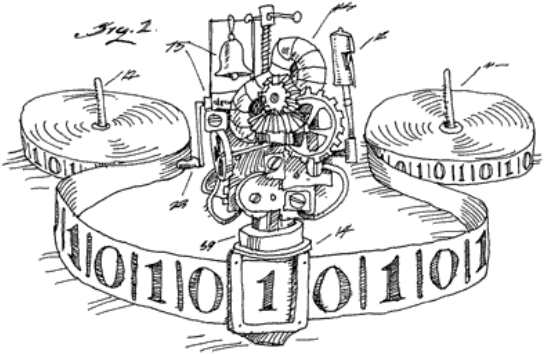
Automata Theory :: Turing Machines

Jörg Endrullis

Vrije Universiteit Amsterdam

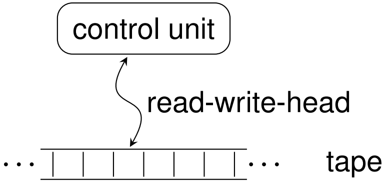
Turing Machines



Turing Machines

Turing machines can read and write the input word.

Input is written on a tape on which a read-write-head works.



In each step:

- the read-write-head reads a symbol from the tape,
- overwrites the symbol, and
- moves one place to the left or right.

The tape is two-sided infinite: unlimited memory!

Turing Machines

We introduce a **blank symbol** $\Box$. The initial tape content is

$$\cdots \Box\Box\Box\Box \;\text{ input word }\; \Box\Box\Box\Box \cdots$$

There is a finite set of states Q and a finite tape alphabet Γ.

The transition function δ has the form

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

Here δ is a **partial function**: $\delta(q, a)$ may be undefined.

$\delta(q, a) = (q', b, X)$ means: if

- the machine is in state q, and
- the head reads a from the tape

then

- then a is overwritten by b,
- the head moves 1 position left if $X = L$, right if $X = R$, and
- the machine switches to state q'.

Turing Machines

A **deterministic Turing machine**, short TM, is a 7-tuple

$$(Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$$

where

- Q is a finite set of states,
- $\Sigma \subseteq \Gamma \setminus \{ \Box \}$ a finite input alphabet,
- Γ a finite tape alphabet,
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{ L, R \}$ a partial transition function,
- q_0 the starting state,
- $\Box \in \Gamma$ the blank symbol,
- $F \subseteq Q$ a set of final (accepting) states.

Assumption: $\delta(q, a)$ is undefined for every $q \in F$ and $a \in \Gamma$.

So the computation stops when reaching a final state.

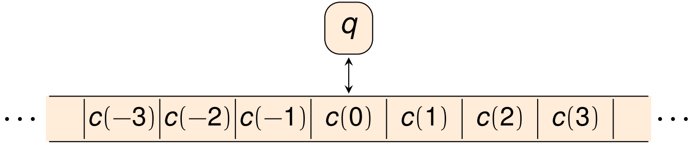
Turing Machine Configuration

A **configuration** (q, c) of a Turing machine consists of

- a state $q \in Q$, and
- a function $c : \mathbb{Z} \to \Gamma$, the **tape content.**

The non-blank positions $\{z \in \mathbb{Z} \mid c(z) \neq \square\}$ are finite.

The head of the machine stands on $c(0)$.



Let $n, m \in \mathbb{N}$ (exist for every configuration) such that

$$\forall i < -n.\ c(i) = \square \qquad \text{and} \qquad \forall i > m.\ c(i) = \square$$

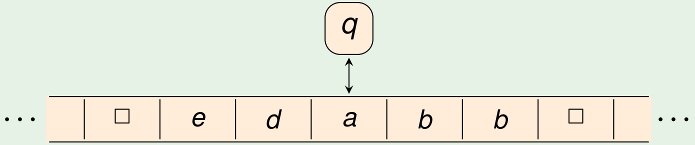
Then we **denote the configuration by the finite word**

$$c(-n)c(-n+1)\cdots c(-1)\ \color{red}{q}\ c(0)c(1)\cdots c(m)$$

Turing Machine Configuration

So configurations are denoted by words from $\Gamma^* \times Q \times \Gamma^*$.

For instance, the configuration



can be denoted by

$$ed\,q\,abb$$

The words

$$ed\,q\,abb\square \quad\approx\quad \square ed\,q\,abb \quad\approx\quad \square\square ed\,q\,abb\square \quad\cdots$$

denote the same configuration.

We write $w \approx v$ if w and v denote the same configuration.

Turing Machine Computations

The **computation steps** $\vdash$ on configurations are defined by:

$$v q a w \vdash v b q' w \qquad \text{if } \delta(q,a) = (q', b, R)$$
$$v c q a w \vdash v q' c b w \qquad \text{if } \delta(q,a) = (q', b, L)$$

where $v, w \in \Gamma^*$, $a, c \in \Gamma$ and $q \in Q$.

We write $\vdash^*$ for a computation of zero or more steps.

Assume that (δ is undefined in all other case)

$$\delta(q_0, a) = (q_0, a, R) \quad \delta(q_1, a) = (q_1, b, L) \quad \delta(q_0, \square) = (q_1, c, L)$$

Then we have steps:

$$q_0 aa \vdash a q_0 a \vdash aa q_0 \vdash a q_1 ac \vdash q_1 abc \vdash q_1 \square bbc$$

Here we use $aaq_0 \approx aaq_0\square$ and $q_1 abc \approx \square q_1 abc$.

A configuration $vqaw$ is a **halting state** if $\delta(q,a)$ is undefined.

Drawing Turing Machines

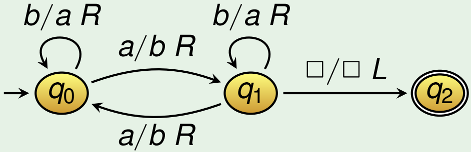
The transition graph for a TMs contains

an arrow $\quad q \xrightarrow{a/b\ X} q' \quad$ whenever $\quad \delta(q, a) = (q', b, X)$

The Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ with $\Sigma = \{ a, b \}$, $\Gamma = \{ a, b, \Box \}$, $Q = \{ q_0, q_1, q_2 \}$, $F = \{ q_2 \}$ and

$$\delta(q_0, a) = (q_1, b, R) \qquad \delta(q_1, a) = (q_0, b, R)$$
$$\delta(q_0, b) = (q_0, a, R) \qquad \delta(q_1, b) = (q_1, a, R)$$
$$\delta(q_1, \Box) = (q_2, \Box, L)$$

can be visualised as



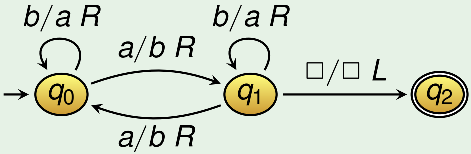
Turing Machines and Languages

The **language** $L(M)$ accepted by TM $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ is

$$\{\, w \in \Sigma^* \mid q_0 w \vdash^* uqv \text{ for some } q \in F,\ u, v \in \Gamma^* \,\}$$

If $w \notin L(M)$ this can have two causes:

- the execution halts in a configuration vqw with $q \notin F$, or
- the execution is infinite (never halts).



What is $L(M)$?

The set of words over $\Sigma = \{\, a, b \,\}$ with an odd number of a's.

A language is **recursively enumerable** if it is accepted by a TM.

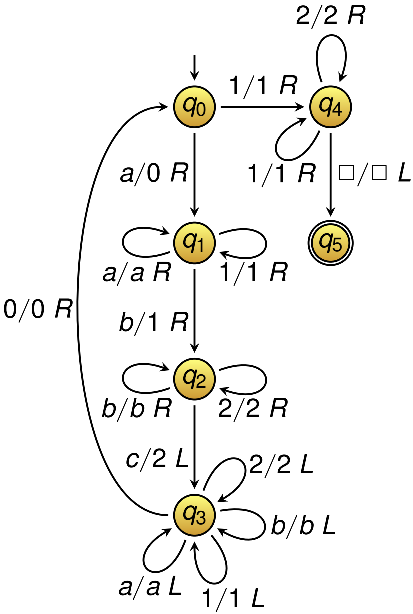
Example

We construct a TM M with $L(M) = \{ a^n b^n c^n \mid n \geq 1 \}$.

Idea: stepwise replace one a by 0, one b by 1 and one c by 2.

- $\Sigma = \{ a, b, c \}$ and $\Gamma = \{ a, b, c, 0, 1, 2, \Box \}$
- q_0: Read a, replace by 0, move right and switch to q_1.
- q_1: Keep moving right until we read b.
  Replace b by 1, move right and switch to q_2.
- q_2: Keep moving right until we read c.
  Replace c by 2, move left and switch to q_3.
- q_3: Keep moving left until we read 0.
  Move right and switch back to q_0.
- If we read 1 in q_0, switch to q_4.
- q_4: Keep moving right to check whether there are a's, b's or c's left. If not, then go to final state q_5.

Example

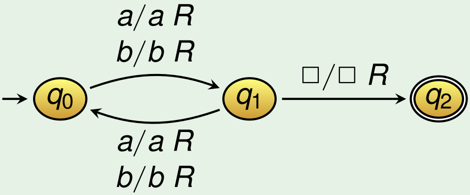


$$
\begin{aligned}
& q_0\, aabbcc \\
\vdash\; & 0 q_1\, abbcc \\
\vdash\; & 0 a q_1\, bbcc \\
\vdash\; & 0 a 1 q_2\, bcc \\
\vdash\; & 0 a 1 b q_2\, cc \\
\vdash\; & 0 a 1 q_3\, b 2 c \\
\vdash\; & 0 a q_3\, 1 b 2 c \\
\vdash\; & 0 q_3\, a 1 b 2 c \\
\vdash\; & q_3\, 0 a 1 b 2 c \\
\vdash\; & 0 q_0\, a 1 b 2 c \\
\vdash^*\; & 0 0 q_0\, 1 1 2 2 \\
\vdash\; & 0 0 1 q_4\, 1 2 2 \\
\vdash^*\; & 0 0 1 1 2 2 q_4 \\
\vdash\; & 0 0 1 1 2 q_5\, 2
\end{aligned}
$$

$$
\begin{aligned}
& q_0\, aabbbcc \\
\vdash^+\; & 0 q_0\, a 1 b b 2 c \\
\vdash^+\; & 0 0 q_0\, 1 1 b 2 2 \\
\vdash\; & 0 0 1 q_4\, 1 b 2 2 \\
\vdash\; & 0 0 1 1 q_4\, b 2 2
\end{aligned}
$$

Exercise

Construct a Turing machine accepting all words of **odd** length over the alphabet $\Sigma = \{a, b\}$.



Multiple labels on an arrow are short for multiple transitions.

Extensions of Turing Machines

Extensions of Turing Machines

Extensions of TMs such as

- multiple tapes, or
- nondeterminism

do **not** give extra expressive power.

Multiple tapes can be simulated using a single tape with polynomial overhead in time complexity.

Nondeterministic Turing machines have as transition function

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$$

A nondeterministic TM can be simulated by deterministic TM using **breadth-first search** (all computations in parallel).

The overhead in **time complexity** is believed to be an **exponential factor**.

Church-Turing Thesis

Church-Turing Thesis

Church-Turing thesis: Every computation of a computer can be simulated by a deterministic Turing machine.

This thesis has stood the test of time.

Also computations of **quantum computers** can be simulated by a Turing machines.

Quantum computers can do certain computations faster than classical computers, but they do not change the limits of computability.

Alonzo Church & Alan Turing

Two of the founders of the **theory of computability.**

Alonzo Church (1903-1995) is inventor of the **λ-calculus.**

Alan Turing (1912-1954)

- introduced the **Turing machine**,
- invented the **Turing test**,
- key role in cracking the German **Enigma machine.**

Both proved **undecidability of validity in predicate logic**.

Not all Languages are Recursively Enumerable

Not all Languages are Recursively Enumerable

A set A is countable if there is a surjective function $f : \mathbb{N} \to A$.

There are **countably** many TMs over an input alphabet Σ.

There are **uncountable** many languages over Σ.

Proof

Let $a \in \Sigma$.

Assume $L_0, L_1, L_2, \ldots$ is enumeration of all languages over $\{a\}$.

Define a language L as follows: for every $i \geq 0$.

$$a^i \in L \iff a^i \notin L_i$$

Then for every $i \geq 0$, we have $L \neq L_i$.

Thus L is **not** part of the above enumeration. Contradiction.

Conclusion: not all languages are recursively enumerable.

Universal Turing Machine

Universal Turing Machine

A computer can execute any program on any input.

A TM is called **universal** if it can simulate every TM.

A universal TM gets as input

- a Turing machine M (described as a word w)
- an input word u

and then executes (simulates) M on u.

The input w and u can be written on the tape as $w\#u$.

Theorem

There exists a universal Turing machine.