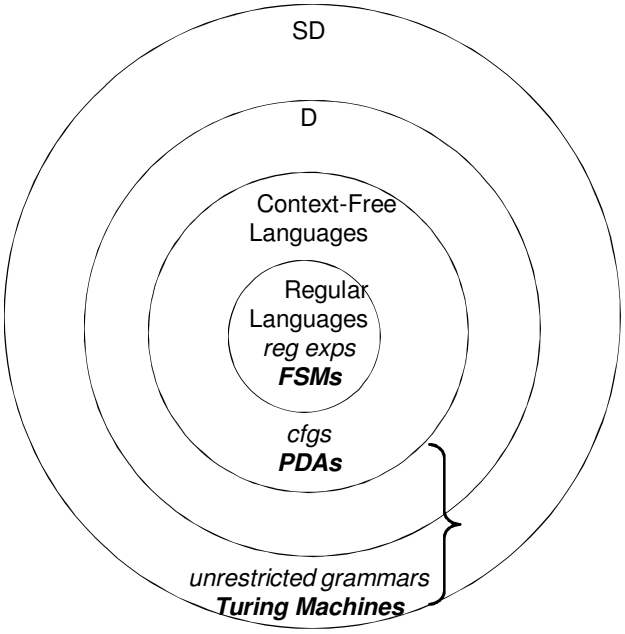
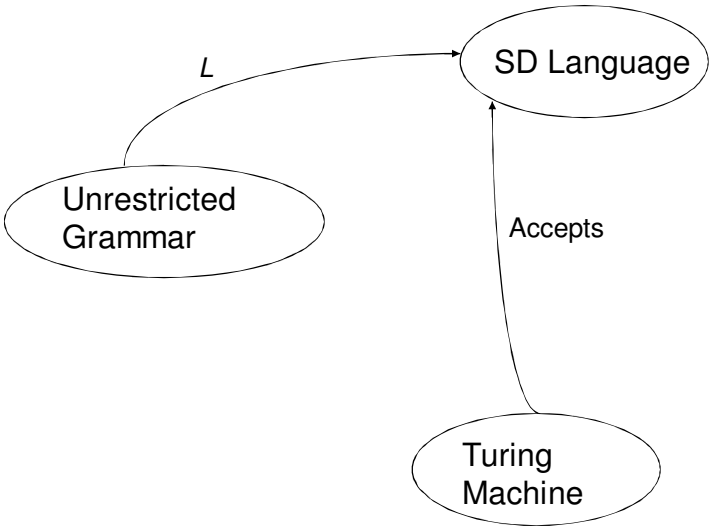
MA/CSSE 474
Theory of Computation

Turing Machines Intro

Languages and Machines



Grammars, SD Languages, and Turing Machines



Turing Machines

We want a new kind of automaton:

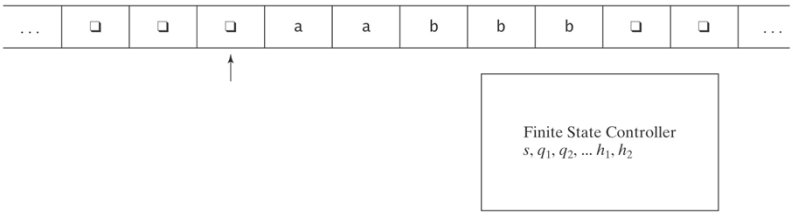
- powerful enough to describe all computable things

  unlike FSMs and PDAs.

- simple enough that we can reason formally about it
  like FSMs and PDAs,
  unlike real computers.

Goal: Be able to prove things about what can and cannot be computed.

Turing Machines



At each step, the machine must:

- choose its next state,
- write on the current square, and
- move left or right.

A Formal Definition

A (deterministic) Turing machine M is $(K, \Sigma, \Gamma, \delta, s, H)$:

- K is a finite set of states;
- Σ is the input alphabet, which does not contain ❑;
- Γ is the tape alphabet, which must contain ❑ and have Σ as a subset.
- $s \in K$ is the initial state;
- $H \subseteq K$ is the set of halting states;
- δ is the transition function:

$(K - H) \times \Gamma$ to $K \times \Gamma \times \{\rightarrow, \leftarrow\}$

non-halting state × tape char → state × tape char × direction to move (R or L)

Notes on the TM Definition

1. The input tape is infinite in both directions.

2. The machine starts at the blank square just to the left of the first input character.

2. δ is a function, not a relation. So this is a definition for deterministic Turing machines.

3. δ must be defined for all (state, input) pairs unless the state is a halting state.

4. Turing machines do not necessarily halt (unlike FSM's and most PDAs). Why? To halt, they must enter a halting state. Otherwise they loop.

5. Turing machines generate output, so they can compute functions.

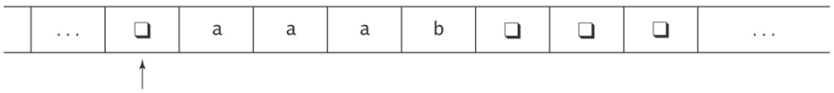
An Example

M takes as input a string in the language:

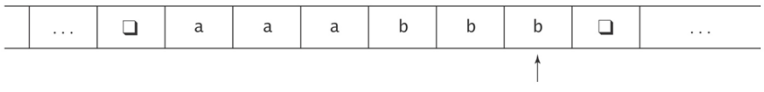
$\{a^i b^j, 0 \le j \le i\}$,

and adds b's as required to make the number of b's equal the number of a's.

The input to *M* will look like this:

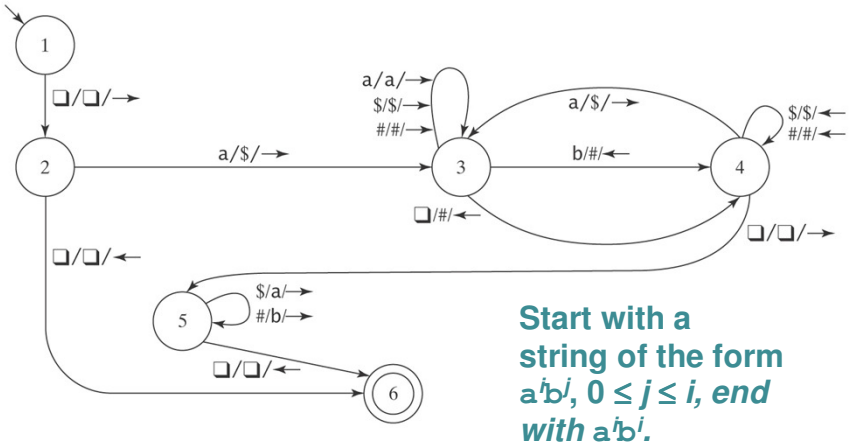


The output should be:



The Details

$K = \{1, 2, 3, 4, 5, 6\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, ❑, \$, \#\}$,
$s = 1$, $H = \{6\}$, $\delta =$



Start with a string of the form $a^i b^j$, $0 \le j \le i$, *end with* $a^i b^i$.

Notes on Programming

The machine has a strong procedural feel, with one phase coming after another.

There are common idioms, like scan left until you find a blank

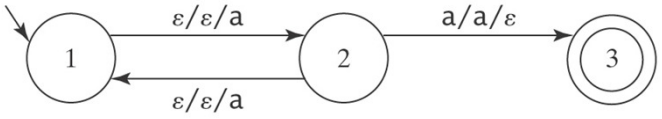
There are two common ways to scan back and forth marking things off.

Often there is a final phase to fix up the output.

Even a very simple machine is a nuisance to write.

Halting

- A DFSM *M*, on input *w*, is guaranteed to halt in |*w*| steps.
- A PDA *M*, on input *w*, is not guaranteed to halt. To see why, consider again *M* =



But there exists an algorithm to construct an equivalent PDA *M´* that is guaranteed to halt.

A TM *M*, on input *w*, is not guaranteed to halt. And there is no algorithm that, given a TM M, will always construct an equivalent TM that is guaranteed to halt.

Formalizing the Operation

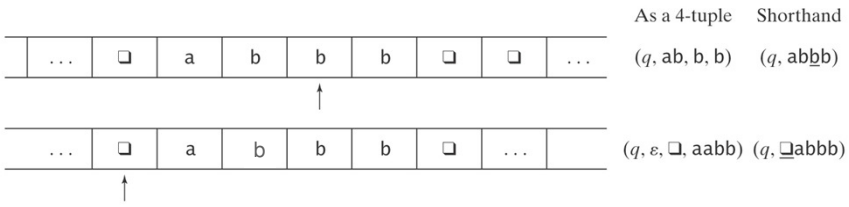
A *configuration* of a Turing machine

M = (*K*, Σ, Γ, *s*, *H*) is an element of:

$$K \times ((\Gamma - \{\square\})\,\Gamma^*) \cup \{\varepsilon\} \times \Gamma \times (\Gamma^*\,(\Gamma - \{\square\})) \cup \{\varepsilon\}$$

state	up to current square	current square	after current square

Example Configurations



(1) (q, ab, b, b) = (q, abb̲b)
(2) (q, ε, ❑, aabb) = (q, ❑̲aabb)

Initial configuration is (s, ❑̲w).

Yields

(q_1, w_1) $\vdash_M$ (q_2, w_2) iff (q_2, w_2) is derivable, *via* δ, in one step.

For any TM M, let $\vdash_M^*$ be the reflexive, transitive closure of $\vdash_M$.

Configuration C_1 ***yields*** configuration C_2 if: $C_1 \vdash_M^* C_2$.

A ***path*** through M is a sequence of configurations C_0, C_1, …, C_n for some $n \geq 0$ such that C_0 is the initial configuration and:

$C_0 \vdash_M C_1 \vdash_M C_2 \vdash_M \ldots \vdash_M C_n$.

A **computation** by M is a path that halts.

If a computation is of *length n* (has n steps), we write:

$C_0 \vdash_M^n C_n$

Exercise

A TM to recognize { ww^R : w ∈ {a, b}* }.

If the input string is in the language, the machine should halt with y as its current tape symbol

If not, it should halt with n as its current tape symbol.

The final symbols on the rest of the tape may be anything.

TMs are complicated

- … and low-level!
- We need higher-level "abbreviations".
  - Macros
  - Subroutines

A Macro language for Turing Machines

(1) Define some basic machines

- Symbol writing machines

  For each $x \in \Gamma$, define M_x, written just x, to be a machine that writes x.

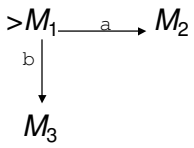
- Head moving machines

  R: for each $x \in \Gamma$, $\delta(s, x) = (h, x, \rightarrow)$
  L: for each $x \in \Gamma$, $\delta(s, x) = (h, x, \leftarrow)$

- Machines that simply halt:
  h, which simply halts (don't care whether it accepts).
  n, which halts and rejects.
  y, which halts and accepts.

Turing Machines Macros Cont'd

Example:



- Start in the start state of M_1.
- Compute until M_1 reaches a halt state.
- Examine the tape and take the appropriate transition.
- Start in the start state of the next machine, etc.
- Halt if any component reaches a halt state and has no place to go.
- If any component fails to halt, then the entire machine may fail to halt.