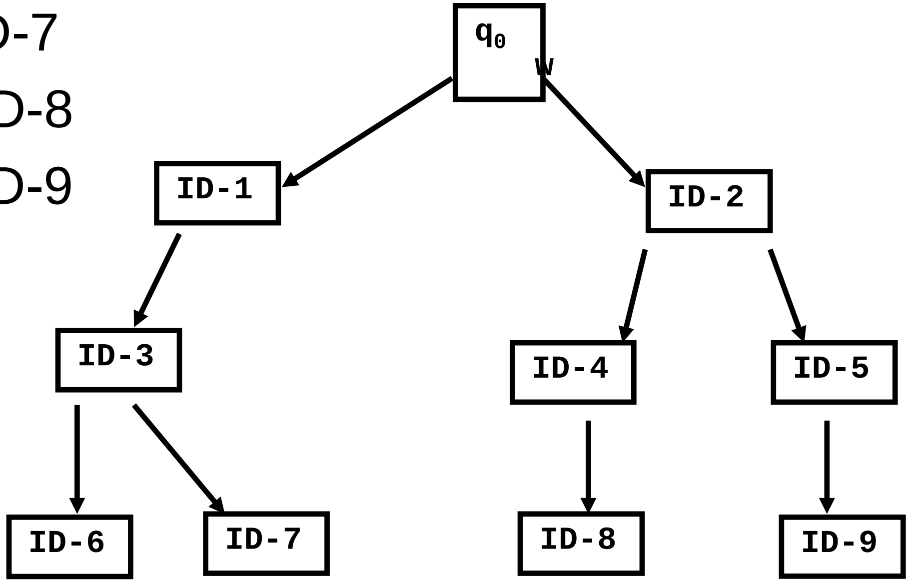


Simulating NTM's

- An NTM N is first simulated by a multitape TM M; we know that M can be then converted to a one-tape TM.
- On one of its tapes, M maintains a *queue* of ID's of N that can arise from a starting ID q_0w. These ID's are separated by a special marker $\otimes$.
- Execution of M goes in *big steps*. If ω is the ID at the front end of the queue, then M computes all possible ID's $\omega_1, \ldots, \omega_k$ that are immediate successors of ω in the execution of N.
- A big step of M consists of dequeuing ω and enqueuing $\omega_1, \ldots, \omega_k$.
- Sipser gives a different, but equivalent construction. The key is that the mechanism visits all the states in a breadth first fashion to be sure that nothing is missed.

- Here is how the queue changes in the first few big steps (|-|-) when the execution of N is as in the picture.

- q_0w |-|- ID-1 ⊗ ID-2
- |-|- ID-2 ⊗ ID-3
- |-|- ID-3 ⊗ ID-4 ⊗ ID-5
- |-|- ID-4 ⊗ ID-5 ⊗ ID-6 ⊗ ID-7
- |-|- ID-5 ⊗ ID-6 ⊗ ID-7 ⊗ ID-8
- |-|- ID-6 ⊗ ID-7 ⊗ ID-8 ⊗ ID-9
-



- Note that if the N-tree with the root q_0w contains an accepting ID ω (one in which the occurring N-state is final), then ω will eventually come to the front of the M-queue, at which point M can recognize it as N-accepting, and accept itself.

- Other tape(s) of M are used for the necessary “localized” simulations of M that each big step requires. For example, M can use a “scratch tape” to copy the first ID ω from the queue, and compute three ω's successors $\omega_1, \dots ,\omega_k$.

TM can encode stateful storage

- Some states of a TM can be structured: one component is the ``state proper'', the others hold useful data.

- **Example**. We have a TM $M=(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ and suppose we want to modify it so that, when in state r, it swaps the contents of the two immediate cells (the scanned one and the next one to the right), and then go to the state s.

Construction

- To do this, we pick two unused symbols p,q and add to Q the states [q,X] and [p,X], for each X $\in$ Γ. We also add the transitions

  δ(r,X) = ([q,X],X,R)

  δ([q,X],Y) = ([p,Y],X,L)

  δ([p,Y],X) = (s,Y,R)

- for all X,Y $\in$ Γ .

- Check that we've achieved the desired effect:
- α rXY β |- α X[q,X]Y β |- α[p,Y]XX β |- α YX β

Example

- A TM for the language of palindromes can use states of the form [q,a] (a $\in \Sigma$).
- Remembering the first symbol of the string, it deletes it (puts B in its place), then moves to the end of the input.
- Then it matches the last symbol against the stored first symbol and, if the match succeeds, it deletes the last symbol, and goes back to the first non-blank symbol, and repeats.