Instantaneous Descriptions

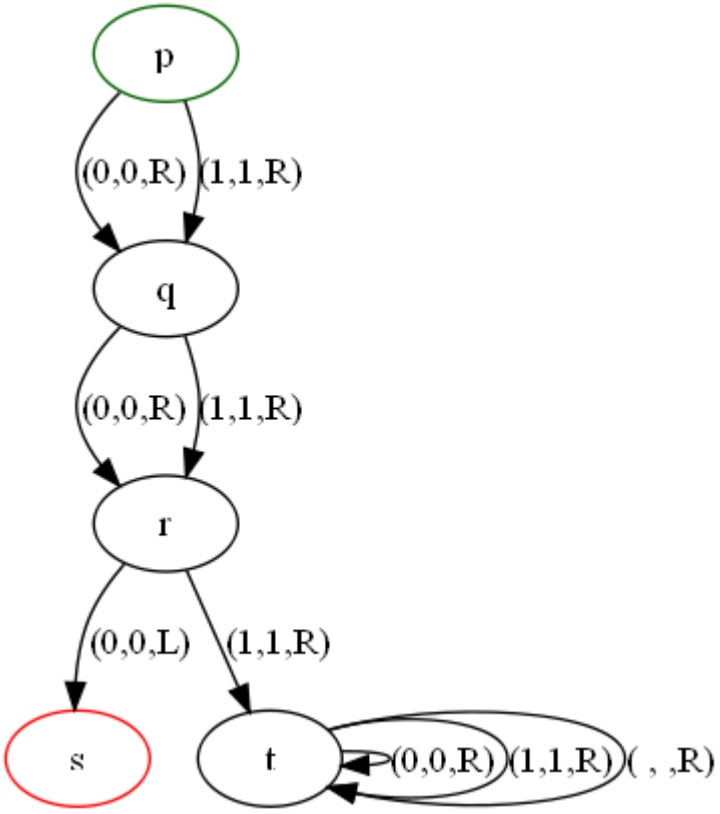
ID's for TM's are strings of the form α –q– β , where $\alpha, \beta \in \Gamma^*$ and q $\in$ Q.

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.

Example Instaneous Description



-p- 1 1 0 3

1 -q- 1 0 3

1 1 -r- 0 3

1 -s- 1 0 3

TM's transitions induce the relation |- between ID's.

Let $\omega = X_1 \ldots X_{i-1} \text{ –q– } X_i \ldots X_k$ be an ID.

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

If $\delta(q,X_i)=(p,Y,R)$ then

$\omega \vdash \omega'$ holds for $\omega' = X_1 \ldots X_{i-1}\, Y \text{ –p– } X_{i+1} \ldots X_k$

Similarly, if $\delta(q,X\_i)=(p,Y,L)$

then $\omega \vdash \omega'$ holds for $\omega' = X_1 \ldots \text{ –p– } X_{i-1}YX_{i+1} \ldots X_k$

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

$$\omega = X_1 \ldots X_{k-1} \text{ –q– } X_k B$$

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:

–p– 0101 |- 0 –q– 101 |-01 –r– 01 |- 0 –s– 101

The machine halts, since there are no moves from the state s. When the input is 0111, the machine goes forever, as follows:

-P- 0111 |- 0 –q- 111 |- 01 –r– 11 |- 011 –t– 1 |- 0111 –t– |- 0111B –t– |- 0111BB –t– |- ...

The Language of a TM

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

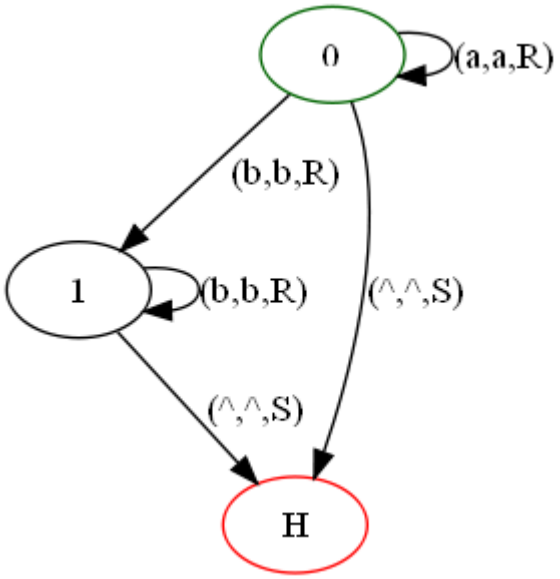
such that: $Q_0\, w \mid\!-^* \alpha \text{ –p– } \beta$

for some $p \in F$ and any α, β

Languages accepted by TM's are call *recursively enumerable* (RE).

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

Example Instantaneous Description 2



Recognize $a^n b^m$

"aabbb"

-0- a a b b b

^ a -0- a b b b

^ a a -0- b b b

^ a a b -1- b b

^ a a b b -1- b

^ a a b b b -1-

^ a a b b b -H- ^

"aabc"

-0- a a b c

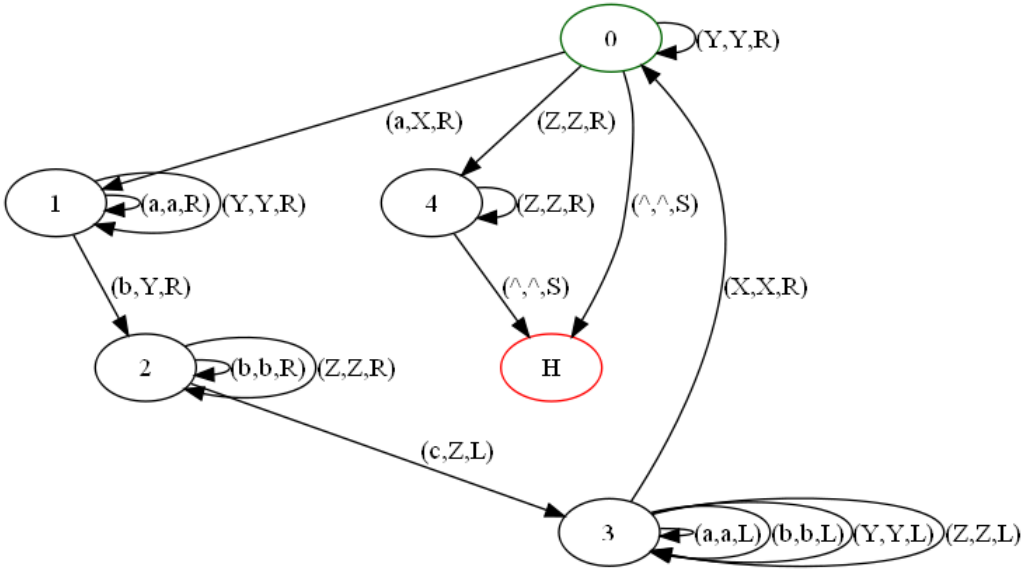
^ a -0- a b c

^ a a -0- b c

Stuck at ^ a a b -1- c

Example Instantaneous Description 3

Recognize $a^n b^n c^n$



"aabbc"

-0- a a b b c c

^ X -1- a b b c c

^ X a -1- b b c c

^ X a Y -2- b c c

^ X a Y b -2- c c

^ X a Y -3- b Z c

^ X a -3- Y b Z c

^ X -3- a Y b Z c

^ -3- X a Y b Z c

^ X -0- a Y b Z c

^ X X -1- Y b Z c

^ X X Y -1- b Z c

^ X X Y Y -2- Z c

^ X X Y Y Z -2- c

^ X X Y Y -3- Z Z

^ X X Y -3- Y Z Z

^ X X -3- Y Y Z Z

^ X -3- X Y Y Z Z

^ X X -0- Y Y Z Z

^ X X Y -0- Y Z Z

^ X X Y Y -0- Z Z

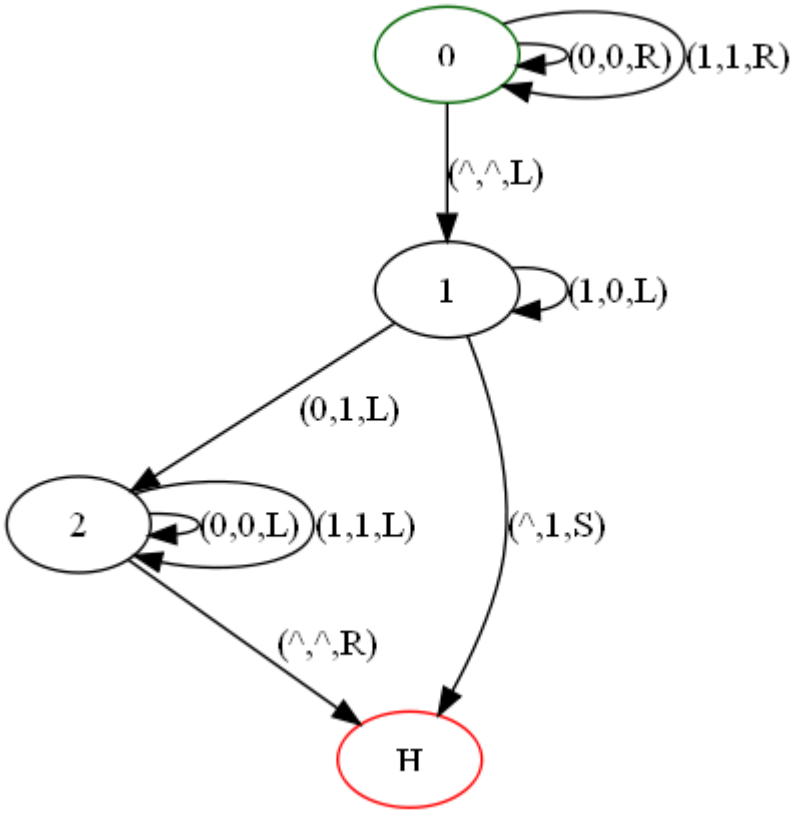
^ X X Y Y Z -4- Z

^ X X Y Y Z Z -4-

^ X X Y Y Z Z -H- ^

Example Instantaneous Description 3

Add 1 to a binary number



"101011"

-0- 1 0 1 0 1 1

^ 1 -0- 0 1 0 1 1

^ 1 0 -0- 1 0 1 1

^ 1 0 1 -0- 0 1 1

^ 1 0 1 0 -0- 1 1

^ 1 0 1 0 1 -0- 1

^ 1 0 1 0 1 1 -0-

^ 1 0 1 0 1 -1- 1 ^

^ 1 0 1 0 -1- 1 0 ^

^ 1 0 1 -1- 0 0 0 ^

^ 1 0 -2- 1 1 0 0 ^

^ 1 -2- 0 1 1 0 0 ^

^ -2- 1 0 1 1 0 0 ^

-2- ^ 1 0 1 1 0 0 ^

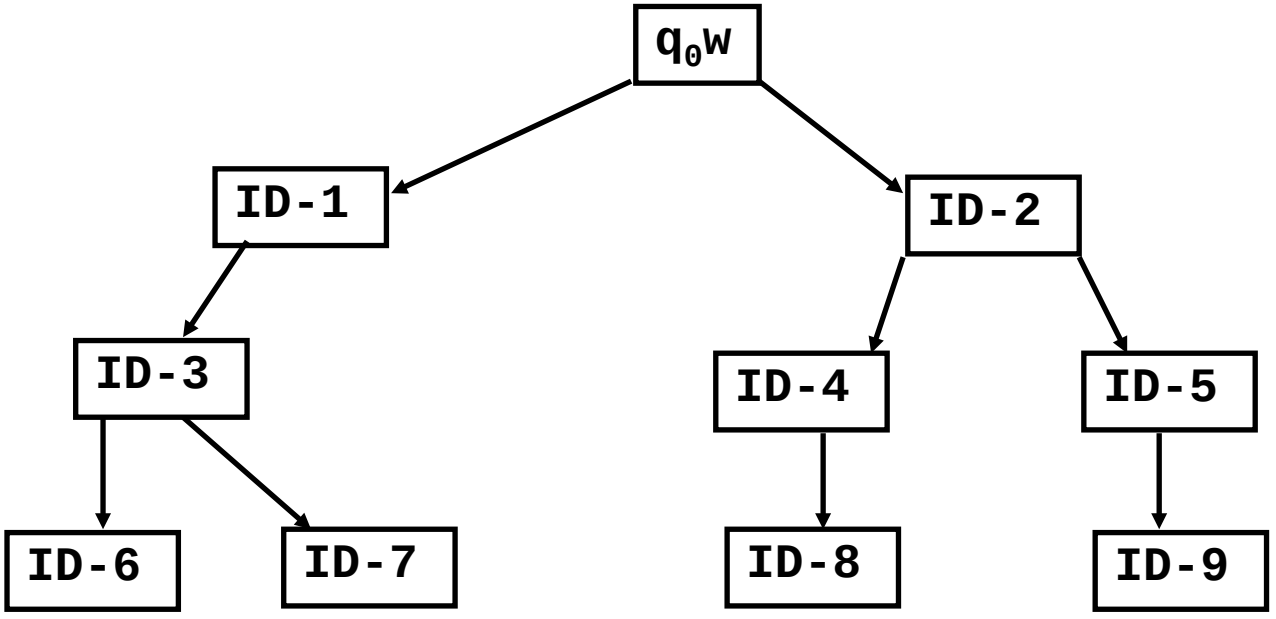
^ ^ -H- 1 0 1 1 0 0 ^

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 $\delta : Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L,R\})$

At each move, an NTM has a finite set of choices.

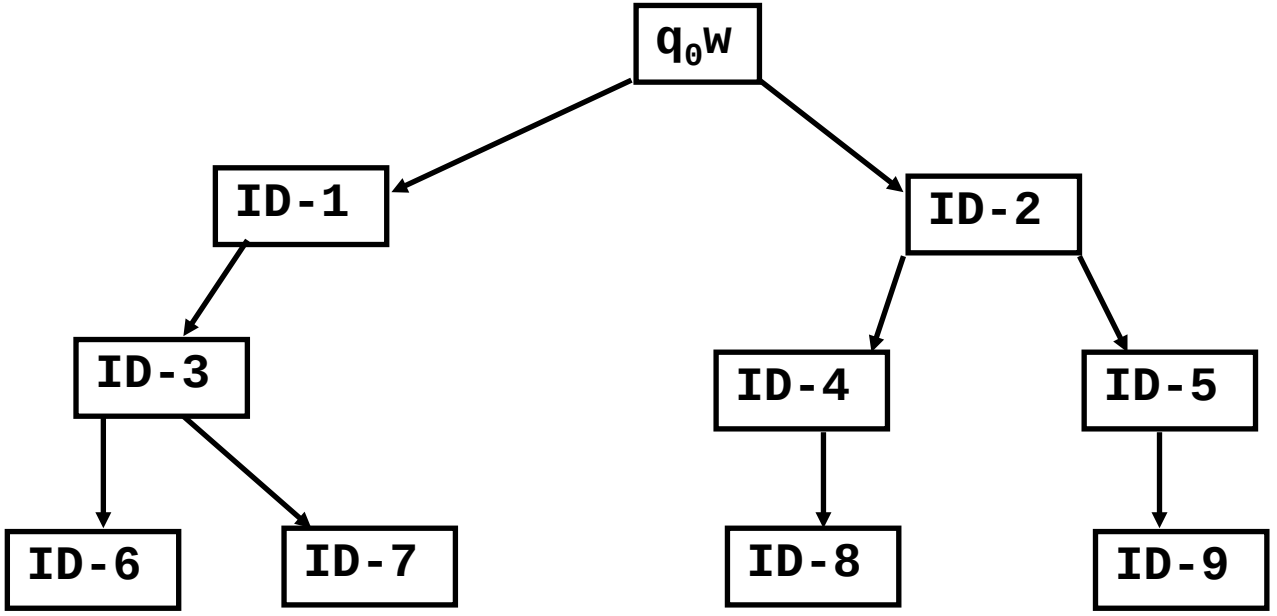
The execution of an NTM is naturally represented by a tree whose non-root nodes are all future ID's (instantaneous descriptions).



IDs and ND-Turing machines

In a nondeterministic Turing machine an instantaneous description can lead to a set of successor Ids

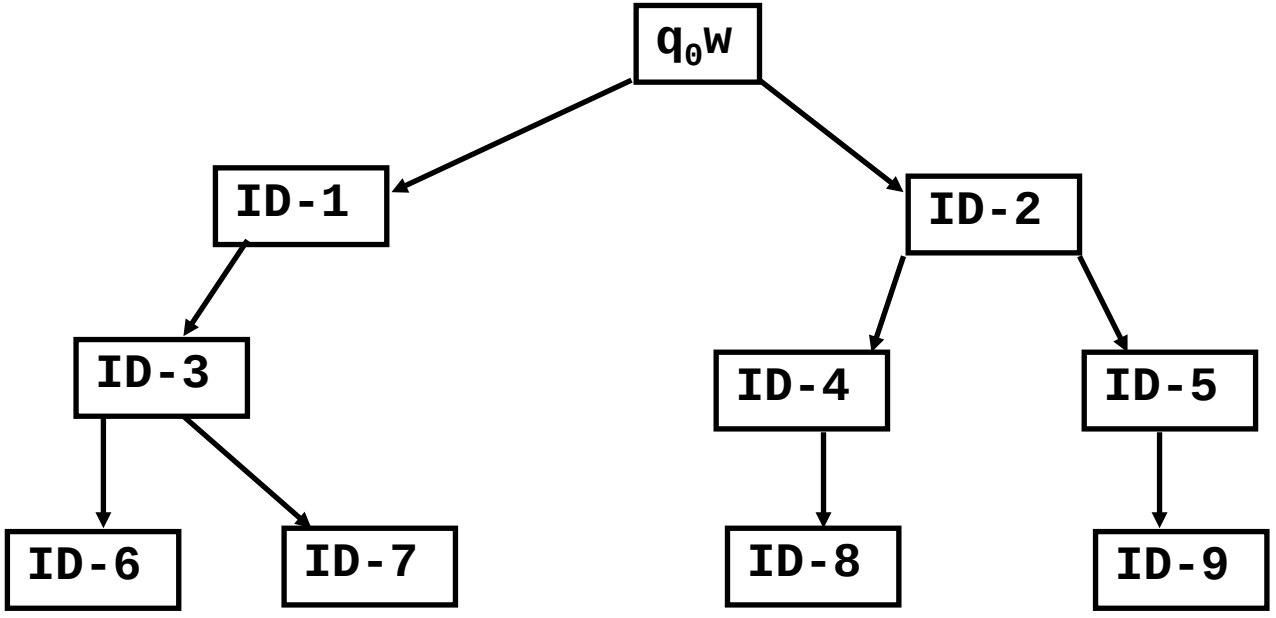
id |- { id1, id2 }



ND acceptance

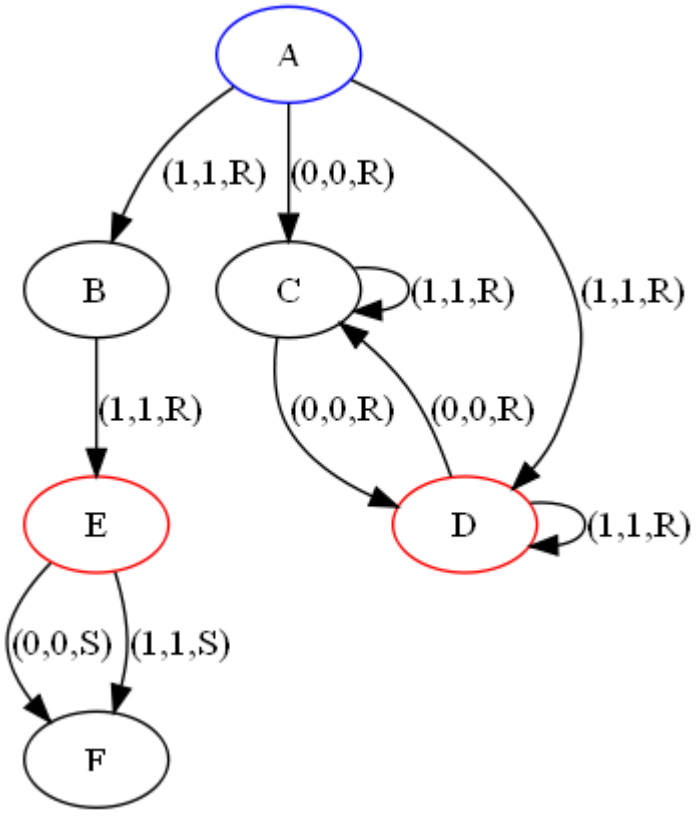
In a ND-Turing machine, if an ID has an empty set of successor IDs, and is in final state, the ND-Turing machine accepts.

To run a ND-Turing machine we must visit all the possible paths ina “fair” manner



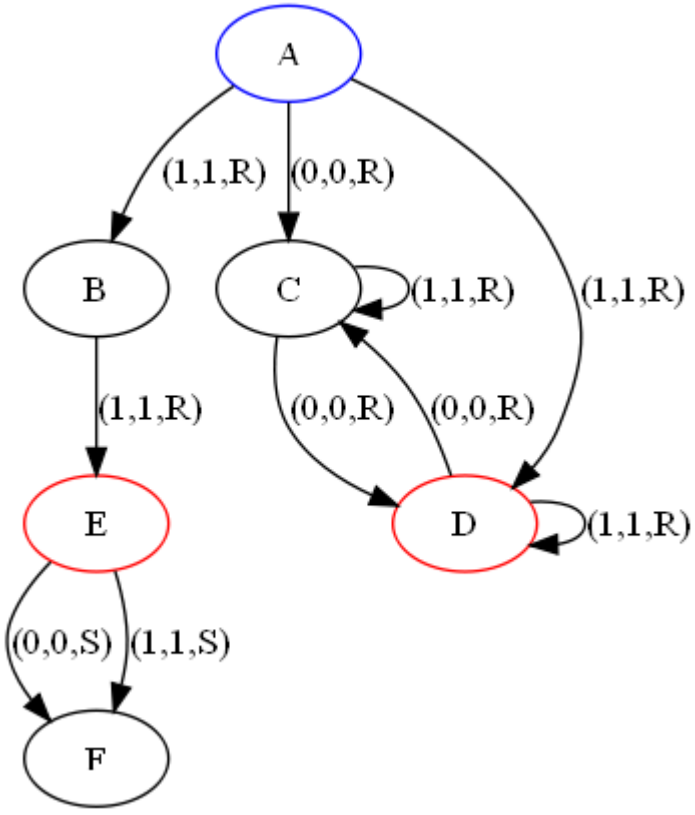
Example

Recall {w | w contains even number of 0s, or is a string of length 2 with exactly two 1's}




```
-A- 1 1
^ 1 -D- 1  |  ^ 1 -B- 1
^ 1 1 -E-  |  ^ 1 -D- 1
^ 1 1 -D-  |  ^ 1 1 -E-
^ 1 1 -E- ^|  ^ 1 1 -D-
^ 1 1 -D- ^|  ^ 1 1 -E- ^
Accepted
^ 1 1 -E- ^
```

Negative example




```
-A- 1 1 0 1
^ 1 -D- 1 0 1 |  ^ 1 -B- 1 0 1
^ 1 1 -E- 0 1 |  ^ 1 -D- 1 0 1
^ 1 1 -D- 0 1 |  ^ 1 1 -E- 0 1
^ 1 1 -F- 0 1 |  ^ 1 1 -D- 0 1
^ 1 1 0 -C- 1 |  ^ 1 1 -F- 0 1
^ 1 1 0 -C- 1
^ 1 1 0 1 -C-
Rejected, no more possibilities
```

Acceptance by Halting

Here is another way of defining a language associated with a TM M.

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 $q_0 w \vdash^* \alpha\ p\ X\ \beta$

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 ordianry TM from one that accepts by halting.

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. (We won't be doing this, of course.)

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, Σ, Γ, δ,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 $X \in \Gamma$. We also add the transitions

$\delta(r,X) = ([q,X],X,R)$

$\delta([q,X],Y) = ([p,Y],X,L)$

$\delta([p,Y],X) = (s,Y,R)$

for all $X,Y \in \Gamma$.

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.

Multiple Tracks

If you'd like the tape cells to contain not one, but three symbols (perhaps from different alphabets Γ_1, Γ_2, Γ_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.

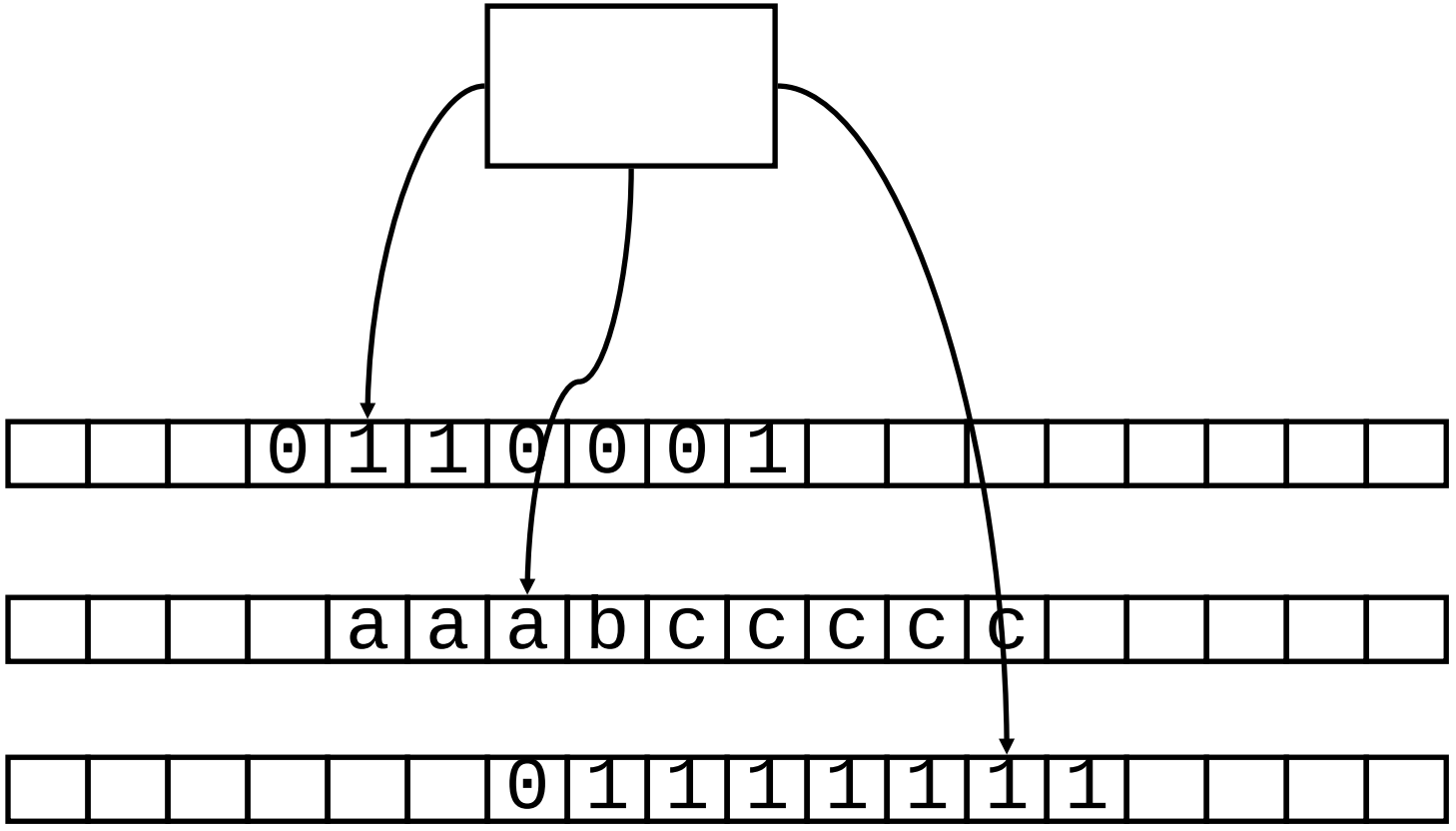
The natural idea is to first check if the input is a palindrome, then count the 0's and 1's.

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 $\Gamma' = \Gamma \times \{*,B\}$. 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.



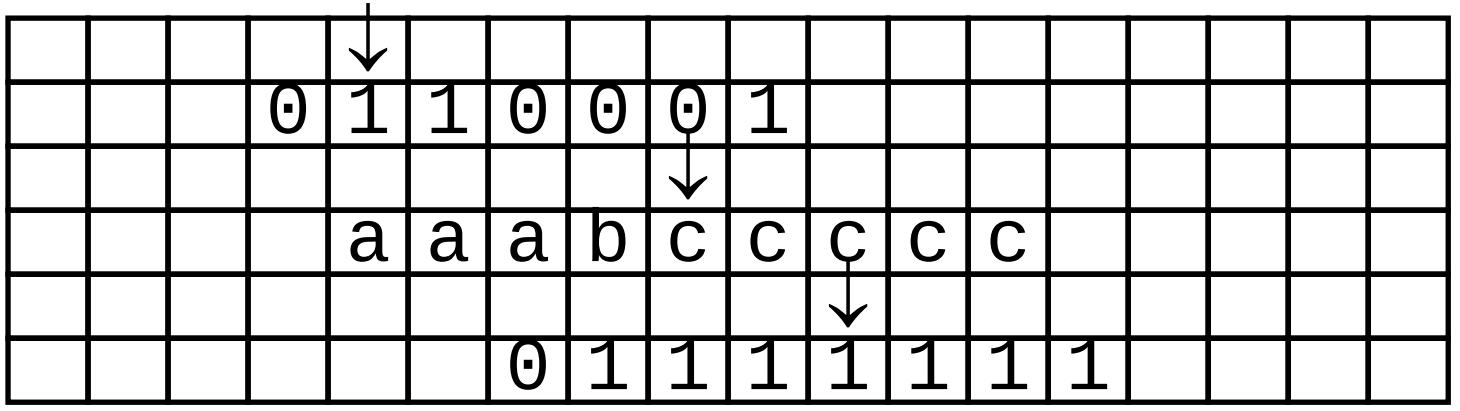
Transitions are determined by the current state and the contents of all scanned cells (one on each tape).

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.

Simulating ND-TM's

An ND-TM 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 $\omega_1, \ldots, \omega_k$.

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

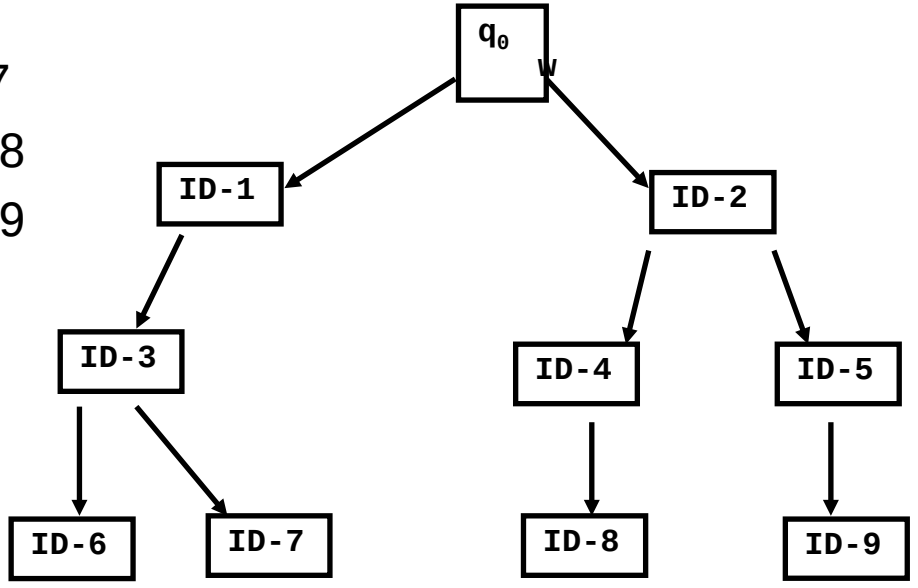
|-|- ID-2 ⊗ ID-3

|-|- 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 $\omega_1, \ldots, \omega_k$.

See the textbook for more details.