Turing Machines

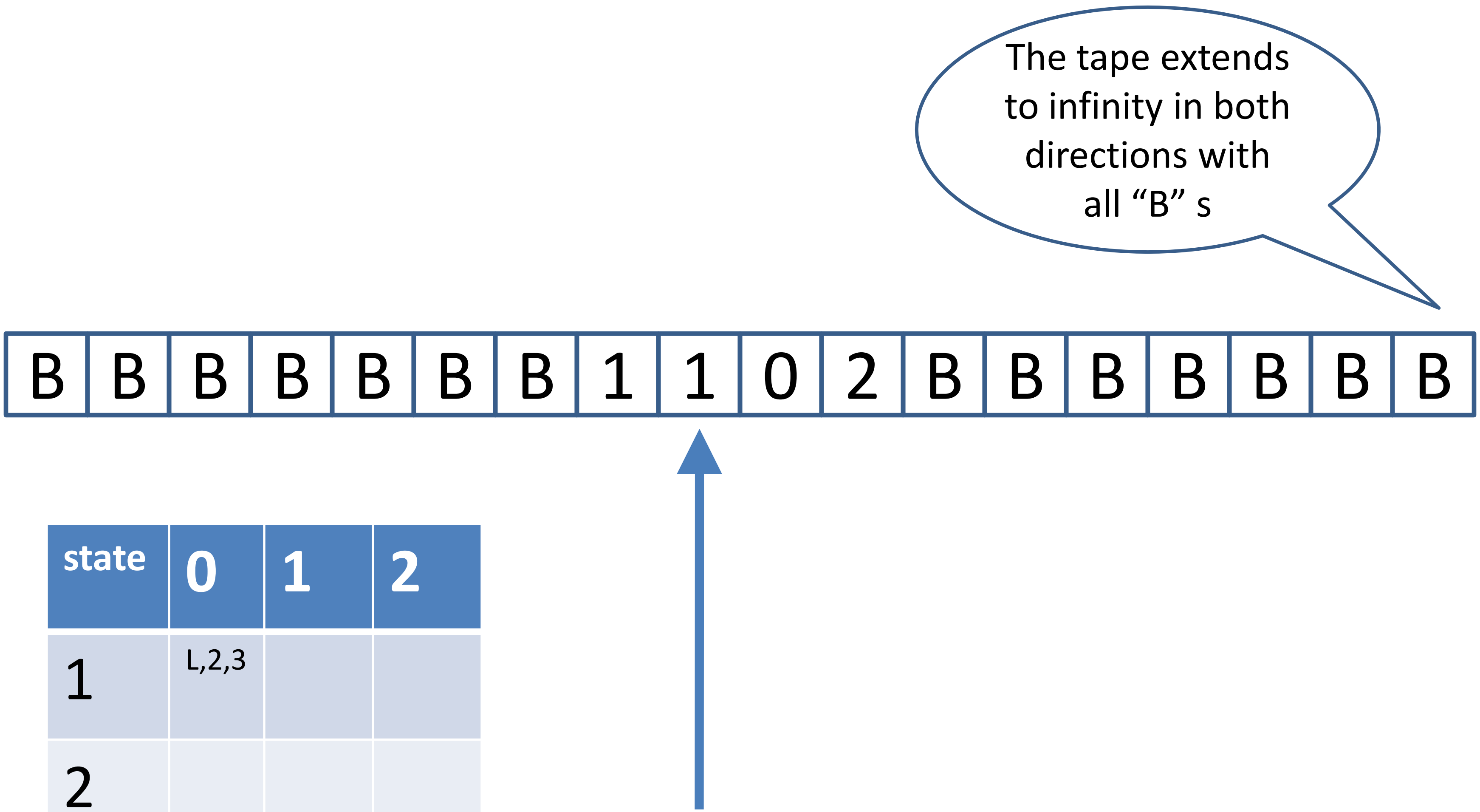
Intro to Turing Machines

- A *Turing Machine* (TM) has finite-state control (like PDA), and an infinite read-write *tape*. The tape serves as both input and unbounded storage device.

- The tape is divided into *cells*, and each cell holds one symbol from the *tape alphabet*.

- There is a special *blank* symbol B. At any instant, all but finitely many cells hold B.

- *Tape head* sees only one cell at any instant. The contents of this cell and the current state determine the next move of the TM.



state	0	1	2
1	L,2,3		
2			
3			
4			

State = 1

Moves

- A *move* consists of:
  - replacing the contents of the scanned cell
  - repositioning of the tape head to the nearest cell on the left, or on the right
  - changing the state

- The *input alphabet* is a subset of the tape alphabet. Initially, the tape holds a string of input symbols (the *input*), surrounded on both sides with in infinite sequence of blanks. The initial position of the head is at the first non-blank symbol.

Formal Definition

- A TM is a septuple $M=(Q,\Sigma,\Gamma,\delta,q_0,B,F)$, where
  - Q is a finite set of states
  - Γ is the tape alphabet, and $\Sigma \subseteq \Gamma$ is the input alphabet
  - $B \in \Gamma - \Sigma$ is the blank symbol
  - $q_0 \in Q$ is the start state, and $F \subseteq Q$ is the set of accepting states
  - $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$ is a partial function. The value of $\delta(q,X)$ is either undefined, or is a triple consisting of the new state, the replacement symbol, and direction (left/right) for the head motion.

Example

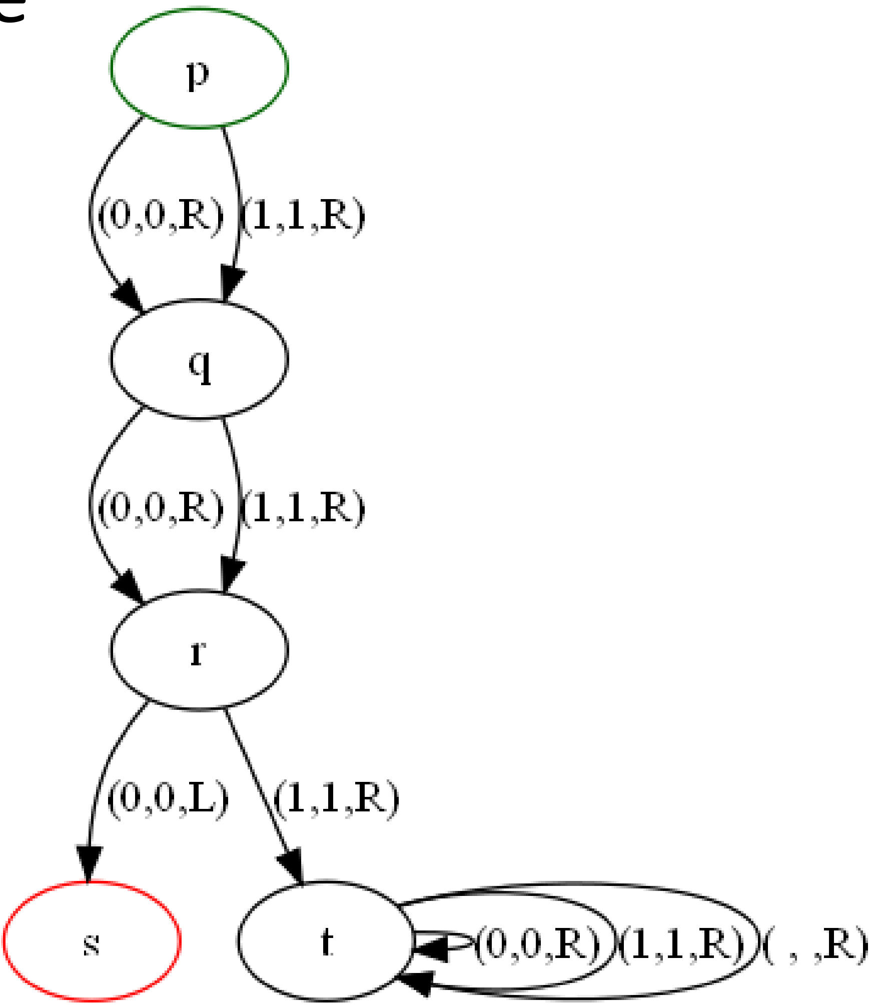
- Here is a TM that checks its third symbol is 0, accepts if so, and runs forever, if not.

- $M=(\{p,q,r,s,t\},\{0,1,\},\{0,1,B\},p,B,\{s\})$

- $\delta(p,X) = (q,X,R)$ for $X=0,1$
- $\delta(q,X) = (r,X,R)$ for $X=0,1$
- $\delta(r,0) = (s,0,L)$
- $\delta(r,1) = (t,1,R)$
- $\delta(t,X) = (t,X,R)$ for $X=0,1,B$

Transisition Diagrams for TM

- Pictures of TM can be drawn like those for PDA's. Here's the TM of the example below.



$\delta(p,X) = (q,X,R)$ for $X=0,1$
$\delta(q,X) = (r,X,R)$ for $X=0,1$
$\delta(r,0) = (s,0,L)$
$\delta(r,1) = (t,1,R)$
$\delta(t,X) = (t,X,R)$ for $X=0,1,B$

Implicit Assumptions

- Input is placed on tape in contiguous block of cells
- All other cells are blank: ‘B’
- Tape head positioned at Left of input block
- There is one start state

- The text uses a single Halt state, an alternative is to have many final states. These are equivalent, why?

Example 2: { $a^n b^m$ | n,m in Nat}

states = 0,1,H

tape alphabet = a,b,^

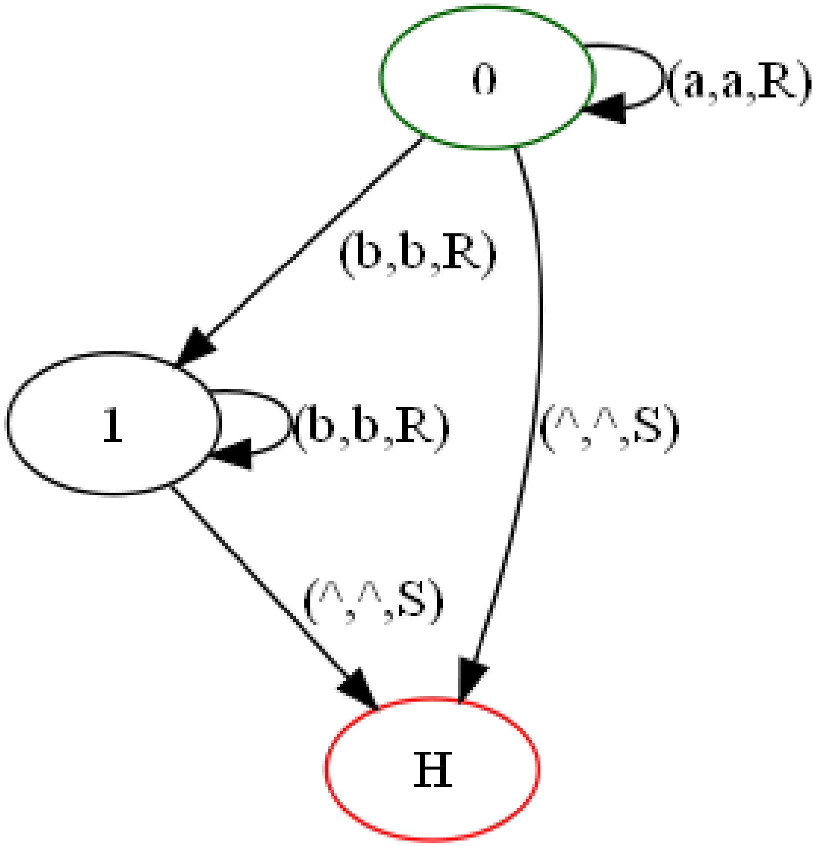
input alphabet = a,b

start = 0

blank = '^'

final = H

delta = (0,^,^,S,H)
(0,a,a,R,0)
(0,b,b,R,1)
(1,b,b,R,1)
(1,^,^,S,H)

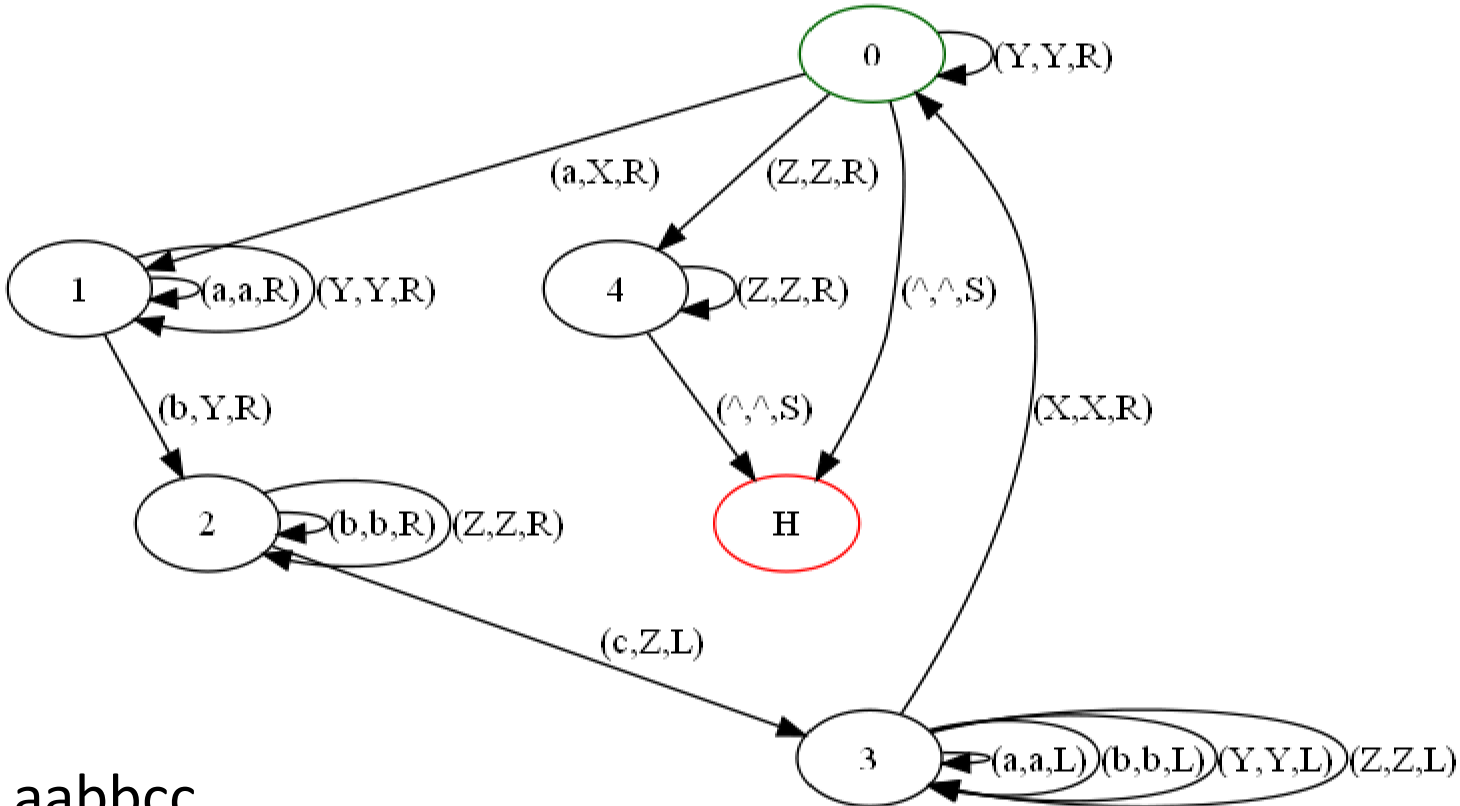


Example 3: { $a^nb^nc^n$ | n in Nat}

delta =

(0,a,X,R,1) Replace a by X and scan right
(0,Y,Y,R,0) Scan right over Y
(0,Z,Z,R,4) Scan right over Z, but make final check
(0,^,^,S,H) Nothing left, so its success
(1,a,a,R,1) Scan right looking for b, skip over a
(1,b,Y,R,2) Replace b by y, and scan right
(1,Y,Y,R,1) scan right over Y
(2,c,Z,L,3) Scan right looking for c, replacxe it by Z
(2,b,b,R,2) scan right skipping over b
(2,Z,Z,R,2) scan right skipping over Z
(3,a,a,L,3) scan left looking for X, skipping over a
(3,b,b,L,3) scan left looking for X, skipping over b
(3,X,X,R,0) Found an X, move right one cell
(3,Y,Y,L,3) scan left over Y
(3,Z,Z,L,3) scan left over Z
(4,Z,Z,R,4) Scan right looking for ^, skip over Z
(4,^,^,S,H) Found what we're looking for, success!

tape alphabet = a,b,c,^,X,Y,Z
input alphabet = a,b,c
start = 0
blank = '^ '
final = H



aabbcc

Xabbcc

XaYbcc

XaYbZc

XXYbZc

XXYYZc

XXYYZZ

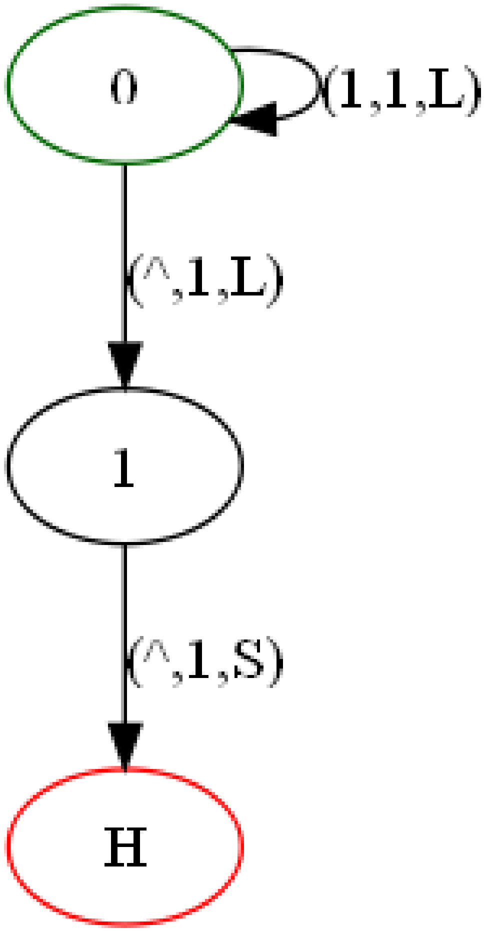
Turing machines with output

- A Turing machine can compute an output by leaving an answer on the tape when it halts.

- We must specify the form of the output when the machine halts.

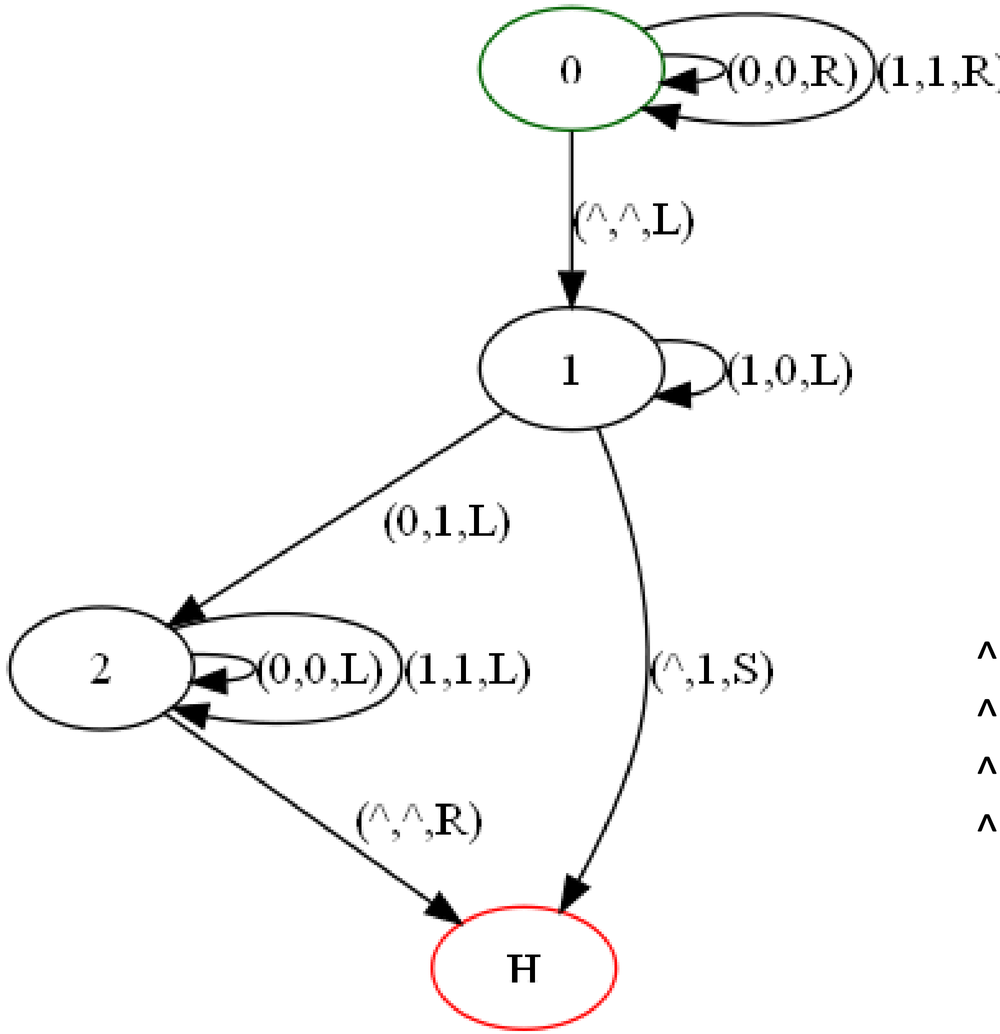
Adding two to a number in unary

states = 0,1,H
tape alphabet = 1,^
input alphabet = 1
start = 0
blank = '^'
final = H
delta =
  (0,1,1,L,0)
  (0,^,1,L,1)
  (1,^,1,S,H)



Adding 1 to a Binary Number

states = 0,1,2,H
tape alphabet = 1,0,^
input alphabet = 1,0
start = 0
blank = ‘^ '
final = H
delta =
(0,0,0,R,0)
(0,1,1,R,0)
(0,^,^,L,1)
(1,0,1,L,2)
(1,1,0,L,1)
(1,^,1,S,H)
(2,0,0,L,2)
(2,1,1,L,2)
(2,^,^,R,H)

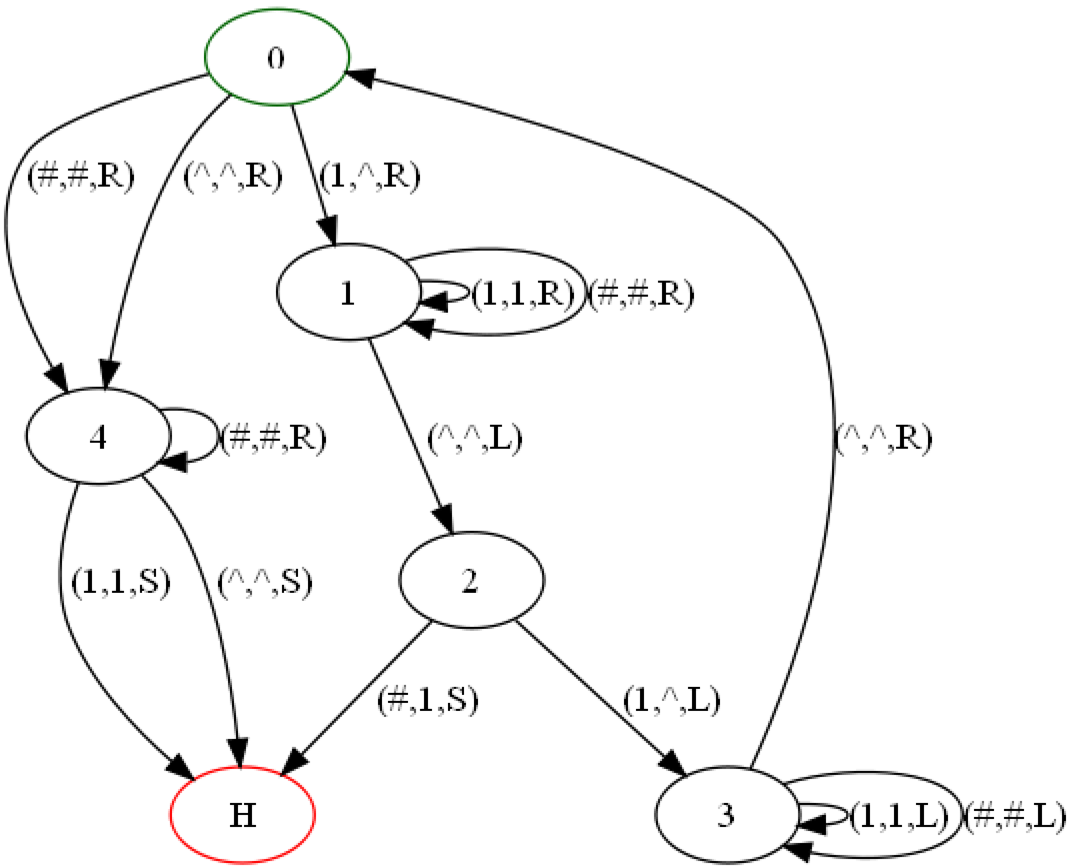


^1011^
^1010^
^1000^
^1100^

An equality Test

states = 0,1,2,3,4,H
tape alphabet = 1,0,#,^
input alphabet = 1,0,#
start = 0
blank = '^'
final = H

delta =
(0,1,^,R,1)
(0,^,^,R,4)
(0,#,#,R,4)
(1,1,1,R,1)
(1,^,^,L,2)
(1,#,#,R,1)
(2,1,^,L,3)
(2,#,1,S,H)
(3,1,1,L,3)
(3,^,^,R,0)
(3,#,#,L,3)
(4,1,1,S,H)
(4,^,^,S,H)
(4,#,#,R,4)



Instantaneous Descriptions

- ID's for TM's are strings of the form α q β , where $\alpha, \beta \in \Gamma^*$ and q $\in$ Q. (Assume that Q and Γ^* are disjoint sets, guaranteeing unique parsing of ID's.)
- The string α represents the non-blank tape contents to the left of the head.
- The string β represents the non-blank tape contents to the right of the head, including the currently scanned cell.

- Adding or deleting a few blank symbols at the beginning of an ID results in an equivalent ID. Both represent the same instant in the execution of a TM.

Sipser terminology

- Sipser calls instantaneous descriptions configurations

- Starting Configuration
- Accepting Configuration
- Rejecting Configuration

Both of these are halting configurations

- TM's transitions induce the relation |- between ID's.
- Let $\omega = X_1 \ldots X_{i-1}\ q\ X_i \ldots X_k$ be an ID.

- If $\delta(q,X_i)$ is undefined, then there are no ID's ω ' such that ω |- ω '.

- If $\delta(q,X_i)=(p,Y,R)$ then
  ω |- ω ' holds for $\omega' = X_1 \ldots X_{i-1}\ Y\ p\ X_{i+1} \ldots X_k$

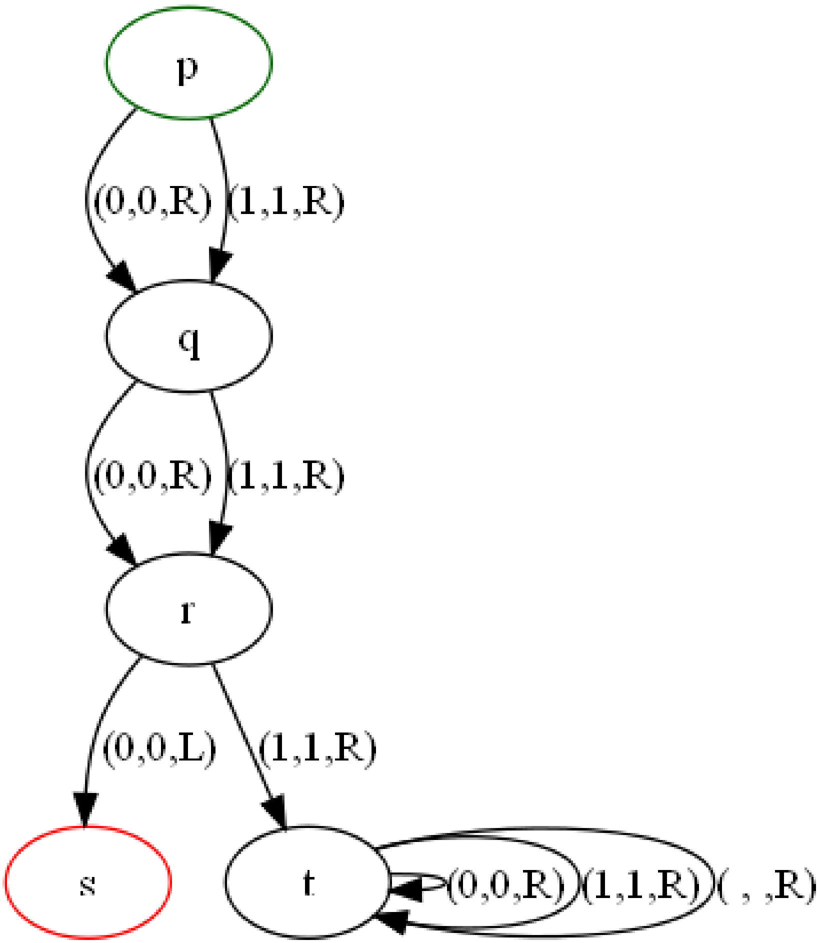
- Similarly, if δ(q,X_i)=(p,Y,L)
  then ω |- ω’ holds for $\omega' = X_1 \ldots pX_{i-1}YX_{i+1} \ldots X_k$

- When ω |- ω’ Sipser says: “ ω yields ω’ ”

Note

- If, in the first case, we have i=k, (that is we are at the end of the non-blank portion of the tape to the right) then we need to use the equivalent representation
- $\omega = X_1 \ldots X_{k-1}\, q\, X_k\, B$
- for our formula to make sense. Similarly, we add a B to the beginning of ω whenever necessary.

Example



- Here is the sequence of ID's of our example machine,showing its execution with the given input 0101:
- p0101 |- 0q101 |-01r01 |- 0s101
- The machine halts, since there are no moves from the state s. When the input is 0111, the machine goes forever, as follows:
- p0111 |- 0q111 |- 01r11 |- 011t1 |- 0111t |- 0111Bt |- 0111BBt |- …

The Language of a TM

- We define the language of the TM M to be the set L(M) of all strings $w \in \Sigma^*$

- such that: $Q_0 w \mid\!-^* \alpha\, p\, \beta$
- for some $p \in F$ and any α, β

- Languages accepted by TM's are call *recursively enumerable* (RE). Sipser calls this Turing-recognizable

- **Example**. For our example machine, we have L(M)= $(0+1)(0+1)0(0+1)^*$

- If the machine recognizes some language, and also halts for all inputs. We say the language is Turing-decidable.

Acceptance by Halting

- Some text books define an alternative way of defining a language associated with a TM M. (But not Sipser, though the idea is still interesting).
- We denote it H(M), and it consists of strings that cause the TM to halt. Precisely, a string $w \in \Sigma^*$ belongs to H(M)
- iff $q_0\, w \mid\!-^* \alpha\, p\, X\, \beta$
- where $\delta(p,X)$ is undefined.

- **Example**. For our example machine, we have
- $H(M) = \varepsilon + 0 + 1 + (0+1)(0+1) + (0+1)(0+1)0(0+1)^*$

Equivalence of
Acceptance by Final State and Halting

- How would we prove such an equivalence?

- 1. Construct a TM that accepts by Halting from an ordinary one.
- 2. Construct an ordinary TM from one that accepts by halting.

Computable Functions

- Importance of having precise definitions of *effectively* computable functions, or algorithms, was understood in the 1930's. There were several attempts to formalize the basic notions of computability:
  - Turing Machines (1936)
  - Post Systems (1936)
  - Recursive Functions (Kleene, 1936)
  - Markov Algorithms (1947)
  - λ-calculus (Church 1936)

- On the surface, these approaches look quite different. It turned out, however, that they are all equivalent! All these, and all later formalizations (combinatory logic, *while* programs, C programs, etc.) give essentially the same meaning to the word *algorithm* .

Church’s Thesis

- The statement that these formalizations correspond to the intuitive concept of computability is known as *Church's Thesis.*

- Church's Thesis is a belief, not a theorem.

- (though we often act as if we believe it is true, even though we don’t know its is true)

Power of Turing Machines (1)

- Recall the Church Thesis: *Every problem that has an algorithmic solution can be solved by a Turing Machine* !
- How do we become convinced that it is reasonable to believe this thesis?

- **First**, we can develop some programming techniques for TM's, allowing us to write machines for more and more complicated problems. Structuring states and tape symbols is particularly useful. Then, there is a possibility to use one TM as a subroutine for another. After having written enough TM's, we may get a feeling that everything that we can program in a convenient programming language could be done with TM.

Power of Turing Machines (2)

- **Second**, we can consider some generalizations of the concept of TM (multitape TM's, non-deterministic TM's, ...) and prove that they are essentially just as powerful as the plain TM's.

- **Finally**, we can prove that all proposed formalizations of the concept of *computable*, of which TM's is only one, are equivalent. In later lectures we will look at both Kleene and Church’s systems.