Enumerations and Turing Machines

Mridul Aanjaneya

Stanford University

August 07, 2012

Finite Sets

- Intuitively, a finite set is a set for which there is a particular integer that is the count of the nuber of members.
- **Example:** {a, b, c} is a finite set, its cardinality is 3.
- It is impossible to find a 1-1 mapping between a finite set and a proper subset of itself.

Infinite Sets

- Formally, an infinite set is a set for which there is a 1-1 correspondence between itself and a proper subset of itself.
- **Example:** the positive integers $\mathbb{Z} = \{1, 2, 3, \ldots\}$ is an infinite set.
  - There is a 1-1 correspondence between $1 \leftrightarrow 2$, $2 \leftrightarrow 4$, $3 \leftrightarrow 6$, $\ldots$ between this set and a proper subset (the set of even integers).

Countable Sets

- A countable set is a set with a 1-1 correspondence with the positive integers $\mathbb{Z}^+$.
  - Hence, all countable sets are infinite.
- **Example:** All integers.
  - $0 \leftrightarrow 1$, $-i \leftrightarrow 2i$, $+i \leftrightarrow 2i+1$.
  - Thus, order is 0, -1, 1, -2, 2, -3, 3, ...

Enumerations

- An enumeration of a set if a 1-1 correspondence between the set and the positive integers $\mathbb{Z}^+$.

How Many Languages?

- Are the languages over $\{0,1\}^*$ countable?
- No, here's a proof.
- Suppose we could enumerate all languages over $\{0,1\}^*$ and talk about the *i*th language.
- Consider the language $L = \{w \mid w$ is the *i*th binary string and w is not in the *i*th language$\}$.

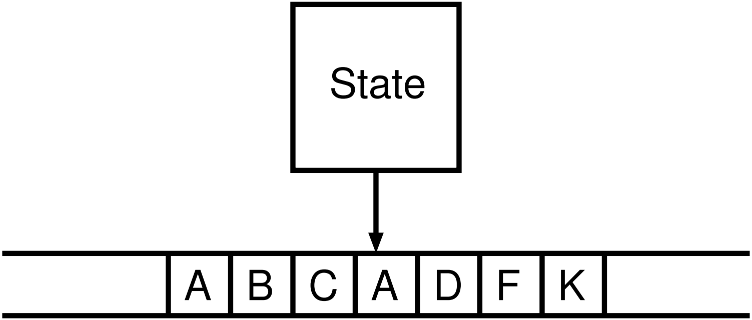
Proof

- Clearly, L is a language over $\{0,1\}^*$.
- Thus, it is the *j*th language for some particular *j*.
- Let x be the *j*th string.
- Is x in L?
  - If so, x is not in L (by definition).
  - If not, then x is in L (by definition).
- We have a contradiction: x is neither in L nor not in L, so our sole assumption (that there was an enumeration of the languages) is wrong.

Turing-Machine Theory

- The purpose of the theory of Turing Machines is to prove that certain specific languages have no algorithm.
- Start with a language about Turing Machines themselves.
- Reductions are used to prove more common questions undecidable.

Picture of a Turing Machine



Why Turing Machines?

- Why not deal with C programs or something like that?
- Answer: You can, but it is easier to prove things about TM's, because they are so simple.
  - And yet they are as powerful as any computer.
  - More so, in fact, since they have infinite memory.

Then why not FSM's to model computers?

- In principle, you could, but it is not instructive.
- Programming models don't build in a limit on memory.
- In practice, you can do to Fry's and buy another disk.
- But finite automata vital at the chip level (model-checking).

Turing-Machine Formalism

- A TM is described by:
  1. A finite set of states (Q, typically).
  2. An input alphabet (Σ, typically).
  3. A tape alphabet (Γ, typically).
  4. A transition function (δ, typically).
  5. A start state (q_0, in Q, typically).
  6. A blank symbol (B, in $\Gamma - \Sigma$, typically).
     - All tape except for the input is blank initially.
  7. A set of final states ($F \subseteq Q$, typically).

Conventions

- a, b,... are input symbols.
- ..., X, Y, Z are tape symbols.
- ..., w, x, y, z are strings of input symbols.
- α, β,... are strings of tape symbols.

The Transition Function

- Takes two arguments:
  1. A state in Q.
  2. A tape symbol in Γ.
- δ(q,Z) is either undefined or a triple of the form (p,Y,D).
  - p is a state.
  - Y is the new tape symbol.
  - D is a direction, L or R.

Actions of the TM

- If δ(q,Z) = (p,Y,D) then, in state q, scanning Z under its tape head, the TM:
  - Changes the state to p.
  - Replaces Z by Y on the tape.
  - Moves the head one square in direction D.
  - D = L: move left, D = R: move right.

Example: Turing Machine

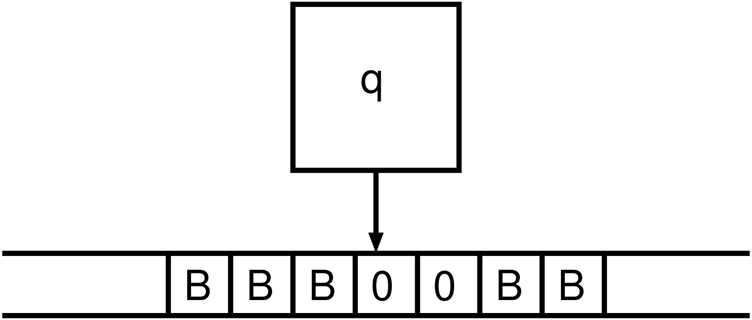
- This TM scans its input right, looking for a 1.
- If it finds one, it changes it to a 0, goes to final state f, and halts.
- If it reaches a blank, it changes it to a 1 and moves left.

Example: Turing Machine

- States = {q, f}.
- Input symbols = {0, 1}.
- Tape symbols = {0, 1, B}.
- δ(q,0) = (q,0,R).
- δ(q,1) = (f,0,R).
- δ(q,B) = (q,1,L).

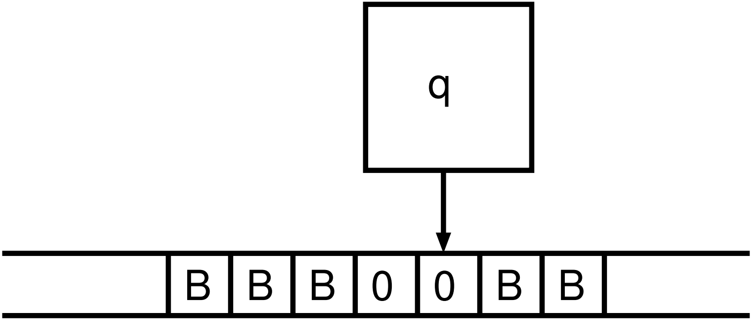
Example: Turing Machine

- $\delta(q,0) = (q,0,R)$.



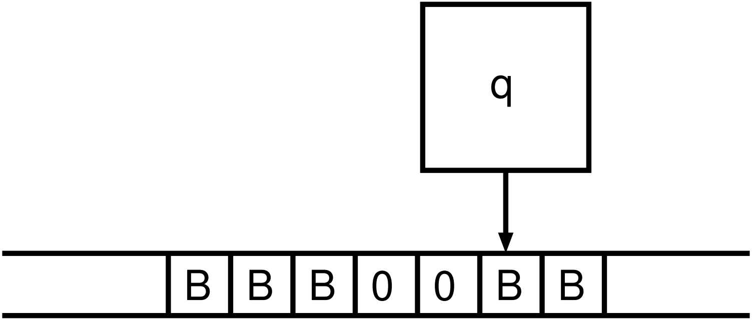
Example: Turing Machine

- $\delta(q,0) = (q,0,R)$.



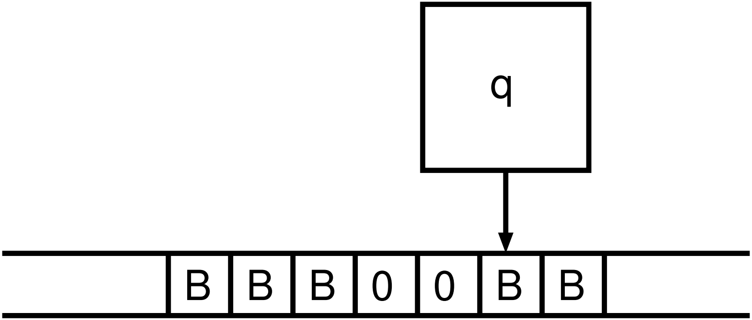
Example: Turing Machine

- $\delta(q,0) = (q,0,R)$.



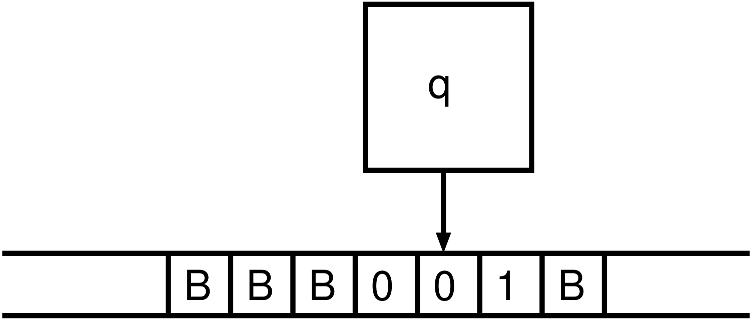
Example: Turing Machine

- δ(q,B) = (q,1,L).



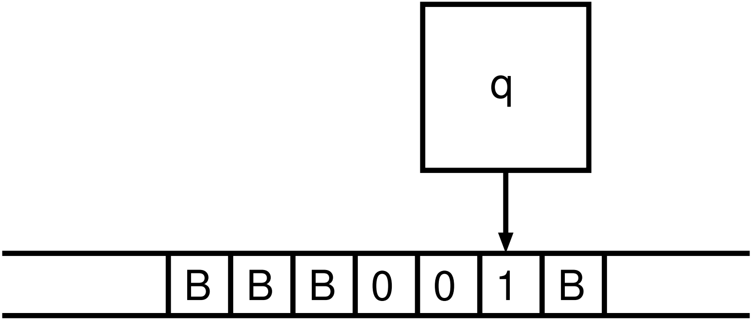
Example: Turing Machine

- $\delta(q,0) = (q,0,R)$.



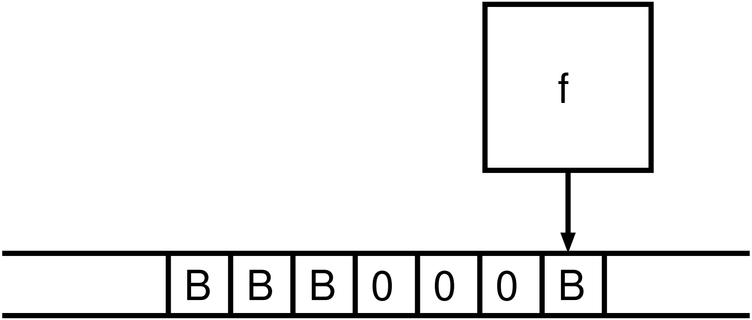
Example: Turing Machine

- $\delta(q,1) = (f,0,R)$.



Example: Turing Machine

- δ(q,1) = (f,0,R).



- Initially, a TM has a tape consisting of a string of input symbols surrounded by an infinity of blanks in both directions.
- The TM is in the start state, and the head is at the leftmost input symbol.

Instantaneous Descriptions of a Turing Machine

- An ID is a string $\alpha \mathrm{q} \beta$, where $\alpha\beta$ is the tape between the leftmost and rightmost nonblanks (inclusive).
- The state q is immediately to the left of the tape symbol scanned.
- If q is at the right end, it is scanning B.
    - If q is scanning a B at the left end, then consecutive B's at and to the right of q are part of α.

Instantaneous Descriptions of a Turing Machine

- As for PDA's we may use symbols $\vdash$ and $\vdash^*$ to represent becomes in one move and becomes in zero or more moves, respectively, on ID's.
- **Example:** The moves of the previous TM are $q00 \vdash 0q0 \vdash 00q \vdash 0q01 \vdash 00q1 \vdash 000f$.

Formal Definition of Moves

1. If $\delta(q,Z) = (p,Y,R)$, then
   - $\alpha qZ\beta \vdash \alpha Yp\beta$
   - If Z is the blank B, then also $\alpha q \vdash \alpha Yp$
2. If $\delta(q,Z) = (p,Y,L)$, then
   - For any X, $\alpha XqZ\beta \vdash \alpha pXY\beta$
   - In addition, $qZ\beta \vdash pBY\beta$

Languages of a TM

- A TM defines a language by final state as usual.
  - $L(M) = \{w \mid q_0w \vdash^* I$, where I is an ID with a final state$\}$.
- Or, a TM can accept a language by halting.
  - $H(M) = \{w \mid q_0w \vdash^* I$, and there is no move from ID $I\}$.

Equivalence of Acceptance and Halting

1. If $L = L(M)$, then there is a TM M' such that $L = H(M')$.
2. If $L = H(M')$, then there is a TM M'' such that $L = L(M'')$.

Acceptance → Halting

- Modify M to become M' as follows:
  1. For each accepting state of M, remove any moves, so M' halts in that state.
  2. Avoid having M' accidentally halt.
  3. Introduece a new state s, which runs to the right forever, i.e., δ(s,X) = (s,X,R) for all symbols X.
  4. If q is not accepting, and δ(q,X) is undefined, let δ(q,X) = (s,X,R).

Halting $\rightarrow$ Acceptance

- Modify M to become M" as follows:
  1. Introduce a new state f, the only accepting state of M".
  2. f has no moves.
  3. If δ(q,X) is undefined for any state q and symbol X, define it by δ(q,X) = (f,X,R).

Recursively Enumerable Languages

- We now see that the classes of languages defined by TM's using final state and halting are the same.
- This class of languages is called the recursively enumerable languages.
    - Why? The term actually predates the Turing Machine and refers to another notion of computation of functions.

Recursive Languages

- An algorithm is a TM that is guaranteed to halt whether or not it accepts.
- If $L = L(M)$ for some TM M that is an algorithm, we say L is a recursive language.
  - Why? Again, don't ask. It is a term with a history.

Example: Recursive Languages

- Every CFL is a recursive language.
  - Use the CYK algorithm.
- Every regular language is a CFL (think of its DFA as a PDA that ignores its stack); therefore every regular language is recursive.
- Almost anything you can think of is recursive.