Chapter 1

课程简介与基础知识

1.1 课程简介

作为计算机科学的核心, 计算理论是关于计算知识的有系统的整体, 本是数学的一个研究领域, 诞生于数理逻辑学家对计算本质的探索. 这里的计算 (*Computation*) 并不是指纯粹的算术 (*Calculation*), 而是指一种以 "机械而有效的" 方式获取问题答案的过程.

随着计算理论的发展和相关技术的进步, 最终促使了计算机的发明. 此后, 计算理论的重心也逐渐从数学转移到了计算机科学. 计算理论, 乃至计算机科学, 所关心的核心问题是:

计算机的基本能力和限制究竟是什么?

这个问题中包含了两个内容, 分别对应计算理论的两个研究方向: 可计算性理论和计算复杂性理论, 而形式语言与自动机理论正是这两个重要的研究方向的理论基础.

(1) 可计算性理论的核心问题是: 究竟哪些问题, 可以通过计算解决?

计算作为一种能力, 是否有边界? 是不是任何问题都可以通过计算来解决? 如果是, 它会是什么样的? 如果不是, 有哪些问题可以, 而哪些问题不可以? 为什么不可以?

为了能够严谨的研究这种机械而有效的计算过程, 我们需要严格定义的概念去描述它, 需要严谨的计算模型去分析它. 这些模型呢, 就是我们将要学习的自动机理论; 而这个概念, 其实就是已经被我们大家所熟知的 — 算法 (*Algorithm*).

所谓算法, 可以追溯到公元前 3 世纪, 那时的人们, 就已经有了算法的直觉概念, 并且试图寻找解决各种数学问题的算法, 其中最著名的就是求两整数最大公约数的欧几里得算法, 也称为辗转相除法. 如果一个问题, 有了具体的算法以后, 解决起来就不再需要太多的人的智慧, 只要按照算法的步骤机械的计算, 就可以得到答案. 比如, 寻找两个整数的最大公约数, 使用欧几里得算法, 一个小学生和一个数学家可以做的一样好.

数学家们也一直在寻找更为通用的算法, 试图解决更基本的问题. 20 世纪初, 数学家希尔

伯特曾经很乐观的试图寻找“整数上一阶谓词演算中判断任何公式是否为真”的算法. 如果这样的算法存在, 那么, 数学家就可以一劳永逸地解决几乎所有的数学命题. 但是, 由于当时对计算和算法本质的认识还不够深刻, 还不能使用数学工具去分析计算的能力. 所以这样的企图失败了, 但在这个过程当中, 积累了有关算法的重要知识.

到 20 世纪 30 年代, 数理逻辑学家在研究可计算的整数函数时, 通过图灵机和 λ 演算等计算模型, 首次将算法的概念形式化. 从那以后, 人们才可以利用数理逻辑方法研究计算的本质, 并且发现计算也不是万能的, 确实存在一些问题是不可能通过计算解决的, 或者说, 这些问题是不存在算法的. 我们在课程的最后部分中, 会给出这样的问题和相关的证明.

(2) 计算复杂性理论的核心问题是: *解决可计算的问题, 究竟需要多少资源?*

也就是计算一个问题时, 需要消耗的时间和占用的存储空间, 会达到什么程度? 如果一个问题, 无论使用什么算法, 求解过程都需要相当多的资源, 那么其中的原因是什么? 究竟是什么, 造成了一些问题很难计算, 而另一些问题却很容易?

虽然, 其中的原因仍然是未知的, 但在分析各种有效计算模型的过程中, 人们发现了一个按照难度给问题分类的完美体系. 如同元素周期表对化学元素性质的分类. 有了这个体系, 我们就可以将未知的问题分类, 根据难易程度选择用什么样的对策去解决. 目前, 计算复杂性理论仍然是计算机科学领域重要的研究热点, 但超出了形式语言与自动机课程的内容, 我们不会涉及太多, 但是这些内容却是通往复杂性理论研究的必经之路.

因为, 为了可计算性和计算复杂性的研究, 需要使用和构造什么样的计算模型, 就是形式语言与自动机理论的主要内容. 这些模型都是高度抽象化的计算装置, 简单明确但功能强大, 不但便于在理论分析中进行推导和证明, 在很多实际问题中也有非常直接的应用.

计算理论在一系列不同的领域中都有先驱者. 研究神经细胞网络模型的生物学家, 发展开关理论用于硬件设计的机电工程师, 从事数理逻辑研究的数学家, 研究自然语言文法的语言学家等. 这些研究中逐渐发展起来的一些模型, 对于理论计算机科学的研究是至关重要的.

什么是自动机理论?

自动机理论: 研究抽象机器及其所能解决问题的理论.

- 图灵机
- 有限状态机
- 文法, 下推自动机

其中最重要的就是图灵机, 它具有现如今各种计算机的所有能力, 是计算机的理论模型, 它区分了哪些问题是可计算的哪些是不可计算的. 而其他一些稍简单的模型, 比如有穷自动机, 在数字电路, 通讯协议和游戏 AI 的设计等实际问题上有重要的应用; 下推自动机和文法在计算机程序设计语言的设计和编译器实现上发挥了重要的作用.

什么是形式语言?

形式语言: 经数学定义的语言.

		自然语言		形式语言	
		English	中文	化学分子式	C 语言
语言	字符	A,a,B,b,...	天, 地,...	A-Z,a-z,0-9...	`A-Z,a-z,0-9...`
	单词	apple	苹果	H_2O	`char`
	句子	How're you?	早上好!	$2H_2+O_2=2H_2O$	`char a=10;`
	语法	Grammar	语法规则	精确定义的规则	

如果自动机是研究计算的模型, 那么语言就是研究计算的问题或实例. 而所谓*形式语言*, 就是指经数学定义的语言. 我们要以数学的方法从解决问题的角度研究计算, 那么我们首先需要以数学的方法来描述问题, 这种描述就是形式语言. 使用语言这个概念, 似乎有些奇怪, 但和我们的常识其实是一致的. 我们可以以这样的观点, 理解语言的构成, 由字符, 单词, 句子, 和语法构成了语言. 比如自然语言中的英文, 中文等. 只要有严谨的数学定义, 就可以称为形式语言, 比如化学分子式, 程序设计语言等. 定义一个语言, 我们首先要确定基本字符有哪些, 再确定如何构成单词和句子的基本规则, 单词和句子在形式语言中我们都认为是字符串, 最关键的是如何描述这个基本规则, 在形式语言与自动机理论中, 这种描述实际上就是自动机. 所以形式语言与自动机是密不可分的, 一方面自动机以语言为处理对象, 另一方面语言是以自动机为形式定义的.

计算科学的核心是使用模型研究算法. 本课程的大部分内容, 本质上也是各种算法与计算模型, 它们都能够处理一维的序列数据, 比如线性表、文本、图或树的遍历等等. 事实上, 任何程序所处理的, 包括输入和输出, 最终都可以看作是有限长的序列 — 最终都可以看作字符串. 那么, 对所谓“计算”的论证, 显然离不开对“字符串”的论证.

教材

- *Introduction to Automata Theory, Languages, and Computation* – 3rd Ed.
  - 作者 John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman
  - `http://infolab.stanford.edu/~ullman/ialc.html`
  - 影印版《自动机理论、语言和计算导论》机械工业出版社

参考书

- *Models of Computation*, by Jeff Erickson — `http://algorithms.wtf`
- *Introduction to the Theory of Computation*, 3ed, by Michael Sipser

1.2 基础知识

1.2.1 基本概念

1. 字母表: 符号 (或字符) 的非空有穷集.

$$\Sigma_1 = \{0,1\},$$
$$\Sigma_2 = \{a,b,\ldots,z\},$$
$$\Sigma_3 = \{x \mid x \text{ 是一个汉字}\}.$$

这里的“符号”是一个抽象的实体, 我们不再去形式的定义它, 如同几何学中对“点”和“线”的概念不加定义一样. 字母和数字是经常使用的一些符号的例子.

2. 字符串: 由某字母表中符号组成的有穷序列.

若 $\Sigma_1 = \{0,1\}$, 那么 $0,1,00,111001$ 为 Σ_1 上的字符串;

若 $\Sigma_2 = \{a,b,\ldots,z\}$, 那么 $ab, xkcd$ 为 Σ_2 上的字符串.

3. 空串: 记为 ε, 有 0 个字符的串.

字母表 Σ 可以是任意的, 但都有 $\varepsilon \notin \Sigma$.

4. 字符串的长度: 字符串中符号所占位置的个数, 记为 $|\text{␣}|$.

若字母表为 Σ, 可递归定义为:

$$|w| = \begin{cases} 0 & w = \varepsilon \\ |x|+1 & w = xa \end{cases},$$

其中 $a \in \Sigma$, w 和 x 是 Σ 中字符组成的字符串.

⋆. 符号使用的一般约定:

- 字母表: $\Sigma, \Gamma, \Delta, \ldots$
- 字符: $a, b, c, \ldots$
- 字符串: $\ldots, w, x, y, z$
- 集合: $A, B, C, \ldots$

5. 字符串 x 和 y 的连接: 将首尾相接得到新串的运算, 记为 $x\cdot y$ 或 xy.

   同样, 可递归定义为

$$x\cdot y=\begin{cases} x & y=\varepsilon \\ (x\cdot z)a & y=za \end{cases},$$

   其中 $a\in\Sigma$, 且 x,y,z 都是字符串.

   对任何字符串 x, 有 $\varepsilon\cdot x=x\cdot\varepsilon=x$.

   连接运算的符号 "$\cdot$" 一般省略.

6. 字符串 x 的 n 次幂($n\geq 0$), 递归定义为

$$x^n=\begin{cases} \varepsilon & n=0 \\ x^{n-1}x & n>0 \end{cases}.$$

   例如,

$$\begin{aligned} (ba)^2 &= (ba)^1 ba \\ &= (ba)^0 baba \\ &= \varepsilon baba \\ &= baba \end{aligned} \qquad \begin{aligned} ba^2 &= ba^1 a \\ &= ba^0 aa \\ &= b\varepsilon aa \\ &= baa \end{aligned}$$

7. 集合 A 和 B 的连接, 记为 $A\cdot B$ 或 AB, 定义为

$$A\cdot B=\{w\mid w=x\cdot y,\ x\in A\text{且}y\in B\}.$$

8. 集合 A 的 n 次幂($n\geq 0$), 递归定义为

$$A^n=\begin{cases} \{\varepsilon\} & n=0 \\ A^{n-1}A & n\geq 1 \end{cases}.$$

   那么, 若 Σ 为字母表, 则 Σ^n 为 Σ 上长度为 n 的字符串集合. 如果 $\Sigma=\{0,1\}$, 有

$$\begin{aligned} \Sigma^0 &= \{\varepsilon\} \\ \Sigma^1 &= \{0,1\} \\ \Sigma^2 &= \{00,01,10,11\} \\ \Sigma^3 &= \{000,001,010,011,100,101,110,111\} \\ &\vdots \end{aligned}$$

   一般来说, 我们不明确区分长度为 1 的字符串与字符, 简单的认为 $\Sigma=\Sigma^1$.

9. 克林闭包 (*Kleene Closure*):

$$\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i.$$

10. 正闭包 (*Positive Closure*):

$$\Sigma^+ = \bigcup_{i=1}^{\infty} \Sigma^i.$$

显然,

$$\Sigma^* = \Sigma^+ \cup \{\varepsilon\}.$$

其他概念如有向图, 树, 字符串的前缀, 后缀等定义这里省略.

1.2.2 语言和问题

定义. *若 Σ 为字母表且 $\forall L \subseteq \Sigma^*$, 则 L 称为字母表 Σ 上的语言.*

- 自然语言, 程序设计语言等
- $\left\{0^n 1^n \,\middle|\, n \geq 0\right\}$
- The set of strings of 0's and 1's with an equal number of each:

$$\{\varepsilon, 01, 10, 0011, 0101, 1100, \ldots\}$$

- $\varnothing$, $\{\varepsilon\}$ 和 Σ^* 都是任意字母表 Σ 上的语言, 但注意 $\varnothing \neq \{\varepsilon\}$

关于语言

唯一重要的约束就是所有字母表都是有穷的.

自动机理论中的典型问题

判断给定的字符串 w 是否属于某个具体的语言 L,

$$w \in L?$$

- 任何所谓问题, 都可以转为语言成员性的问题
- 语言和问题其实是相同的

“$w \in L$?” 也称为语言的成员性问题, 这样的问题是具有广泛性的, 各种实际问题都可以通过编码等方式转换成这种问题. 语言和问题其实是相同的东西. 我们想用这种特定的语言问题, 去探索类似的一般性问题的通用解法.

我们可以让 w 是参数, 比如是一个数字, 让 L 是一个具体的语言 (集合), 比如是所有的素数, 那这个问题就是判断 “某数字是否是素数?” 我们关心的是, 是否能用某种计算模型来回答它, 无论答案是肯定的还是否定的, 都能明确回答. 显然这个判断数字是否为素数的问题是可以被回答的, 因为我们可以逐个检查比它小的数是否是它的因子. 无论答案是肯定或否定, 都能明确回答的, 这样的问题, 我们称为是可判定的问题.

如果我们让 L 以一种特定的规则来描述, 比如, 让 w 是某个程序设计语言 (比如 C) 的程序源码, 而 L 以该语言的语法规则描述, 那么这个问题就是 “程序 w 能否被正确的编译?” 这就是程序设计语言和编译器设计首先要考虑的问题. 是否容易回答依赖于语法规则的设计是否合理. 我们将会看到, 只有当满足特定条件的语法规则时, 这个问题才容易实现, 如果语法规则过于自由, 是无法设计编译器的. 正是因为计算理论在这些方面的研究成果, 才使得现在程序设计语言与编译器的设计变得容易.

如果我们再放宽限制, 将 w 和 L 都看做是可变的参数, 那么解决这样问题的方法, 如果存在, 将会非常有用. 同样, 我们会看到, 有些这样的问题, 是干脆无法回答的; 而有些, 对肯定的答案可以明确回答, 但对否定的答案则不然, 其中最著名的就是图灵停机问题, 这些都是不可判定的问题, 在课程的最后两章我们会详细的介绍.

1.2.3 形式化证明

形式化证明: 演绎法, 归纳法和反证法

例 1. 若 x 和 y 是 Σ 上的字符串, 请证明 $|xy| = |x| + |y|$.

证明: 通过对 $|y|$ 的归纳来证明

1. 基础: 当 $|y| = 0$, 即 $y = \varepsilon$

$$\begin{aligned} |x\varepsilon| &= |x| && \text{连接的定义} \\ &= |x| + |\varepsilon| && \text{长度的定义} \end{aligned}$$

2. 递推: 假设 $|y| = n\ (n \geq 0)$ 时命题成立, 那么当 $|y| = n + 1$, 即 $y = wa$

$$\begin{aligned} |x(wa)| &= |(xw)a| && \text{连接的定义} \\ &= |xw| + 1 && \text{长度的定义} \\ &= |x| + |w| + 1 && \text{归纳假设} \\ &= |x| + |wa| && \text{长度的定义} \end{aligned}$$

□

证明: 通过对 y 的结构归纳来证明

1. 基础: $y = \varepsilon$ 时

$$\begin{aligned} |x\varepsilon| &= |x| && \text{连接的定义} \\ &= |x| + |\varepsilon| && \text{长度的定义} \end{aligned}$$

2. 递推: 假设 $y = w\,(w \in \Sigma^*)$ 时命题成立, 那么当 $y = wa$ 时

$$\begin{aligned} |x(wa)| &= |(xw)a| && \text{连接的定义} \\ &= |xw| + 1 && \text{长度的定义} \\ &= |x| + |w| + 1 && \text{归纳假设} \\ &= |x| + |wa| && \text{长度的定义} \end{aligned}$$

□

然而, 具体的定义和证明本身并不是重点, 重要的是通过这些内容, 熟悉定义与形式化证明的方法, 特别是递归定义与归纳法证明. 字符串是简单明确的, 非常适合用来作为形式化研究的对象. 课程中很多地方使用了递归定义和归纳法证明, 不要拘泥于定义、定理和证明自身, 要思考其中的结构.

1.3 练习题

1. 字符串的长度可以是任意的, 那么是否也可以是无穷长的?
2. 任意有穷集合的克林闭包, 是否一定是无穷集合?

chunyu@hit.edu.cn
http://iilab.net/chunyu