Complexity Theory

Lecture 2: Abstracting algorithms via Turing machines

Tom Gur

http://www.cl.cam.ac.uk/teaching/2324/Complexity

Formalising Algorithms

To prove a lower bound on the complexity of a problem, rather then a specific algorithm, we need to prove a statement about all algorithms for solving it.

Formalising Algorithms

To prove a lower bound on the complexity of a problem, rather then a specific algorithm, we need to prove a statement about all algorithms for solving it.

In order to prove facts about all algorithms, we need a mathematically precise definition of an algorithm.

Formalising Algorithms

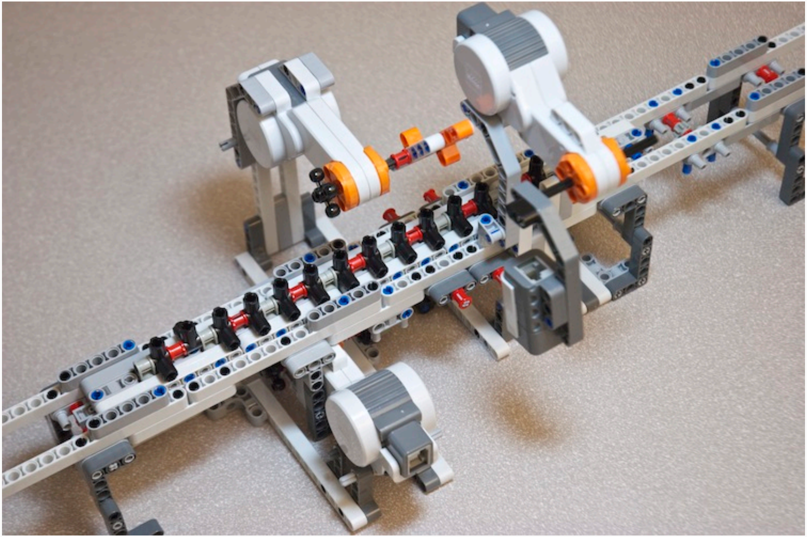
To prove a lower bound on the complexity of a problem, rather then a specific algorithm, we need to prove a statement about all algorithms for solving it.

In order to prove facts about all algorithms, we need a mathematically precise definition of an algorithm.

We will use the Turing machine.

> The simplicity of the Turing machine means it's not useful for actually expressing algorithms, but very well suited for proofs about all algorithms.

Turing Machines



For our purposes, a Turing Machine consists of:

- Q — a finite set of states;

For our purposes, a Turing Machine consists of:

- Q — a finite set of states;
- Σ — a finite set of symbols, including $\sqcup$ and $\triangleright$.

Turing Machines

For our purposes, a Turing Machine consists of:

- Q — a finite set of states;
- Σ — a finite set of symbols, including $\sqcup$ and $\triangleright$.
- $s \in Q$ — an initial state;

Turing Machines

For our purposes, a Turing Machine consists of:

- Q — a finite set of states;
- Σ — a finite set of symbols, including $\sqcup$ and $\triangleright$.
- $s \in Q$ — an initial state;
- $\delta : (Q \times \Sigma) \to (Q \cup \{acc, rej\}) \times \Sigma \times \{L, R, S\}$
  A transition function that specifies, for each state and symbol a next state (or accept acc or reject rej), a symbol to overwrite the current symbol, and a direction for the tape head to move (L – left, R – right, or S - stationary)

Configurations

A complete description of the configuration of a machine can be given if we know what state it is in, what are the contents of its tape, and what is the position of its head. This can be summed up in a simple triple:

Configurations

A complete description of the configuration of a machine can be given if we know what state it is in, what are the contents of its tape, and what is the position of its head. This can be summed up in a simple triple:

Definition
A configuration is a triple (q, w, u), where $q \in Q$ and $w, u \in \Sigma^\star$

Configurations

A complete description of the configuration of a machine can be given if we know what state it is in, what are the contents of its tape, and what is the position of its head. This can be summed up in a simple triple:

Definition
A configuration is a triple (q, w, u), where $q \in Q$ and $w, u \in \Sigma^\star$

The intuition is that (q, w, u) represents a machine in state q with the string wu on its tape, and the head pointing at the last symbol in w.

Configurations

A complete description of the configuration of a machine can be given if we know what state it is in, what are the contents of its tape, and what is the position of its head. This can be summed up in a simple triple:

Definition
A configuration is a triple (q, w, u), where $q \in Q$ and $w, u \in \Sigma^\star$

The intuition is that (q, w, u) represents a machine in state q with the string wu on its tape, and the head pointing at the last symbol in w.

The configuration of a machine completely determines the future behaviour of the machine.

Computations

Given a machine $M = (Q, \Sigma, s, \delta)$ we say that a configuration (q, w, u) yields in one step (q', w', u'), written

$$(q, w, u) \rightarrow_M (q', w', u')$$

Computations

Given a machine $M = (Q, \Sigma, s, \delta)$ we say that a configuration (q, w, u) yields in one step (q', w', u'), written

$$(q, w, u) \rightarrow_M (q', w', u')$$

if

- $w = va$;

Computations

Given a machine $M = (Q, \Sigma, s, \delta)$ we say that a configuration (q, w, u) yields in one step (q', w', u'), written

$$(q, w, u) \rightarrow_M (q', w', u')$$

if

- $w = va$;
- $\delta(q, a) = (q', b, D)$; and

Computations

Given a machine $M = (Q, \Sigma, s, \delta)$ we say that a configuration (q, w, u) yields in one step (q', w', u'), written

$$(q, w, u) \rightarrow_M (q', w', u')$$

if

- $w = va$;
- $\delta(q, a) = (q', b, D)$; and
- either $D = L$ and $w' = v$ and $u' = bu$
  or $D = S$ and $w' = vb$ and $u' = u$
  or $D = R$ and $w' = vbc$ and $u' = x$, where $u = cx$. If u is empty, then $w' = vb\sqcup$ and u' is empty.

Computations

The relation $\to_{\mathrm{M}}^{\star}$ is the reflexive and transitive closure of $\to_{\mathrm{M}}$.

Computations

The relation $\rightarrow_{M}^{\star}$ is the reflexive and transitive closure of $\rightarrow_{M}$.

A sequence of configurations $c_1, \ldots, c_n$, where for each i, $c_i \rightarrow_{M} c_{i+1}$, is called a computation of M.

Computations

The relation $\to_M^\star$ is the reflexive and transitive closure of $\to_M$.

A sequence of configurations $c_1, \ldots, c_n$, where for each i, $c_i \to_M c_{i+1}$, is called a computation of M.

The language $L(M) \subseteq \Sigma^\star$ accepted by the machine M is the set of strings

$$\{x \mid (s, \triangleright, x) \to_M^\star (acc, w, u) \text{ for some } w \text{ and } u\}$$

Computations

The relation $\rightarrow_M^\star$ is the reflexive and transitive closure of $\rightarrow_M$.

A sequence of configurations $c_1, \ldots, c_n$, where for each i, $c_i \rightarrow_M c_{i+1}$, is called a computation of M.

The language $L(M) \subseteq \Sigma^\star$ accepted by the machine M is the set of strings

$$\{x \mid (s, \triangleright, x) \rightarrow_M^\star (acc, w, u) \text{ for some } w \text{ and } u\}$$

A machine M is said to halt on input x if for some w and u, either $(s, \triangleright, x) \rightarrow_M^\star (acc, w, u)$ or $(s, \triangleright, x) \rightarrow_M^\star (rej, w, u)$

Example

Consider the machine with δ given by:

	$\triangleright$	0	1	$\sqcup$
s	$\mathrm{s}, \triangleright, \mathrm{R}$	$\mathrm{rej}, 0, \mathrm{S}$	$\mathrm{rej}, 1, \mathrm{S}$	$\mathrm{q}, \sqcup, \mathrm{R}$
q	$\mathrm{rej}, \triangleright, \mathrm{R}$	$\mathrm{q}, 1, \mathrm{R}$	$\mathrm{q}, 1, \mathrm{R}$	$\mathrm{q}', 0, \mathrm{R}$
q'	$\mathrm{rej}, \triangleright, \mathrm{R}$	$\mathrm{rej}, 0, \mathrm{S}$	$\mathrm{q}', 1, \mathrm{L}$	$\mathrm{acc}, \sqcup, \mathrm{S}$

Example

Consider the machine with δ given by:

	$\triangleright$	0	1	$\sqcup$
s	$s, \triangleright, R$	$rej, 0, S$	$rej, 1, S$	$q, \sqcup, R$
q	$rej, \triangleright, R$	$q, 1, R$	$q, 1, R$	$q', 0, R$
q'	$rej, \triangleright, R$	$rej, 0, S$	$q', 1, L$	$acc, \sqcup, S$

This machine, when started in configuration $(s, \triangleright, \sqcup 1^n 0)$ eventually halts in configuration $(acc, \triangleright \sqcup 1^{n+1} 0 \sqcup, \varepsilon)$.

Why Turing Machines?

The Church-Turing thesis states that a Boolean function can be computed if and only if it is computable by a Turing machine.

Why Turing Machines?

The Church-Turing thesis states that a Boolean function can be computed if and only if it is computable by a Turing machine.

The Extended Church-Turing thesis adds that this also captures efficient computation.

Why Turing Machines?

The Church-Turing thesis states that a Boolean function can be computed if and only if it is computable by a Turing machine.

The Extended Church-Turing thesis adds that this also captures efficient computation.

Hence, the model does not matter. We can use whichever is most convenient.

Why Turing Machines?

The Church-Turing thesis states that a Boolean function can be computed if and only if it is computable by a Turing machine.

The Extended Church-Turing thesis adds that this also captures efficient computation.

Hence, the model does not matter. We can use whichever is most convenient.

To date, the only widely accepted contender to the Extended Church-Turing thesis is Quantum Computing.

The formalisation of Turing machines extends in a natural way to multi-tape machines. For instance a machine with k tapes is specified by:

- Q, Σ, s; and

Example: Multi-Tape Machines

The formalisation of Turing machines extends in a natural way to multi-tape machines. For instance a machine with k tapes is specified by:

- Q, Σ, s; and
- $\delta : (Q \times \Sigma^k) \to (Q \cup \{acc, rej\}) \times (\Sigma \times \{L, R, S\})^k$

Example: Multi-Tape Machines

The formalisation of Turing machines extends in a natural way to multi-tape machines. For instance a machine with k tapes is specified by:

- Q, Σ, s; and
- $\delta : (Q \times \Sigma^k) \to (Q \cup \{acc, rej\}) \times (\Sigma \times \{L, R, S\})^k$

Example: Multi-Tape Machines

The formalisation of Turing machines extends in a natural way to multi-tape machines. For instance a machine with k tapes is specified by:

- Q, Σ, s; and
- $\delta : (Q \times \Sigma^k) \to (Q \cup \{acc, rej\}) \times (\Sigma \times \{L, R, S\})^k$

Similarly, a configuration is of the form:

$$(q, w_1, u_1, \ldots, w_k, u_k)$$

A language $L \subseteq \Sigma^\star$ is recursively enumerable if it is $L(M)$ for some M.

Decidability

A language $L \subseteq \Sigma^\star$ is recursively enumerable if it is $L(M)$ for some M.

A language L is decidable if it is $L(M)$ for some machine M which halts on every input.

Decidability

A language $L \subseteq \Sigma^\star$ is recursively enumerable if it is $L(M)$ for some M.

A language L is decidable if it is $L(M)$ for some machine M which halts on every input.

A language L is semi-decidable if it is recursively enumerable.

Decidability

A language $L \subseteq \Sigma^\star$ is recursively enumerable if it is $L(M)$ for some M.

A language L is decidable if it is $L(M)$ for some machine M which halts on every input.

A language L is semi-decidable if it is recursively enumerable.

A function $f : \Sigma^\star \to \Sigma^\star$ is computable, if there is a machine M, such that for all x, $(s, \triangleright, x) \to_M^\star (acc, \triangleright f(x), \varepsilon)$

Running Time

With any Turing machine M, we associate a function $r : \mathbb{N} \to \mathbb{N}$ called the running time of M.

Running Time

With any Turing machine M, we associate a function $r : \mathbb{N} \to \mathbb{N}$ called the running time of M.

$r(n)$ is defined to be the largest value R such that there is a string x of length n so that the computation of M starting with configuration $(s, \triangleright, x)$ is of length R (i.e. has R successive configurations in it) and ends with an accepting configuration.

Running Time

With any Turing machine M, we associate a function $r : \mathbb{N} \to \mathbb{N}$ called the running time of M.

$r(n)$ is defined to be the largest value R such that there is a string x of length n so that the computation of M starting with configuration $(s, \triangleright, x)$ is of length R (i.e. has R successive configurations in it) and ends with an accepting configuration.

In short, $r(n)$ is the length of the longest accepting computation of M on an input of length n.

Running Time

With any Turing machine M, we associate a function $r : \mathbb{N} \to \mathbb{N}$ called the running time of M.

$r(n)$ is defined to be the largest value R such that there is a string x of length n so that the computation of M starting with configuration $(s, \triangleright, x)$ is of length R (i.e. has R successive configurations in it) and ends with an accepting configuration.

In short, $r(n)$ is the length of the longest accepting computation of M on an input of length n.

We let $r(n) = 0$ if M does not accept any inputs of length n.

For any function $f : \mathbb{N} \to \mathbb{N}$, we say that a language L is in $\mathsf{TIME}(f)$ if there is a machine $M = (Q, \Sigma, s, \delta)$, such that:

- $L = L(M)$; and

For any function $f : \mathbb{N} \to \mathbb{N}$, we say that a language L is in $\mathsf{TIME}(f)$ if there is a machine $M = (Q, \Sigma, s, \delta)$, such that:

- $L = L(M)$; and
- The running time of M is $O(f)$.

Complexity

For any function $f : \mathbb{N} \to \mathbb{N}$, we say that a language L is in $\mathsf{TIME}(f)$ if there is a machine $M = (Q, \Sigma, s, \delta)$, such that:

- $L = L(M)$; and
- The running time of M is $O(f)$.

For any function $f : \mathbb{N} \to \mathbb{N}$, we say that a language L is in $\mathsf{TIME}(f)$ if there is a machine $M = (Q, \Sigma, s, \delta)$, such that:

- $L = L(M)$; and
- The running time of M is $O(f)$.

Similarly, we define $\mathsf{SPACE}(f)$ to be the languages accepted by a machine which uses $O(f(n))$ tape cells on inputs of length n.

For any function $f : \mathbb{N} \to \mathbb{N}$, we say that a language L is in $\mathsf{TIME}(f)$ if there is a machine $M = (Q, \Sigma, s, \delta)$, such that:

- $L = L(M)$; and
- The running time of M is $O(f)$.

Similarly, we define $\mathsf{SPACE}(f)$ to be the languages accepted by a machine which uses $O(f(n))$ tape cells on inputs of length n.

In defining space complexity, we assume a machine M, which has a read-only input tape, and a separate work tape. We only count cells on the work tape towards the complexity.

Questions?