Turing Machines

A Turing Machine has a finite state controller and a tape that is infinite in both directions. The input is on the tape at the start of the computation; the rest of the tape is blank. The controller starts at the beginning of the input and reads one tape symbol at a time.

A move consists of the following steps, taken in order:

a) Read the current tape symbol
b) Change the state of the controller
c) Write a symbol over the one just read
d) Move left or right one symbol on the tape.

The Turing Machine accepts the input if it ever enters a final state, whether or not it has read the entire input.

Formally a Turing Machine is (Q, Σ, Γ, δ, s, B, F) where

Q is the finite set of states

Σ is the alphabet of input symbols

Γ is the tape alphabet

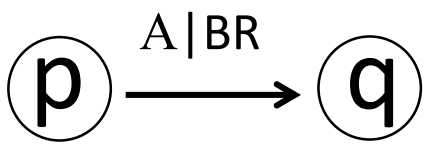
δ is the transition function (δ(q,a)=(p,b,d) where q and p are states, a and b are tape symbols, and d is a direction to move on the tape)

s is the start state

B is the blank symbol on the tape

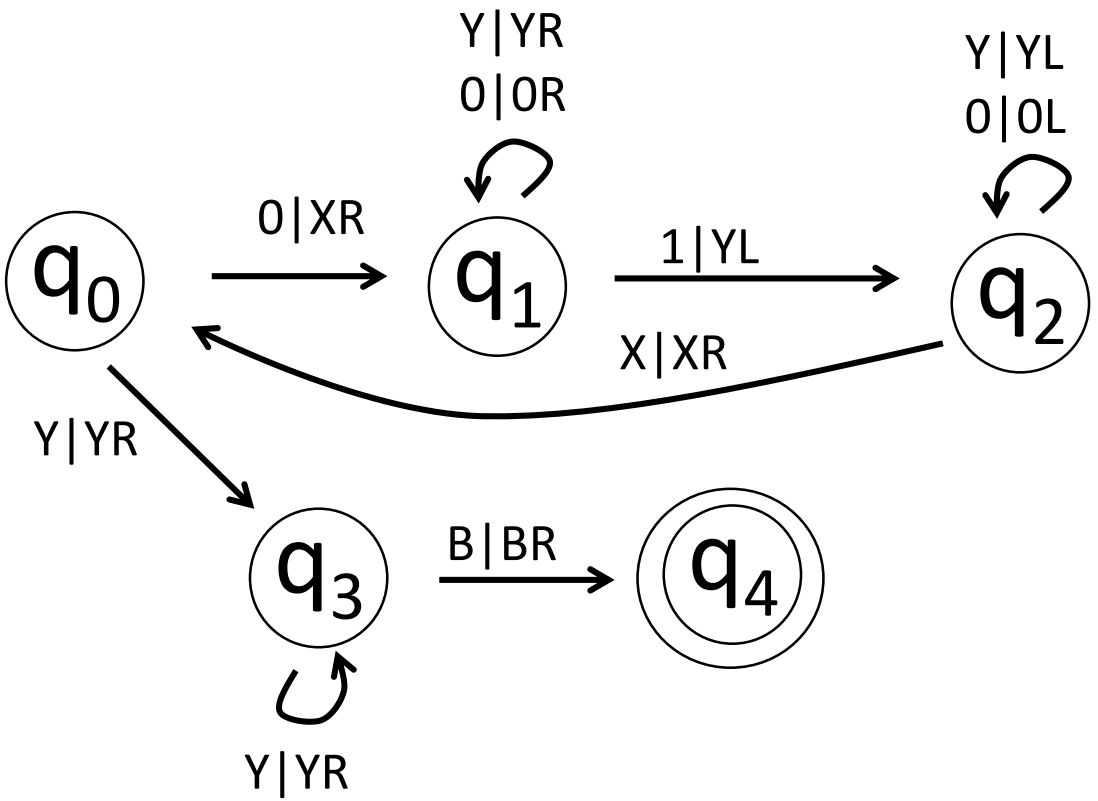
F is the set of final states

Notation for transitions:



means: "If you are in state p and the current tape symbol is A, transition to state q, overwrite the A on the tape with B and move to the right on the tape."

Example: This accepts $\{0^n1^n \mid n > 0\}$



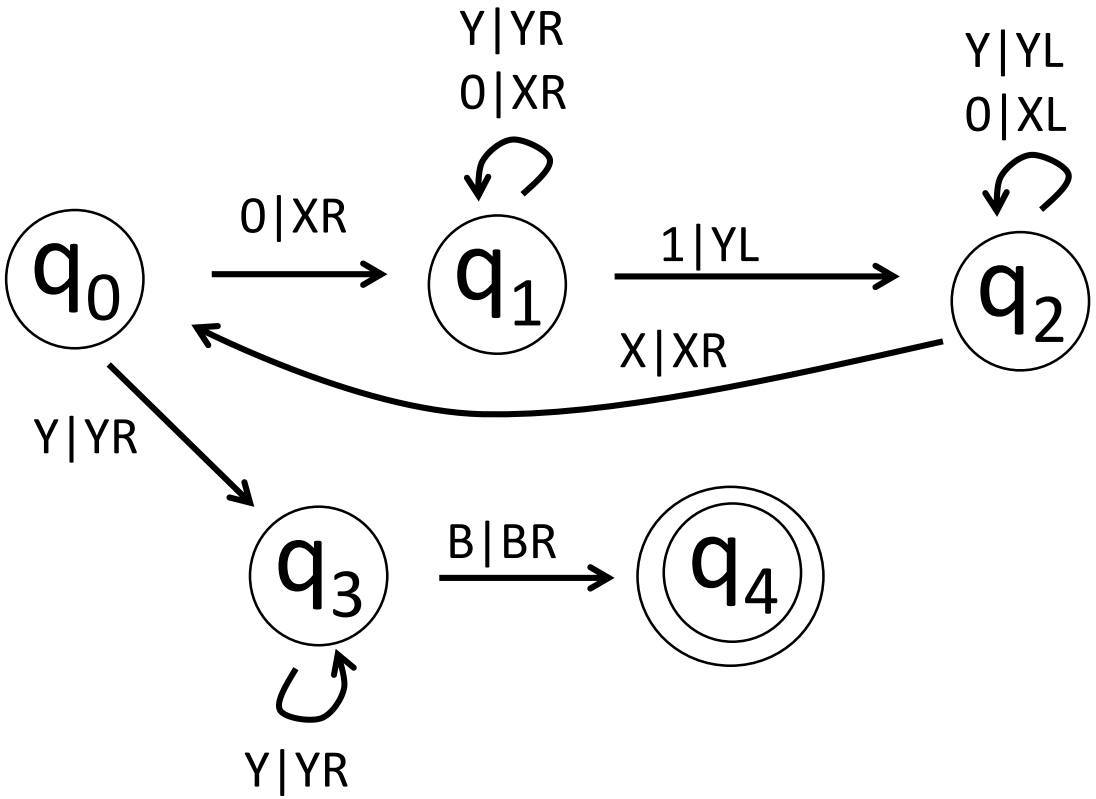
q_0: If 0 overwrite with X and go to q_1; if Y goto q_3.

q_1: Walk over 0s until you find 1, then overwrite with Y and go to q_2.

q_2: Move left over Ys and 0s looking for X; when you find X more right and go to q_0.

q_3: Move right looking for a blank; if you find it go to q_4.

q_4: Accept



XXXYYY

000111 end in q_4; accept

XXYY

00111 doesn't get to a blank from q_3.

XXXYY

00011 doesn't get to a 1 from q_1.

Configuration descriptions: let $X_1...X_{i-1}pX_i..X_n$ mean that the tape contents are $X_1..X_n$, the automaton is currently in state p, and the tape head is over X_i. We will use symbol => to indicate one step of the computation:

$q_0$0011 => X$q_1$011 => X0$q_1$11 => X$q_2$0Y1 => q_2X0Y1 => X$q_0$0Y1 =>

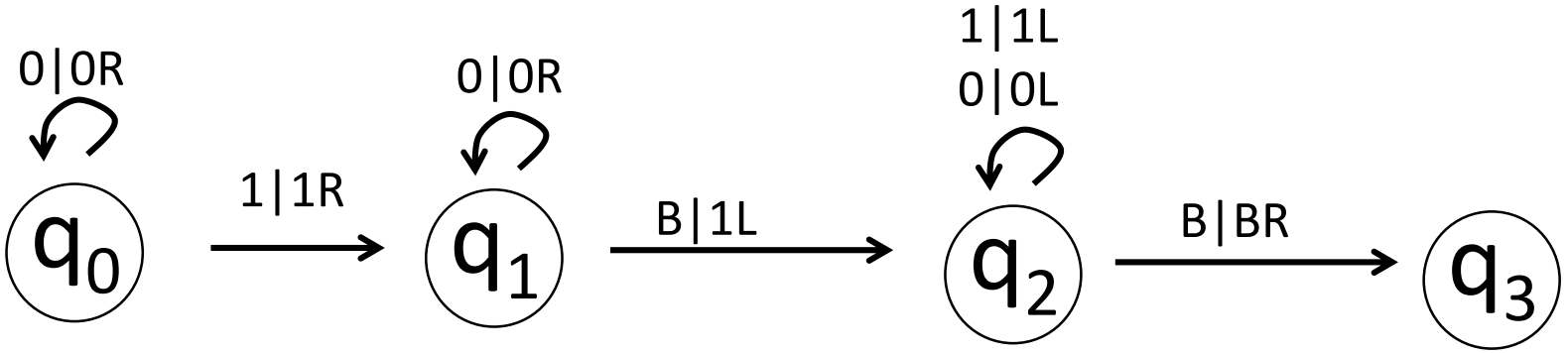
=> XXq_1Y1 => XXY$q_1$1 => XXq_2YY => Xq_2XYY => XXq_0YY => XXYq_3Y =>

=> XXYYq_3 => XXYYq_4 => ACCEPT

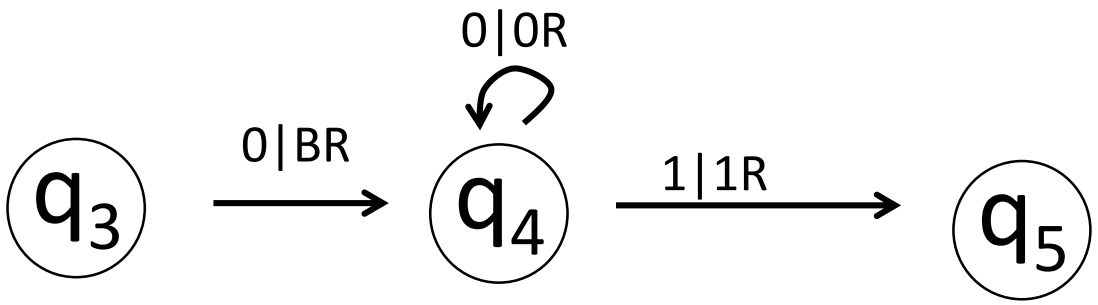
A more complex example: This starts with 0^m10^n on the tape and ends with 0^{nxm}; it performs multiplication.

Step 1: Start with 0^m10^n; write 1 at the end of the input to give 0^m10^n1.
Step 2a: If there is a 0 at the start replace it with B (blank); go to the first 1.
2b: For each 0 prior to the second 1 replace it with X and copy a 0 at the end of the tape. This step makes a copy of the second set of 0s.
2c: Replace all of the X's with 0s, then go back to the beginning of the input. Repeat Step 2.
Step 3: When there is a 1 at the beginning of the input erase both 1s and the 0s in between.
Step 4: Halt (Accept)

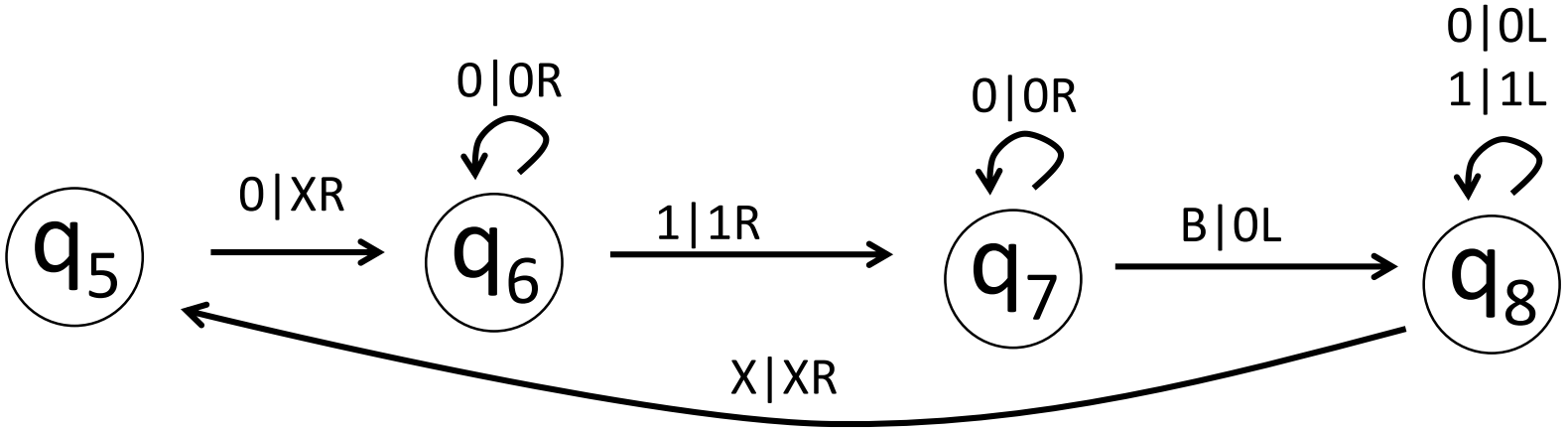
Step 1:



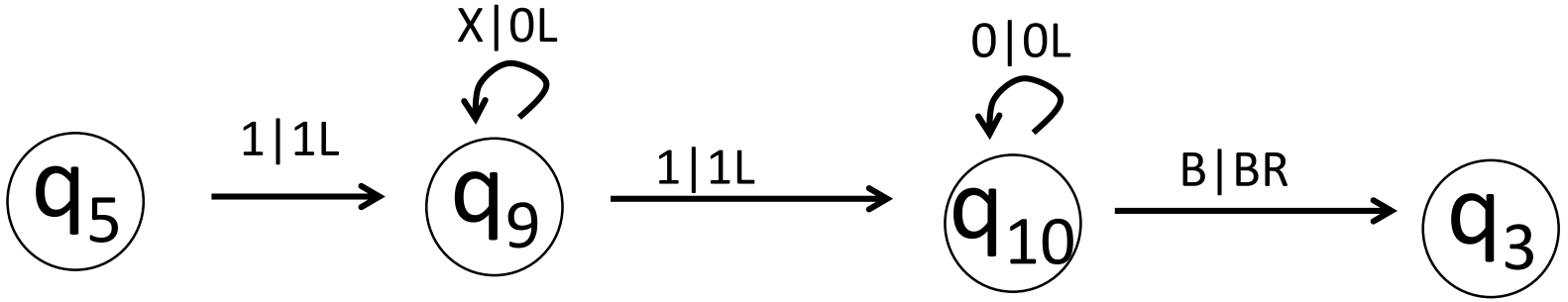
Step 2a:



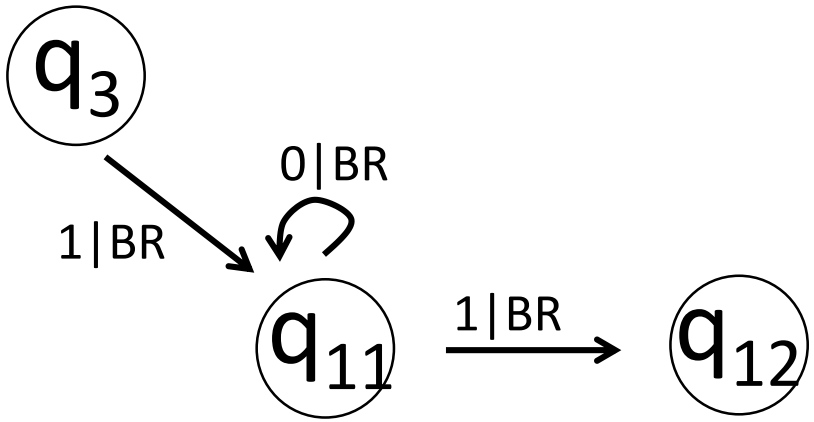
Step 2b:



Step 2c:



Step 3:



Terminology: We say that a language is *recursively enumerable* if there is a Turing Machine that accepts it. We say that a language is *recursive* if there is a Turing Machine that halts on all inputs and accepts the language.

With a recursive language we can tell if any string is in the language or not: its Turing Machine either halts in a final state or a non-final state.

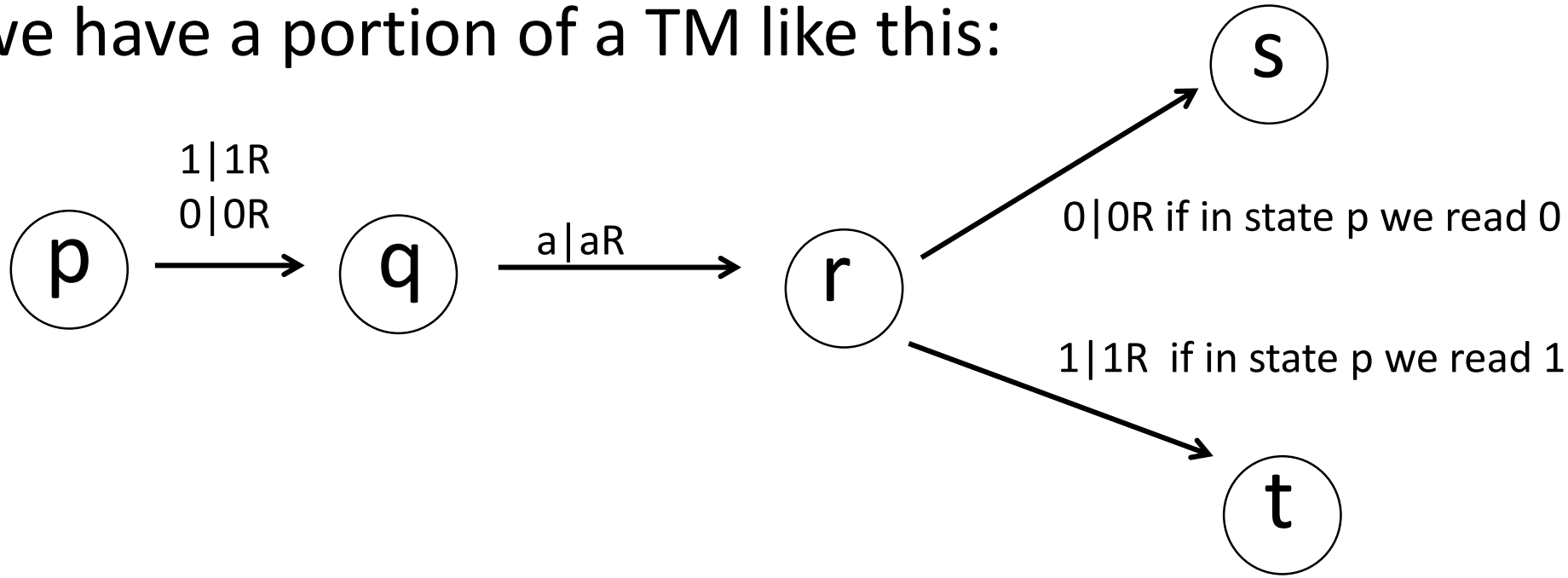
With a recursively enumerable language the TM will halt and accept on strings that are in the language, but might run forever on strings that aren't in the language.

The "recursively enumerable" terminology puzzles many students. Here is where it came from. If you have a Turing Machine T that accepts a language, we could design a super Turing Machine S that generates all possible strings, one at a time, and simulates the next few steps of T on each of the strings that has been generated so far. Each time it finds a string that would be accepted by T, it outputs that string. In this way S outputs, or *enumerates* all of the strings accepted by T. We will talk more about Turing Machine simulators a few days from now.

Programming Tricks

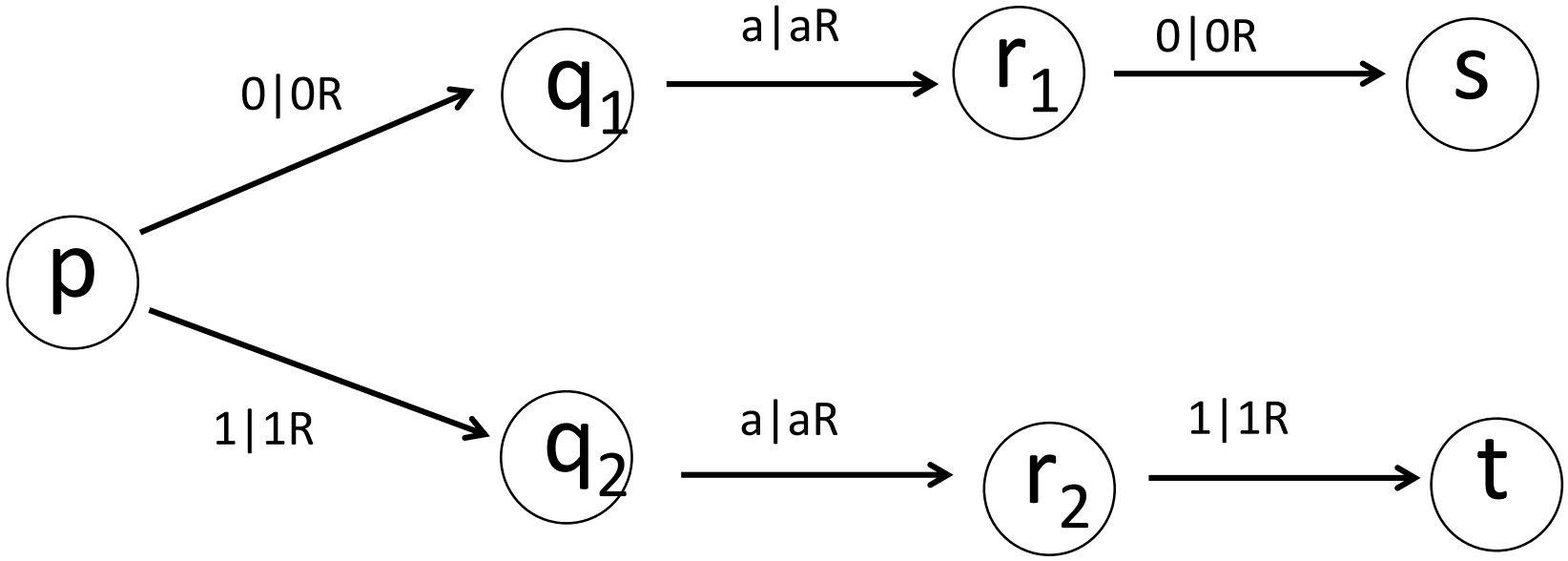
I. Remembering Data

Suppose we have a portion of a TM like this:

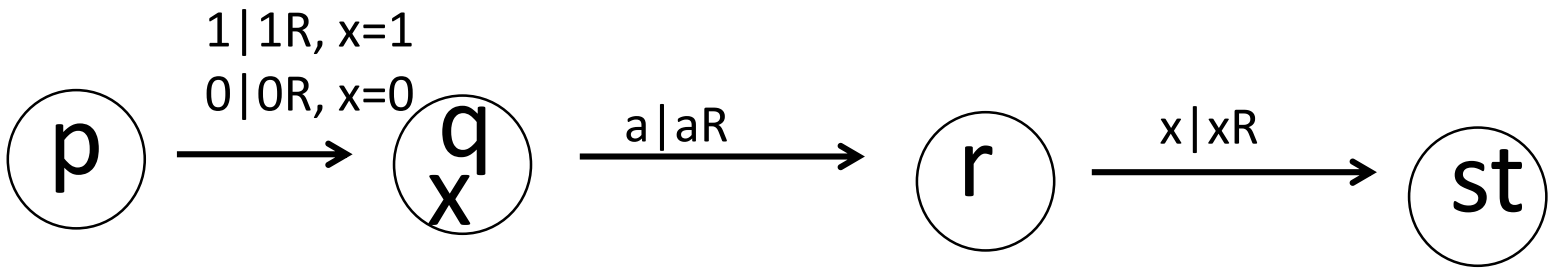


On input 0a0 we want to end up in state s; on 1a1 we want to end up in state t. In other words, we want to "remember" the value we read in state p.

To do this we make 2 copies of the p-to-r sequence:



We can think of this as storing information in the controller:



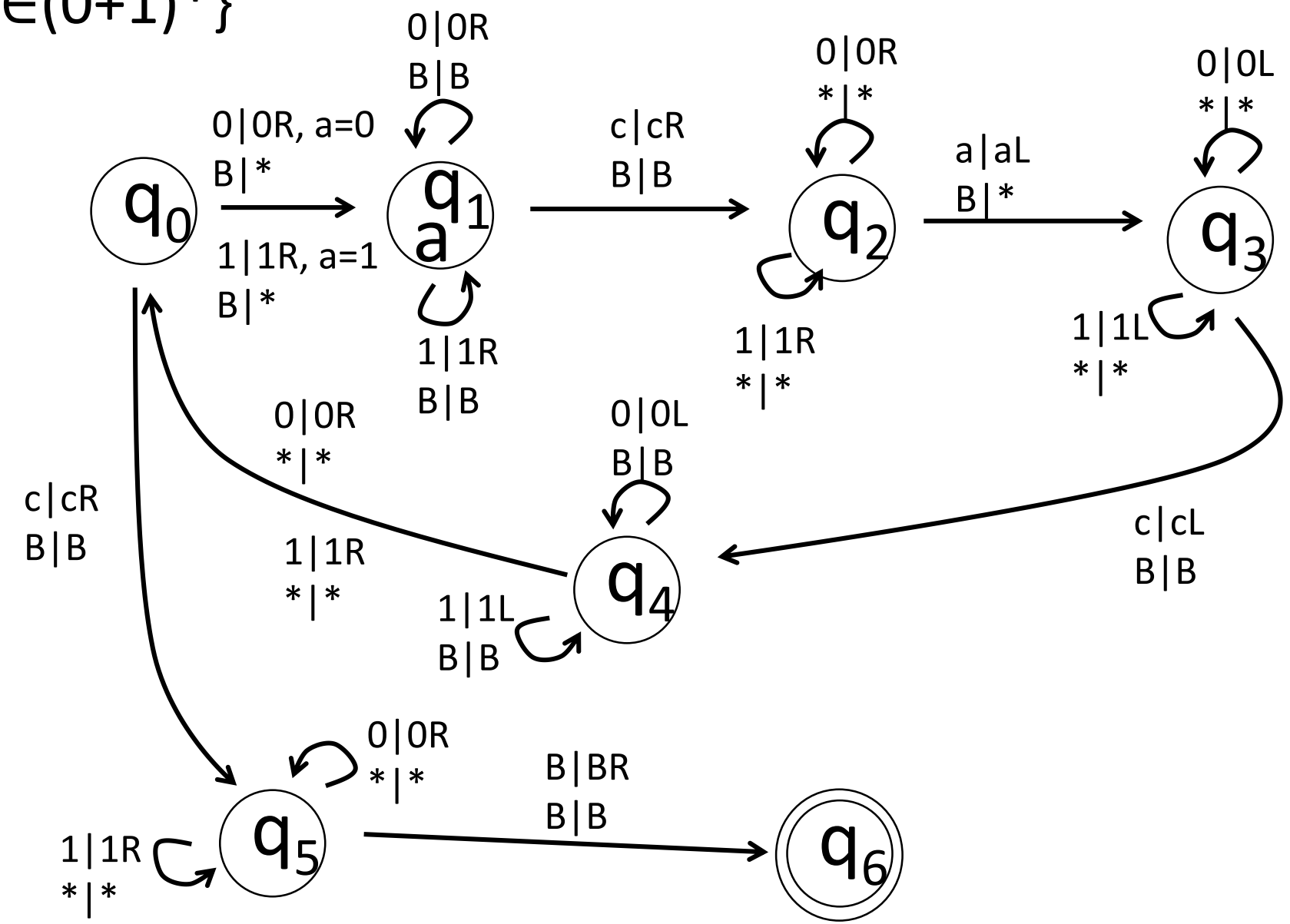
II. Multiple Tracks

We can think of a TM as having multiple tracks that we process together. The tape alphabet Γ is finite so instead of one track with Γ we might have 3 with $\Gamma x \Gamma x \Gamma$, where tape "symbol" (a,b,c) represents a on track 1, b on track 2, and c on track 3. We read all 3 tracks at each step.

We usually think of the first track as containing the original input and the other tracks as initially blank areas for scratch work developed during the execution of the TM.

Example: This 2-track TM accepts the non-context-free language

{wcw | w∈(0+1)*}



III. Multiple Tape TMs (with independent tape heads)

We can define a Turing Machine with k independent tapes. We start with the input on the first tape and the others blank. At each step we read the current symbol off each tape, use those k symbols to transition to a new state, write a new symbol on each tape and choose a direction to move on each tape.

We can simulate such a k-tape TM by one with 1 tape and 2k tracks. We will represent each tape with 2 tracks: one with the contents of the tape and one with a marker showing us the current position of the tape head.

Each node of the k-tape automaton is split into 3^k nodes in the simulator to keep track of the answers to the k questions: where is the current position of tape i relative to the current position in the simulator: is the current position of the simulator to the left (L), to the right (R) or at the current position (C) of tape i?

i.e., with a simulator for a 2-tape TM node q is split into qLL, qLC, qLR, qCL, qCC, qCR, qRL, qRC, qRR. If s is the start symbol we start the simulator in sCC.

To simulate a move of the multiple-tape TM

a) Visit the current position of each tape and gather the symbols.
b) Revisit each position, overwrite the tape symbol and update the current tape position.
c) Update the simulator's state.

Note that if we are moving left looking for the current position on tape 1 and we pass the current position of tape 2, we transition from state qRR to qRC to qRL, and keep moving left. The 3^k variants of each node really do allow us to keep track of how to find the current position of each tape.

These simulators show:

Theorem: Any language accepted by (and any computation performed on) a k-track TM is also accepted by (or performed on) a standard TM.

Theorem: Any language accepted by (and any computation performed on) a k-tape TM is also accepted by (or performed on) a standard TM.

IV. Nondeterministic TMS have multivalued transition functions:

$\delta(p,a) = \{(q_1,Y_1,D_1), (q_2,Y_2,D_2) \ldots (q_n,Y_n,D_n)\}$

We can make a 3-tape TM that computes the reachable configurations of a nondeterministic TM and halts in an accept state of the nondeterministic TM ever does.

Here are the three tapes:

Tape 1 holds the input, such as 0110

Tape 2 has 2 tracks. The first track has a list of computed configurations separated by *-symbols. The second track is blank except for a symbol that marks the start of the current configuration:

$$q_0 0110*0q_1 110*0Xq_1 10*.....$$
$$\uparrow$$

Tape 3 just has scratch work

Step 1: Copy the input to Tape 2, prefix it with q0 to make it a configuration, and mark it as the current configuration.

Step 2: Copy the current configuration to Tape 3. From the configuration you can read the current state q and current input symbol a. For each transition value in $\delta(q,a)$ copy the current configuration to the end of the configuration track on Tape 2 and update it according to the transition rules. For example, if the current configuration is $01Xq_11$ and $\delta(q_1,1)=\{(q_2,X,R),(q_3,Y,L)\}$ then we want to write onto Tape 2 $*01XXq_2*01q_3XY$

Step 3 Update the current configuration marker to the next configuration and redo Step 2.

If this process ever writes a configuration onto Tape 2 containing a final state, the simulator halts and accepts.

This gives us:

Theorem: If language $\mathcal{L}$ is accepted by a nondeterministic TM then $\mathcal{L}$ is also accepted by a standard deterministic TM.