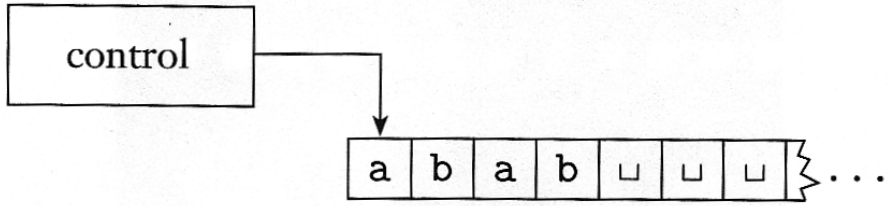
Turing Machines

A Turing machine is a FA with an infinite tape as memory.



Initially, the tape contains the input to the Turing machine.

The following list summarizes the differences between finite automata and Turing machines.

1. A Turing machine can both write on the tape and read from it.
2. The read–write head can move both to the left and to the right.
3. The tape is infinite.
4. The special states for rejecting and accepting take effect immediately.

Church-Turing Thesis

Why do we study Turing machines?

Intuitive Notion of Algorithms
equals
Turing Machine Algorithms

This equivalence cannot be proved but up to now no algorithm has been found that could not be implemented on a Turing machine.

Turing Machines

Example: Construct a TM, call it M_1, that tests whether a string is a member of the language $B = \{u\#u \mid u \in \{0,1\}^*\}$. That is, if some string $w \in B$ then *accept* otherwise *reject*. Assume that the string w is loaded on the tape before the machine runs; the tape will look something like this for $w = 101\#101$,

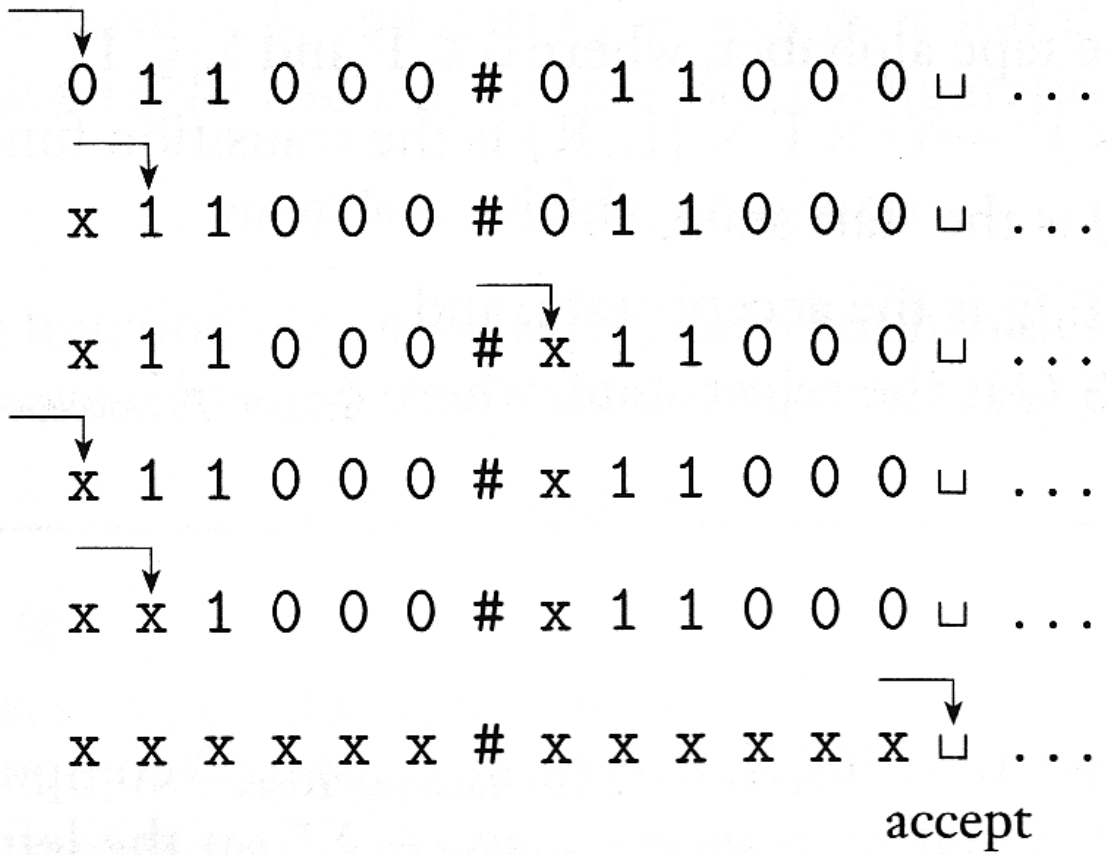
$$1\ 0\ 1\ \#\ 1\ 0\ 1\ \sqcup \ldots$$

Algorithm:

$M_1 =$ "On input string w:

1. Zig-zag across the tape to corresponding positions on either side of the # symbol to check whether these positions contain the same symbol. If they do not, or if no # is found, *reject*. Cross off symbols as they are checked to keep track of which symbols correspond.
2. When all symbols to the left of the # have been crossed off, check for any remaining symbols to the right of the #. If any symbols remain, *reject*; otherwise, *accept*."

Turing Machines



Turing Machines

Example: Construct a machine M that tests whether a string belongs to the language

$$L = \{a^i b^j c^k \mid i, j, k \geq 0 \wedge i = k = j\}.$$

Algorithm:

$M =$ "On input string w:

1. Scan across the tape and make sure the a's, b's, and c's are properly ordered.
2. Scan across the tape and count the numbers of a's, b's, and c's. If the numbers do not match, *reject*.
3. Otherwise, *accept*."

Formal Definition

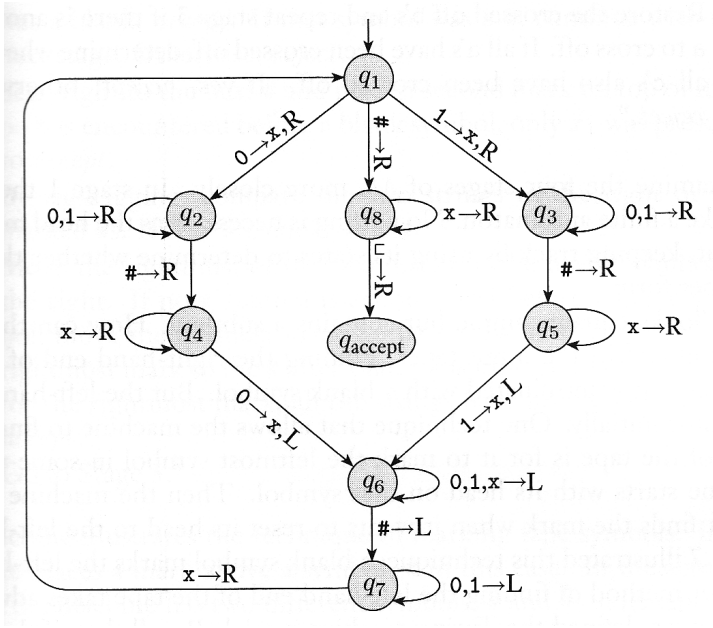
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta: Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

NOTE: A TM computes until it enters either an accept or reject state.

M_1 Revisited

$L(M_1) = \{u\#u \mid u \in \{0,1\}^*\}$



NOTE: Transition into q_{reject} are implicit on symbols not appearing at states.

Configurations

NOTE: The current state, the current tape contents, and the current head location is called a ***configuration***.

start configuration: the TM is in state q_0 with the tape head pointing to the leftmost position.

halting configuration: a state in which the machine is either in an accept state (**accepting configuration**) or in a reject state (**rejecting configuration**)

TM's & Languages

Three outcomes are possible when a TM computes: accept, reject, or loop.

This lead to the following definitions.

> **Definition:** Call a language ***Turing-recognizable*** if some Turing machine recognizes it.

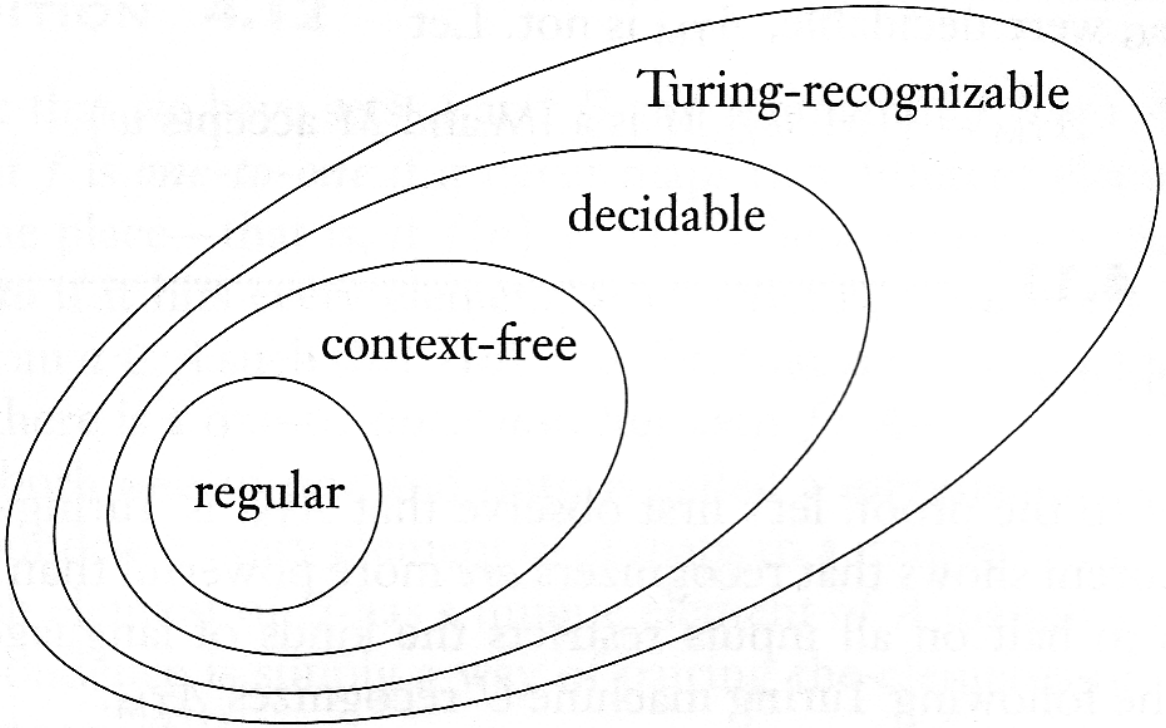
The problem with TM's that recognize a language is that they might loop on some inputs. We prefer machines that always halt. We call such machines ***deciders***.

> **Definition:** Call a language ***Turing-decidable*** or simply ***decidable*** if some Turing machine decides it.

Deciders will answer definitively on the question whether a string belongs to a language or not.

NOTE: Every decidable language is Turing-recognizable (why?).

Language Hierarchy



Generators

The generators associated with TMs are *unconstrained grammars* or *type-0* grammars.

Recall that a type-0 grammar $G = (V, \Sigma, R, s)$ is a grammar where the shape of each rule in R is completely unrestricted:

$$\alpha \rightarrow \beta$$

with $\alpha, \beta \in (V \cup \Sigma)^*$.

Generators

Example: The non-context-free language

$$L = \{a^i b^j c^k \mid i, j, k \geq 0\}$$

is generated by the following type-0 grammar $G = (V, \Sigma, R, s)$,

- $V = \{S, B\}$,
- $\Sigma = \{a, b, c\}$,
- The rule set R is as follows,

$$\begin{array}{rcl} S & \rightarrow & aBSc \\ S & \rightarrow & abc \\ S & \rightarrow & \epsilon \\ Ba & \rightarrow & aB \\ Bb & \rightarrow & bb \end{array}$$

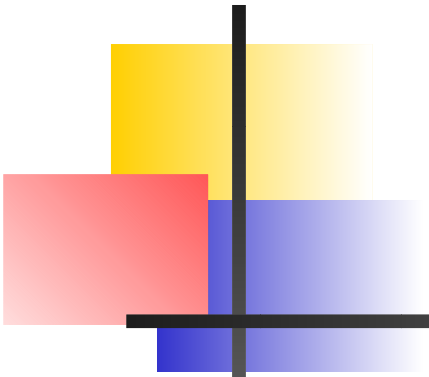
- $s = S$.

Generators

Observation: Unrestricted term rewriting systems are *Turing complete*, that is they can express the same computations that Turing machines can express.[a] [b]

[a] Perhaps not a surprise, because Algebra, Calculus etc. are all just very fancy rewriting systems.

[b] Later on we will discuss an interesting term rewriting system called the lambda calculus which basically models computing with functions.



Assignment

Assignment #2 – see web