Computational Complexity
计算复杂性

张方国

中山大学计算机学院

isszhfg@mail.sysu.edu.cn

Lecture 2. Computational Models

- Uniform Models
  - A concrete model: Turing machines
    Non-deterministic Turing machines, Multi-tape Turing machines, Church-Turing thesis
  - Reasonable model of computation:
    Uncomputable functions and Universal computation
- Non-uniform Models
  - Circuits
  - Machines That Take Advice

A concrete model : Turing machines

A computation is a process that modifies an environment via repeated applications of a predetermined rule.

A standard model which is convenient for most purposes in complexity theory is the Turing Machine.

The model of Turing machines was invented before modern computers were even built, and was meant to provide a concrete model of computation and a definition of computable functions.

A Turing machine is a theoretical device that manipulates symbols on a strip of tape according to a table of rules.

Turing machine is introduced in 1936 by Alan Turing(1912-1954) in his groundbreaking paper(A. Turing. On computable numbers, with an application to the Entscheidungs problem. Proceedings of the London Mathematical Society. ser.2, 42:230-265, 1936. Correction, ibid, vol.43, pp.544-546, 1937)

Turing Machines

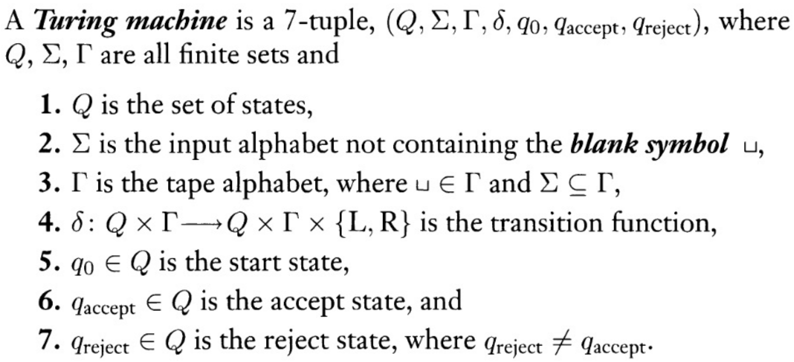
Turing机是由一个控制器和一条无穷长的带子组成。

带子被分成许多小格子，两头是无限的。控制器具有有限个内部状态和一个读写头，在计算的每一步，控制器处于某个内部状态，读写头扫描带的某一个方格，并根据它当前的状态和扫描的方格内的内容决定下一步的动作。

Turing机是最一般的计算模型，计算机能做什么，它就能做什么。

Formal definition of Turing machines

From M. Sipser's book:



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

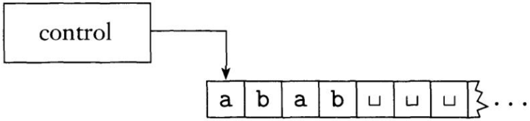
1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta \colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ is the transition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

A high-level description of the model of Turing machines

- The main component in the environment of TM is an infinite sequence of cells(called the tape), each capable of holding a single symbol(i.e., member of a finite set $\Sigma \supset \{0, 1\}$).
  This sequence is envisioned as starting at a leftmost cell, and extending infinitely to the right.
  The current location;
  The internal state(a member of a finite set Q).
  The tape's contents combined with the machine's location and its internal state is called the **instantaneous configuration**(格局) of the machine.

我们称一个单带DTM为单方向无限带DTM，如果对任何输入x，该DTM的探头从不移动到x左边的方格，亦即，我们可以假设该图灵机的记忆带只能单方向向右无限延伸，而在机器开始运转前输入x储存在最左边$|x|$格。

实际上不难证明，单方向无限带DTM模型与双方向单带DTM模型等价。



- The transition function, which is defined over the set of all possible symbol state pairs. Specifically, the transition function is a mapping from $\Sigma \times Q$ to $\Sigma \times Q \times \{-1, 0, +1\}$ (**L**eft, **S**tay, **R**ight).

$$or \quad Q \times \Sigma \to Q \times \Sigma \times \{L, S, R\}$$

In addition, the machine's description specifies an **initial state** and a **halting state**, and the computation of the machine halts when the machine enters its halting state.

当控制器读入带子上当前的符号x时，如果控制器的状态是s，且部分函数有定义为f(s，x)=(s’，x’，d)，则控制器的动作是:

1. 进入状态s’。
2. 擦掉当前格子里的符号x，写上符号x’。
3. 如果d=R，则右移一个格子，如果d=L，则左移一个格子。

我们把这个步骤写成(s，x，s’，x’，d)。如果部分函数f对(s，x)没有定义，则Turing机不动。一种常用的定义Turing机的方法就是指定这个五元组的集合，用这种方法定义Turing机，状态集合和字母表也就能得到了。

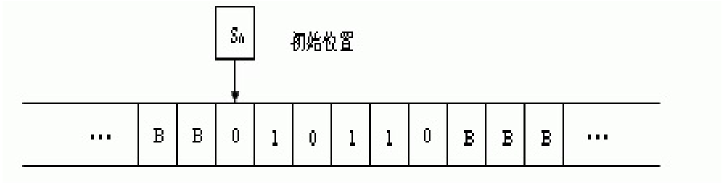
- A single computation step of such a Turing machine depends on its current location on the tape, on the contents of the corresponding cell and on the internal state of the machine. Based on the latter two elements, the transition function determines a new symbol state pair as well as a movement instruction.

Differences between finite automata and Turing machine

- A Turing machine can both **write** on the tape and **read** from it.
- The read-write head can move both to the **left** and to the **right**.
- The tape is **infinite**.
- The special states for **rejecting** and **accepting** take immediate effect.

Examples of Turing machines

例1：“Turing机T用下面七个五元组来定义，(s0，0，s0，0，R)，(s0，1，s1，1，R)，(s0，B，s3，B，R)，(s1，0，s0，0，R)，(s1，1，s2，0，L)，(s1，B，s3，B，R)，(s2，1，s3，0，R)，问T停止的时候，最后的带子是什么样的？”

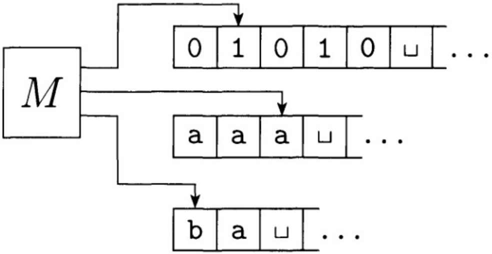


Examples of Turing machines

例2：“程序{(q1,1,q2,0,R), (q2,1,q3,1,R), (q3,1,q3,1,R), (q3,B,q3,1,R), (q2,B,q0,0,R)}算一个函数f使得f在x定义当且仅当x=0并且输出0。”

Multi-tape Turing machines

The transition function of MTM is a mapping from $\Sigma^k \times Q$ to $\Sigma^k \times Q \times \{-1, 0, +1\}$



A function can be computed by a single-tape Turing machine if and only if it is computable by a multi-tape Turing machine

定理：对任何多带图灵机M，存在单带图灵机M_1，与M计算同一函数。

证明： 设M有k条带并且使用带用符号集合Γ和空格符号B。我们让M_1具有带用符号集合$\Gamma_1 = \Gamma \cup \Gamma_2$，其中$\Gamma_1 = (\Gamma \times \{X, B\})^k$并且$X$不在$\Gamma$里。$M_1$使用空格符号$(B, B, ..., B)$($2k$个$B$)。

最初，M_1带上的输入与M的输入一样，除空格符号外，用的符号都属于Γ。M_1做的第一件事就是从左到右扫描一遍，然后回到原处并在此过程中将所有Γ中符号换为Γ_2中元素。

之后，M_1模拟M的每次移动。在开始时让M_1的读写头置于最左边含X的方格处，此时M_1的控制状态是个$k+1$维向量(第一个分量是M的控制状态，其余k个分量记载M的k条带上探头读到的符号。

最后当M停机时，M_1将非空格做些处理后也停止工作。

Non-deterministic Turing machines

A non-deterministic Turing machine is defined as TM except that the transition function maps symbol-state pairs to subsets of triples (rather than to a single triple) in $\Sigma \times Q \times \{-1, 0, +1\}$.

The transition function of NTM is a mapping:

$$\Sigma \times Q \to 2^{\Sigma \times Q \times \{-1,0,+1\}}.$$

转移函数是单值还是多值，称图灵机为确定型或非确定型。

Accordingly, the configuration following a specific instantaneous configuration may be one of several possibilities, each determine by a different possible triple. Thus, the computations of a non-deterministic machine on a fixed input may result in different outputs.

非确定型图灵机M在输入串ω上的计算过程可以表示为一棵树，不同的分支对应着每一步计算的不同的可能性。只要有任意一个分支进入接受状态，则称M接受ω；只要有任意一个分支进入拒绝状态，则称M拒绝ω；某些分支可能永远无法停机，但只要有一个分支可以进入接受或拒绝状态，我们就说M在输入ω上可停机。

确定型图灵机 DTM 的机器状态只依赖于时间和输入，是完全确定的。每一步都由前一步所达到的状态惟一确定。如果机器最后进入一个接受状态，就认为机器接受了输入。这种计算称为确定型的，计算的过程可以用一条链来表示，如图 3 所示。确定型是最简单的一种计算类型。

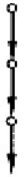
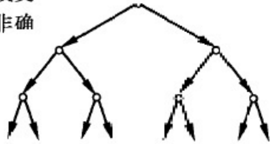


图 3　确定型计算

非确定型图灵机 NDTM 在某一时刻可以有若干种选择，进入若干不同状态，而在新的情况下又有若干不同选择。这时计算过程可用一棵树表示，如图 4 所示。若规定只要有一条路从树根通向一个接受的顶点，就认为机器接受了输入字符串，这种接受方式就叫作非确定型的。此时，树高被看作是非确定计算所需的时间。

图 4　非确定型计算



非确定型算法:

第一步：猜测一个变量的真值赋值;

第二步：检查该赋值是否满足

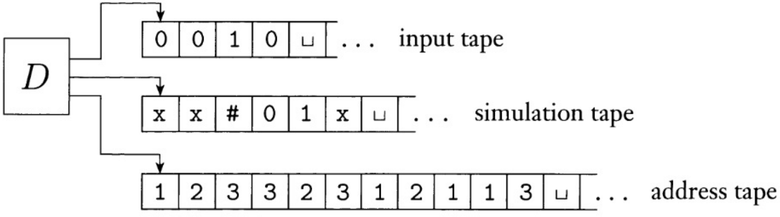
非确定型算法的计算时间：各种可能的计算过程的最短时间

定理：如果问题被非确定型图灵机M在多项式时间内解决，则一定存在多项式P使得该问题被时间复杂度为$O(2^{P(n)})$的确定型图灵机程序所接受。

非确定型图灵机和确定型图灵机的不同之处在于，在计算的每一时刻，根据当前状态和读写头所读的符号，机器存在多种状态转移方案，机器将任意地选择其中一种方案继续运作，直到最后停机为止。

Theorem: Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Proof:

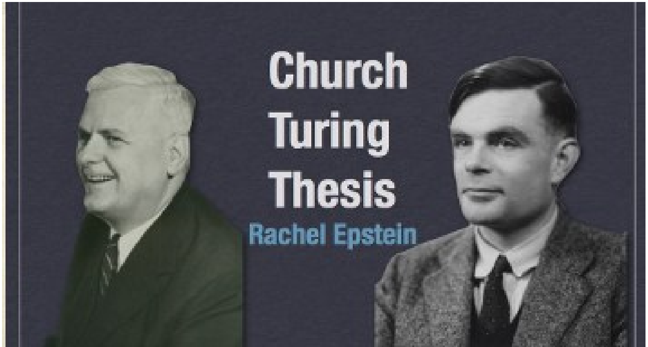

D
0 0 1 0 ␣ ...
input tape
x x # 0 1 x ␣ ...
simulation tape
1 2 3 3 2 3 1 2 1 1 3 ␣ ...
address tape

“It is a common misconception that quantum computers are NTMs. It is believed but has not been proven that the power of quantum computers is incomparable to that of NTMs. That is, problems likely exist that an NTM could efficiently solve but that a quantum computer cannot. A likely example of problems solvable by NTMs but not by quantum computers in polynomial time are NP-complete problems”

量子计算机模型是介于确定型图灵机和非确定型图灵模型之间的一种计算模型。

The Church-Turing Thesis

Alonzo Church(1903.6.14—1995.8.11)是美国数学家，1936年发表可计算函数的第一份精确定义，对算法理论的系统发展做出巨大贡献。邱奇在普林斯顿大学受教并工作四十年，曾任数学与哲学教授。是图灵的博士生导师。1967年迁往UCLA。



A function can be computed by some Turing machine if and only if it can be computed by some machine of any other “reasonable and general” model of computation.

什么是合理的计算模型？ 如下是三个起码的条件：

- 计算一个函数只要有限条指令；
- 每条指令可由模型中的有限个计算步骤完成；
- 执行指令的过程是确定型的。

Church-Turing论题是可计算性理论中的一基本论题，它断言图灵机可计算函数类就是直观可计算的函数类。因为直观可计算函数并不是精确的数学概念，所以Church-Turing论题不能用数学方法加以证明。但是有许多令人信服的论据支持这个论题，人们后来提出许多不同的计算模型都被证明与图灵机等价，即各种模型所定义的可计算函数类都是图灵机可计算函数类。这表明图灵机及其他等价模型确实合理地定义了可计算性。因此，Church-Turing论题得到了计算机科学界和数学界的公认。

Uncomputable functions

Only relatively few functions are computable.

In fact,

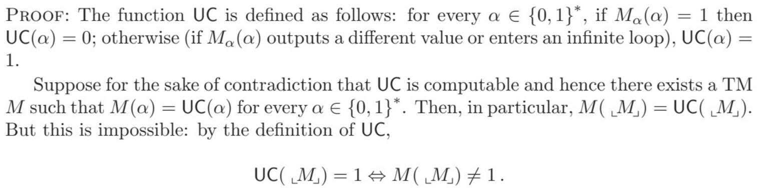
Theorem

The set of computable functions is countable, whereas the set of all functions(from strings to string) has cardinality $\aleph$.

Existence of uncomputable functions

Theorem

There exists a function UC : $\{0,1\}^* \to \{0,1\}$ *that is not computable by any TM.*



PROOF: The function UC is defined as follows: for every $\alpha \in \{0,1\}^*$, if $M_\alpha(\alpha) = 1$ then $\mathsf{UC}(\alpha) = 0$; otherwise (if $M_\alpha(\alpha)$ outputs a different value or enters an infinite loop), $\mathsf{UC}(\alpha) = 1$.

Suppose for the sake of contradiction that UC is computable and hence there exists a TM M such that $M(\alpha) = \mathsf{UC}(\alpha)$ for every $\alpha \in \{0,1\}^*$. Then, in particular, $M(\llcorner M \lrcorner) = \mathsf{UC}(\llcorner M \lrcorner)$. But this is impossible: by the definition of UC,

$$\mathsf{UC}(\llcorner M \lrcorner) = 1 \Leftrightarrow M(\llcorner M \lrcorner) \neq 1 \,.$$

An example of uncomputable functions

The Halting Function:

$h : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ is defined such that $h(\langle M \rangle, x) \overset{\text{def}}{=} 1$ if and only if M halts on input x.

(We rely on the postulate that each machine in the model has finite description, and denote the description of machine M by $\langle M \rangle \in \{0,1\}^*$.)

Theorem (undecidability of the halting problem:)

The halting function is not computable.

图灵机停机问题: 能否给出一个判断任意一个图灵机是否停机的一般方法? 答案是NO.

这个问题实际上是问: 是否存在一台"万能的"图灵机H, 把任意一台图灵机M 输入给H, 它都能判定M 最终是否停机, 输出一个明确的“yes” 或“no” 的答案?

可以利用反证法来证明这样的H 不可能存在. 假定存在一个能够判定任意一台图灵机是否停机的万能图灵机H(M), 如果M 最终停机, H 输出“halt”; 如果M 不停机, H 输出“loop”. 我们把H 当作子程序, 构造如下程序P:

```
function P(M) {
if (H(M)==“loop”) return “halt”;
else if (H(M)==“halt”) while(true); // loop forever
}
```

因为P 本身也是一台图灵机, 可以表示为一个字符串, 所以我们可以把P 输入给它自己, 然后问P(P) 是否停机. 按照程序P 的流程, 如果P 不停机无限循环, 那么它就停机, 输出"halt"; 如果P 停机, 那么它就无限循环, 不停机; 这样无论如何我们都将得到一个矛盾, 所以假设前提不成立, 即不存在这样的H. 或者说, 图灵机停机问题是不可判定的(undecidable).

Turing-reduction: an algorithm that solves one problem by using as a subroutine an algorithm that solves another problem.

The undecidability of the halting problem (or rather the fact that the function d is uncomputable) is a special case of a more general phenomenon. Every non trivial decision problem regarding the function computed by a given Turing machine has no algorithmic solution:

Theorem (Rice's Theorem)

Let $\mathcal{F}$ be a non-trivial subset of the set of all computable partial functions, and let $\mathcal{S}_{\mathcal{F}}$ be the set of strings that describe machines that compute functions in $\mathcal{F}$. Then deciding membership in $\mathcal{S}_{\mathcal{F}}$ cannot be solved by an algorithm.

Universal computation

The problem with Turing Machines is that a different one must be constructed for every new computation to be performed, for every input output relation.

This is why we will introduce the notion of a universal Turing machine (UTM), which along with the input on the tape, takes in the description of a machine M. The UTM can go on then to simulate M on the rest of the contents of the input tape. A universal Turing machine can thus simulate any other machine.

Definition (universal machines)

A universal Turing machine is a Turing machine that on input a description of a machine M and an input x returns the value of $M(x)$ if M halts on x and otherwise does not halt.

That is, a universal Turing machine computes the partial function u that is defined over pairs $(< M >, x)$ such that M halts on input x in which case it holds that $u(< M >, x) = M(x)$. We note that if M halts on all possible inputs then $u(< M >, x)$ is defined for every x.

Theorem

There exists a universal Turing machine.

Oracle Machines

An oracle machine is a machine that is augmented such that it may pose questions to the outside.

An oracle machine is a Turing machine connected to an oracle.

using an oracle

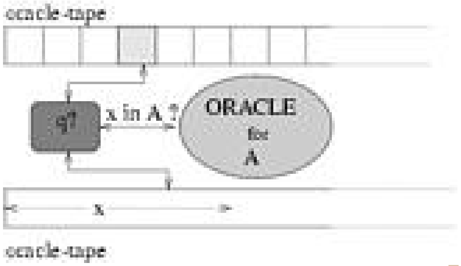
a. An oracle machine is a Turing machine with an additional tape, called the oracle tape, and two special states, called oracle invocation（神喻询问）and oracle spoke（神喻解答）.

b. The computation of the oracle machine M on input x and access to the oracle $f : \{0,1\}^* \to \{0,1\}^*$ is defined based on the successive configuration function. For configurations with state different from oracle invocation the next configuration is defined as usual（普通状态）. Let γ be a configuration in which the machine's state is oracle invocation and suppose that the actual contents of the oracle tape is q(i.e., q is the contents of the maximal prefix of the tape that holds bit values). Then, the configuration following γ is identical to γ, except that the state is oracle spoke, and the actual contents of the oracle tape is $f(q)$. The string q is called M's query and $f(q)$ is called the oracle's reply.

c. The output of M on input x when given oracle access to f is denote $M^f(x)$.

oracle machine M在没有进入特殊状态之前与通常的TM一样的计算。M可以从某些普通状态进入询问状态，但不能从任何普通状态进入解答状态。

神喻所执行的计算在运算时间的估计上只记为一个时间单位。



The oracle f can be imagined as some kind of “black box”: it answers the query of whether it contains q or not within one step of M’s computation.

如果存在一个带函数神喻TM的M使得M^f计算函数g，那么称g图灵归约于函数f。

所谓神喻实际上就是子程序调用。

Applications to cryptography

In cryptography, oracles are used to make arguments for the security of cryptographic protocols where a hash function is used. A security reduction for the protocol is given in the case where, instead of a hash function, a random oracle answers each query randomly but consistently; the oracle is assumed to be available to all parties including the attacker, as the hash function is. Such a proof shows that unless the attacker solves the hard problem at the heart of the security reduction, they must make use of some interesting property of the hash function to break the protocol; they cannot treat the hash function as a black box (i.e., as a random oracle).

Thank You Very Much!