CSE 2001:
Introduction to Theory of Computation

Winter 2013

Suprakash Datta

datta@cse.yorku.ca

Office: CSEB 3043

Phone: 416-736-2100 ext 77875

Course page: http://www.cse.yorku.ca/course/2001

Next

- Closure properties of CFL

Union Closure Properties

Lemma: Let A_1 and A_2 be two CF languages, then the *union* $A_1 \cup A_2$ is context free as well.

Proof: Assume that the two grammars are $G_1=(V_1,\Sigma,R_1,S_1)$ and $G_2=(V_2,\Sigma,R_2,S_2)$.
Construct a third grammar $G_3=(V_3,\Sigma,R_3,S_3)$ by:
$V_3 = V_1 \cup V_2 \cup \{ S_3 \}$ (new start variable) with
$R_3 = R_1 \cup R_2 \cup \{ S_3 \to S_1 \mid S_2 \}$.

It follows that $L(G_3) = L(G_1) \cup L(G_2)$.

Intersection & Complement?

Let A_1 and A_2 be two CF languages.

We will prove that, *in general,*

the intersection $A_1 \cap A_2$,

and

the complement $\bar{A}_1 = \Sigma^* \setminus A_1$

are not context free languages.

One proves this with specific counter examples of languages.

Intersection of CFLs

Let $A_1 = \{a^m b^n c^n \mid m,n \geq 0\}$ and $A_2 = \{a^n b^n c^m \mid m,n \geq 0\}$ be two CF languages.

Then the intersection $A_1 \cap A_2 = \{a^n b^n c^n \mid n \geq 0\}$ is not a CFL.

Complements of CFLs

Consider the complement of L = {ww| w is a binary string}

L is not a CFL (proved earlier)

L^c is a CFL.

Complements of CFLs - 2

Suppose that CFLs are closed under complementation. Then for CFLs A_1, A_2, the languages $\bar{A}_1$, $\bar{A}_2$ are CFLs.

So $\bar{A}_1 \cup \bar{A}_2$ is a CFL.
Therefore its complement is a CFL. By de Morgan's laws, this is the language $A_1 \cap A_2$.

This is a contradiction. So CFLs are not closed under complementation.

What do we really know?

Can we always decide if a language L is regular/ context-free or not?

We know:

$\{ 1^x \mid x = 0 \bmod 7 \}$ is regular

$\{ 1^x \mid x \text{ is prime} \}$ is not regular

But what about

$\{ 1^x \mid x \text{ and } x+2 \text{ are prime} \}$?

This is (yet) unknown.

Describing a Language

The problem lies in the informal notion of a description.

Consider:

$\{ n \mid \exists a,b,c: a^n+b^n = c^n \}$

{ x | in year x the first female US president was elected}

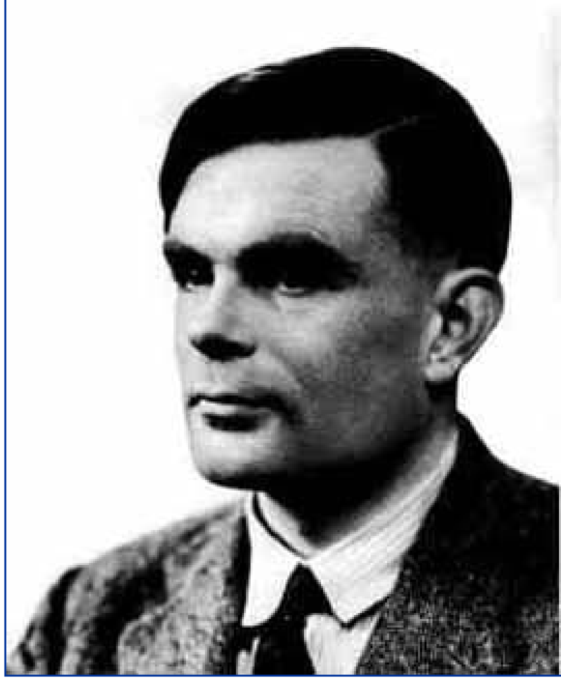
{ x | x is "an easy to remember number" }

We have to define what we mean by "description" and "method of deciding".

Next

- Computability (Ch 3)
  - Turing machines
  - TM-computable/recognizable languages
  - Variants of TMs

Turing Machines



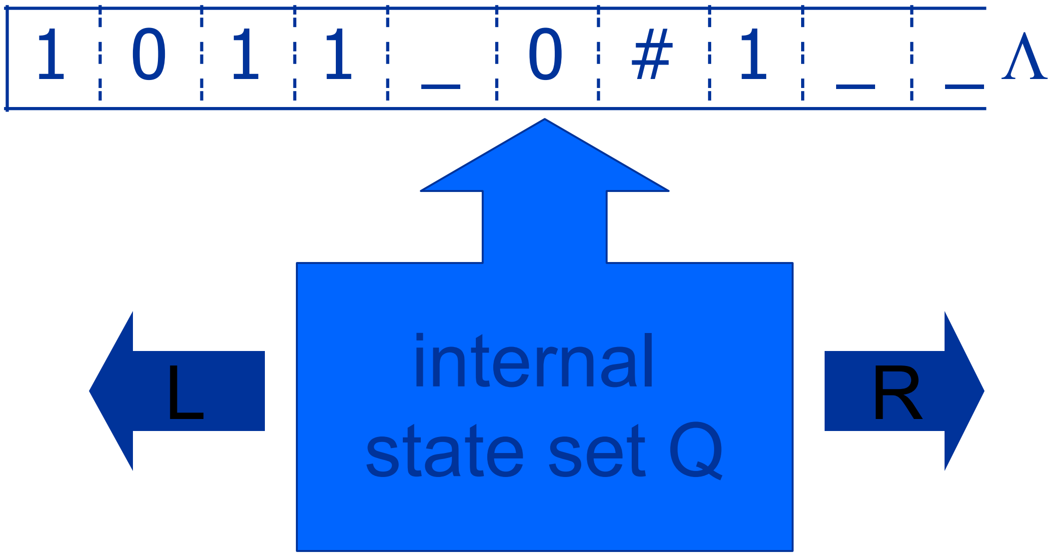
After Alan M. Turing (1912–1954)

In 1936, Turing introduced his abstract model for computation in his article "*On Computable Numbers, with an application to the Entscheidungsproblem*".

At the same time, Alonzo Church published similar ideas and results.

However, the Turing model has become the standard model in theoretical computer science.

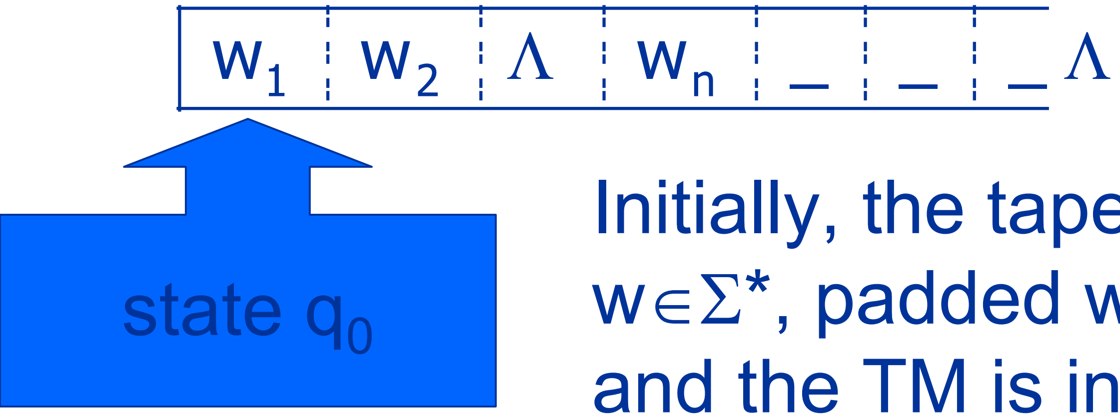
Informal Description TM



At every step, the head of the TM M reads a letter x_i from the one-way infinite tape.

Depending on its state and the letter x_i, the TM
- writes down a letter,
- moves its read/write head left or right, and
- jumps to a new state.

Input Convention

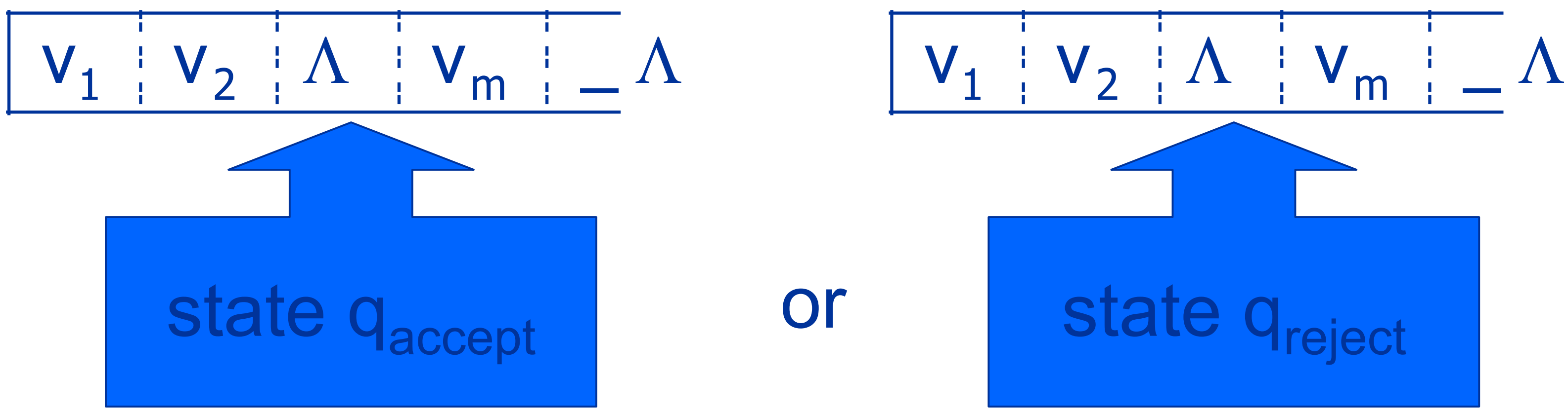


Initially, the tape contains the input $w \in \Sigma^*$, padded with blanks "_", and the TM is in start state q_0.

During the computation, the head moves left and right (but not beyond the leftmost point), the internal state of the machine changes, and the content of the tape is rewritten.

Output Convention

The computation can proceed indefinitely, or the machines reaches one of the two halting states:



Major differences with FA, PDA

- Input can be read more than once
- Scratch memory available, can be accessed without restrictions
- The "running time" is not predictable from the input – the machine can "churn" for a long time even on a short input
- So we need a clear indicator of end of computation

Turing Machine (Def. 3.3)

A Turing machine M is defined by a 7-tuple $(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$, with

- Q finite set of states
- Σ finite input alphabet (without "_")
- Γ finite tape alphabet with $\{\_\} \cup \Sigma \subseteq \Gamma$
- q_0 start state $\in Q$
- q_{accept} accept state $\in Q$
- q_{reject} reject state $\in Q$
- δ the transition function

$$\delta: Q \times \Gamma \to Q \times \Gamma \times \{L,R\}$$

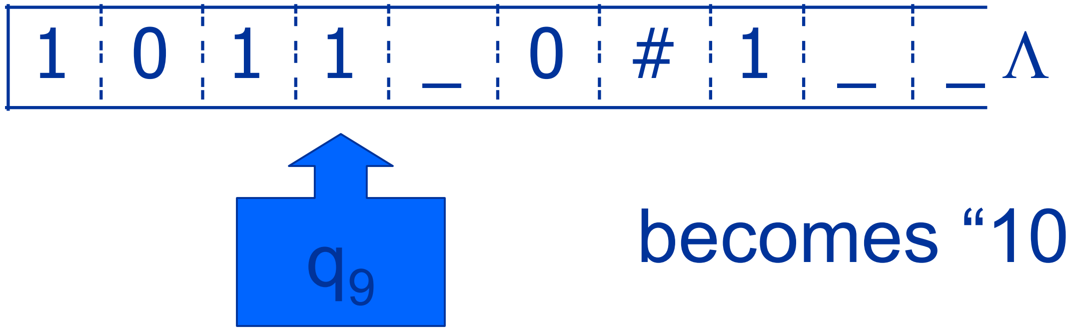
Why do you need these?

Configuration of a TM

The configuration of a Turing machine consists of

- the current state $q \in Q$
- the current tape contents $\in \Gamma^*$
- the current head location $\in \{0,1,2,\ldots\}$

This can be expressed as an element of $\Gamma^* \times Q \times \Gamma^*$:



becomes "101 q_9 1_0#1"

An Elementary TM Step

Let $u,v \in \Gamma^*$; $a,b,c \in \Gamma$; $q_i,q_j \in Q$, and M a TM with transition function δ.

We say that the configuration "ua q_i bv" yields the configuration "uac q_j b" if and only if:

$\delta(q_i,b) = (q_j,c,R)$.

Similarly, "ua q_i bv" yields "u q_j acb" if and only if

$\delta(q_i,b) = (q_j,c,L)$.

Terminology

starting configuration on input w: “q_0w”

accepting configuration: “$uq_{accept}v$”

rejecting configuration: “$uq_{reject}v$”

The accepting and rejecting configurations are the *halting configurations*.

Accepting TMs

A Turing machine M <u>accepts</u> input $w \in \Sigma^*$ if and only if there is a finite sequence of configurations $C_1, C_2, \ldots, C_k$ with

- C_1 the starting configuration "$q_0 w$"
- for all $i=1,\ldots,k-1$ C_i yields C_{i+1} (following M's δ)
- C_k is an accepting configuration "$u q_{accept} v$"

The language that consists of all inputs that are accepted by M is denoted by L(M).

Turing Recognizable (Def. 3.5)

A language L is Turing-recognizable if and only if there is a TM M such that L=L(M).

Also called: a recursively enumerable language.

Note: On an input $w \notin L$, the machine M can halt in a rejecting state, or it can 'loop' indefinitely.

How do you distinguish between a very long computation and one that will never halt?

Turing Decidable (Def. 3.6)

A language L=L(M) is decided by the TM M if on every w, the TM finishes in a halting configuration. (That is: q_{accept} for $w \in L$ and q_{reject} for all $w \notin L$.)

A language L is Turing-decidable if and only if there is a TM M that decides L.

Also called: a recursive language.

Example 3.7: $A = \{ 0^j \mid j=2^n \}$

Approach: If j=0 then "reject"; If j=1 then "accept"; if j is even then divide by two; if j is odd and >1 then "reject". Repeat if necessary.

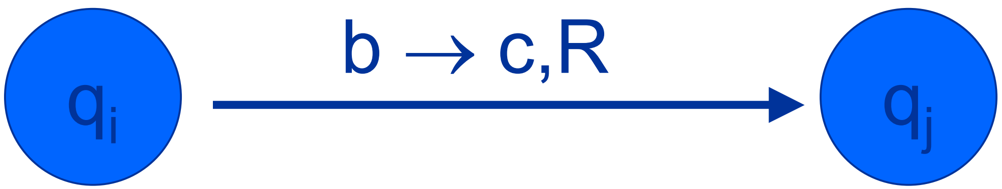
1. Sweep left to right crossing off every other zero.
    1. If the tape has a single 0, accept.
    2. Else If there are an odd number of zeros reject.
2. Return the head to the left-hand end of the tape.
3. goto 1

State diagrams of TMs

Like with PDA, we can represent Turing machines by (elaborate) diagrams.

See Figures 3.8 and 3.10 for two examples.

If transition rule says: $\delta(q_i,b) = (q_j,c,R)$, then:



When Describing TMs

It is assumed that you are familiar with TMs and with programming computers.

Clarity above all: high level description of TMs is allowed but should not be used as a trick to hide the important details of the program.

Standard tools: Expanding the alphabet with separator "#", and underlined symbols $\underline{0}$, $\underline{a}$, to indicate 'activity'. Typical: $\Gamma = \{ 0,1,\#,\_,\underline{0},\underline{1} \}$

Some more examples

- $B=\{w\#w \mid w \in (0,1)^* \}$ (Pg 172)

- $C = \{a^i b^j c^k \mid i*j=k,\ i,j,k >= 1\}$ (Pg 174)

Turing machine variants

- Multiple tapes
- 2-way infinite tapes
- Non-deterministic TMs

Multitape Turing Machines

A k-tape Turing machine M has k different tapes and read/write heads. It is thus defined by the 7-tuple $(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$, with

- Q finite set of states
- Σ finite input alphabet (without "_")
- Γ finite tape alphabet with $\{ \_ \} \cup \Sigma \subseteq \Gamma$
- q_0 start state $\in Q$
- q_{accept} accept state $\in Q$
- q_{reject} reject state $\in Q$
- δ the transition function

$$\delta: Q \setminus \{q_{accept},q_{reject}\} \times \Gamma^k \to Q \times \Gamma^k \times \{L,R\}^k$$

k-tape TMs versus 1-tape TMs

Theorem 3.13: For every multi-tape TM M, there is a single-tape TM M' such that L(M)=L(M').
Or, for every multi-tape TM M, there is an equivalent single-tape TM M'.

Proving and understanding these kinds of robustness results, is essential for appreciating the power of the Turing machine model.

From this theorem Corollary 3.9 follows:
A language L is TM-recognizable if and only if some multi-tape TM recognizes L.

Outline Proof Thm. 3.13

Let $M=(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$ be a k-tape TM.
Construct 1-tape M' with expanded $\Gamma' = \Gamma \cup \underline{\Gamma} \cup \{\#\}$

Represent M-configuration

$$u_1 q_j a_1 v_1, \quad u_2 q_j a_2 v_2, \quad \ldots, \quad u_k q_j a_k v_k$$

by M' configuration,

$$q_j \# u_1 \underline{a}_1 v_1 \# u_2 \underline{a}_2 v_2 \# \ldots \# u_k \underline{a}_k v_k$$

(The tapes are seperated by #, the head positions are marked by underlined letters.)

Proof Thm. 3.13 (cont.)

On input $w = w_1 \ldots w_n$, the TM M' does the following:

- Prepare initial string: $\#\underline{w}_1 \ldots w_n \#\_\#\Lambda\#\_\#\_\ \Lambda$
- Read the underlined input letters $\in \Gamma^k$
- Simulate M by updating the input and the underlining of the head-positions.
- Repeat 2-3 until M has reached a halting state
- Halt accordingly.

PS: If the update requires overwriting a # symbol, then shift the part $\#\ \Lambda\_$ one position to the right.

Non-deterministic TMs

A nondeterministic Turing machine M can have several options at every step. It is defined by the 7-tuple $(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$, with

- Q finite set of states
- Σ finite input alphabet (without "_")
- Γ finite tape alphabet with $\{ \_ \} \cup \Sigma \subseteq \Gamma$
- q_0 start state $\in Q$
- q_{accept} accept state $\in Q$
- q_{reject} reject state $\in Q$
- δ the transition function

$$\delta: Q \backslash \{q_{accept},q_{reject}\} \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L,R\})$$