形式语言与自动机 复习提纲

第2章 有穷状态自动机 (Finite State Automata, FA)

- **核心概念与定义**：引入**有限自动机**模型，包括**确定性有穷自动机**(DFA)和**非确定性有穷自动机**(NFA)。DFA由五元组$(Q, \Sigma, \delta, q_0, F)$定义，其中$Q$为有限状态集合，$\Sigma$为输入字母表，$\delta$为状态转移函数，$q_0$为初始状态，$F$为接受状态集。NFA允许存在$\epsilon$-移动和对给定输入符号有多个或无转移。DFA/NFA都通过状态迁移来读入字符串，最终状态属于F即表示接受。形式定义上，DFA和NFA所**识别的语言**集合相同，都定义**正则语言**（3型语言）类。
- **模型与构造方法**：掌握从**NFA构造等价DFA**的**子集构造法**（子集算法）：将NFA所有可能状态子集视作DFA单一状态，实现NFA到DFA的转换（包括处理ϵ-闭包）。理解**带ϵ-转换的NFA**的处理方法（先求ϵ闭包再转移）。能够根据给定的描述（如用自然语言给出的字符串模式）构造相应的DFA/NFA，以及从状态转移图形式化为五元组定义。掌握**带输出的有限自动机**模型：**Moore机**和**Mealy机**（状态关联输出或转移关联输出），了解它们的输出功能及与DFA的关系（**Moore机与Mealy机是等价的**）。
- **重要性质与定理**：理解**NFA与DFA等价性**：二者识别相同的语言类，即对任意NFA都存在等价DFA。NFA的非确定性不增加识别语言的能力，但可能使状态数指数增长。若两个自动机识别的语言相同，则称它们**等价**。掌握**自动机接受语言**的定义：记为$L(M)$。了解**正则语言**的定义：存在某个有限自动机识别的语言即为正则语言。正则语言也称3型语言，与后续章节正则文法对应。
- **典型题型**：考试常考**术语解释**（例如"确定性有穷自动机的定义及组成部分"）、**区别比较**（如DFA与NFA的区别）。**构造设计类题**包括："为给定语言构造一个最简DFA/NFA"，"将给定NFA转换为等价DFA""绘制自动门或电梯等简单控制系统的状态图模型"等。也可能出现**运行模拟**：给出自动机和输入串，要求给出状态转换过程和结果。考试**重点**在于能正确设计**状态转换图**，写出形式化五元组，以及证明简单性质（如NFA转换为DFA的过程）。
- **常见易错点**：在定义DFA/NFA时遗漏初态或终态；绘制状态图时遗漏必需的转换（尤其是对NFA的ϵ转换或DFA的"陷阱状态"）。混淆NFA和DFA的特点，例如误认为NFA比DFA识别语言更强（实际上能力等价）。在用子集法构造DFA时，易错在计算ϵ-闭包和转换函数。对于Moore/Mealy机，常见错误是弄混输出函数的定义（状态输出 vs 转移输出）。另一个易错点是判断两个自动机等价时，仅对比状态而未严格对比语言，应以语言判定等价。
- **与其他章节联系：正则语言**概念在本章初步提出，后续第4章正则表达式、第5章正则性质中进一步讨论。实际证明**正则语言 = 有限自动机可识别语言 = 3型文法生成语言**这一等价性贯穿多个章节。有限自动机的局限为后续引出更强模型做铺垫：例如，第5章通过**泵引理**证明一些语言非正则；第6章将介绍更强的下推自动机模型处理非正则但上下文无关的语言。

第3章 文法 (Formal Grammars)

- **核心概念与定义**：文法是用于生成语言的形式规则集合。**形式文法**一般表示为四元组$G = (V, T, P, S)$，其中V是非终结符集合（变量符号集合），T是终结符集合（字母表符号），P是产生式集合（形如$\alpha \to \beta$的替换规则），S是开始符号。产生式规则描述如何用非终结符串α替换为β（β可以包括终结符和非终结符），从而逐步推导出终结符串。文法G生成的语言定义为：$L(G) = w \in T^* \mid S \Rightarrow^* w$，即从开始符号经若干步推导得到的由终结符组成的字符串集合。推导过程中，每一步可选择一个字符串中的非终结符，根据某条产生式替换它。*左线性*或*右线性*等术语描述产生式右部非终结符的位置，对应特殊类型文法。
- **文法的乔姆斯基体系**：掌握**乔姆斯基文法分级**（Chomsky Hierarchy）按产生式形式对文法分类为四种类型：① **0型文法**（无约束文法）：产生式无限制，能生成**类型0语言**（短语结构语言），等价于图灵机可识别语言；② **1型文法**（上下文有关文法，CSG）：产生式要求左部长度≤右部长度（保证不缩短字符串），生成**上下文有关语言**，等价于线性有界自动机（LBA）可识别语言；③ **2型文法**（上下文无关文

法，CFG）：产生式左部必须是单一非终结符，生成**上下文无关语言**，等价于下推自动机可识别语言（第6章详述）；④ **3型文法**（正则文法，RG）：产生式形式受限，每条规则要么是$A \to aB$或$A \to a$（右线性文法）或等价的左线性形式，生成**正则语言**，等价于有限自动机识别的语言。需要注意**左线性文法**与**右线性文法**生成语言相同，都属于正则语言范畴。正则文法是右线性文法的特例，也可证明一般的左线性文法可以等价转化为右线性形式。

- **典型题型：概念解释**类题目如"形式文法的定义"，"上下文无关文法与上下文有关文法的区别"。**判断分类**题给出若干产生式规则，要求判定所属的乔姆斯基文法类型（0型/1型/2型/3型）。**构造生成**题如"给出一种文法，生成特定模式的字符串"（例如某语言的文法设计）。也可能要求**推导过程**演示：给定文法和字符串，写出推导步骤或构造语法分析树。文法体系的层次关系和对应的自动机模型常作为**填空或选择题**出现，例如填写"正则文法对应的自动机模型是_____"（答案：有限自动机）等。
- **常见易错点**：容易混淆文法中**非终结符**和**终结符**的角色（非终结符只能通过规则替换掉，终结符不能再替换）。判定文法类型时，忽视了产生式限制条件，如误将左部含多个符号的文法当作上下文无关文法。编写文法规则时可能遗漏或多写符号，导致生成语言不符要求或文法不符合规范（如正则文法右部不得有两个以上非终结符串连）。学生有时对**文法与语言**的关系不明确：文法是生成字符串的规则集，而语言是文法生成的所有字符串集合。另一个易错点是在文法派生过程中，没有区分*直接推导*与*推导闭包*概念，未理解"$\Rightarrow$"和"$\Rightarrow^*$"的区别（一步推导 vs 多步推导）。
- **与其他章节联系**：文法的形式化定义提供了另一种描述语言的方法，后续章节将经常与自动机进行对比。**正则文法**（类型3）生成的语言正是**正则语言**，与第2章有限自动机及第4章正则表达式描绘的是同一类语言。**上下文无关文法**（类型2）生成**上下文无关语言**，对应第6章的下推自动机模型。类型0文法对应第8章图灵机模型的可识别语言，全貌涵盖于计算理论。理解文法分级有助于把握不同语言类的能力边界，例如为什么需要引入PDA/TM等更强模型。

第4章 正则表达式 (Regular Expressions, RE)

- **核心概念与定义**：正则表达式是描述正则语言的另一种形式化工具。基本**正则运算**包括**并集 (+ 或 $\cup$)**、**连接**和**Kleene星号 (*)**。正则表达式的递归定义：空集$\varnothing$和空串ϵ是基本正则表达式；字母表Σ中的任意符号也是正则表达式；若r_1, r_2是正则表达式，则$(r_1 + r_2)$、$(r_1 r_2)$和(r_1^*)也是正则表达式。一个正则表达式对应于一个语言，称为该表达式表示的语言，例如正则表达式0(10)*表示所有以0开头且尾随若干"10"重复的字符串集合。需要熟悉常用正则表达式简写，如字符集合、范围表示等（若课程涉及）。理解正则表达式描述语言**的方式与自动机/文法的区别：正则表达式更偏向模式描述。
- **正则表达式与FA的等价**：重要结论是**正则表达式与有限自动机具有等价的描述能力**。具体而言：每个正则表达式都可转换为某个等价的NFA（例如通过Thompson构造法），从而证明**RE** $\Longrightarrow$ **FA**（正则表达式定义的语言是正则语言）；反之，对于任意有限自动机（DFA/NFA），都存在某个正则表达式描述其识别的语言（可通过子集构造法及状态消除法得到）**FA** $\Longrightarrow$ **RE**。因此，"**一个语言是正则的当且仅当存在一个正则表达式描述它**"。此外，了解**正则表达式的代数定律**：包括并运算的交换律和结合律、连接的结合律、分配律、幂等律以及与闭包相关的恒等式等。这些代数性质可用于化简或变换正则表达式，但应用时需注意等价转换保持语言不变。
- **应用与实例**：正则表达式广泛应用于文本模式匹配（如UNIX的grep命令）、**词法分析**器的模式规则等。考试中可能出现简单应用题，例如给出一个正则表达式，要求写出若干满足条件的字符串，或判断某字符串是否被该表达式匹配。
- **典型题型：构造类题**：根据语言描述要求写出正则表达式，例如"写出包含子串01的所有二进制串的正则表达式"。反向也可能考查**由正则表达式构造NFA**（例如利用Thompson算法绘出ϵ-NFA）。**等价转换**题：如证明两个正则表达式等价（通常通过构造两者各自的自动机并互相转换或利用代数定律化简）。**术语解释**可能涉及"正则表达式的定义""正则运算的含义"等。**综合题**有时会将本章与第2章结合，如给出一个语言描述，要求同时写出正则表达式和画出自动机，以验证二者等价。正则表达式的**应用题**例如要求写出正则表达式来验证简单的输入格式（如Email的简化格式），这些通常作为选做或加分。考试重点仍在理解**RE与自动机的等价性**和基本构造技巧。

- **常见易错点**：书写正则表达式时**括号**易漏掉，导致运算优先级混淆（如连接与并的优先级，需要用括号明确结合顺序）。误用正则运算符，例如将"∅"*错用于表示"任意字符"（实际上*表示前一模式的Kleene闭包）。构造正则表达式漏涵盖边界情况，如忘记ϵ可能需要显式包含。将正则表达式与正则文法概念混淆：正则文法是一类产生式规则，而正则表达式是一种模式语言，两者等价于描述相同语言类但形式不同。学生在转换NFA到正则表达式时，常犯错误包括状态消除顺序不当或遗漏路径，导致表达式错误。应注意按照系统方法（如**GNFA状态消除法**或**Arden引理**）逐步推导正则表达式，以确保完整涵盖所有路径。
- **与其他章节联系**：本章与第2章互为镜像：正则表达式和有限自动机描述的是同一类语言（正则集），提供了验证两种表述等价性的思路。第5章将讨论正则语言的性质和判定法，其中正则表达式作为正则性的判定手段之一。在编译原理等后续课程中，正则表达式与有限自动机（DFA最小机）结合用于**词法分析**，体现了理论与实践的联系。正则表达式的限制也为引出更高级的语言描述方式（如上下文无关文法）做准备。

第5章 正则语言的性质 (Properties of Regular Languages)

- **FA与正则文法等价性**：首先确认**正则语言等价于正则文法生成语言**，即每个正则语言都有等价的3型文法（右线性文法）生成。构造方法包括从DFA逆向构造正则文法（状态对应非终结符，转移对应产生式）等。这进一步巩固了**正则语言 = 有限自动机识别 = 正则表达式描述 = 正则文法生成**的结论。
- **正则语言的泵引理**：掌握**正则语言泵引理**（pumping lemma）的内容和使用方法。泵引理表述：对于每个无限的正则语言L，存在一个最小泵长度p，使得L中任意长度不小于p的字符串都可拆分为三个部分$s = xyz$，满足拆分后中间部分y可反复"抽泵"仍留在语言中。理解泵引理的**必要条件**性质：**凡是正则语言必须满足泵引理**（有限语言真子集情况特殊但也可视作满足泵引理的 vacuous 情况）。会用泵引理**证明语言非正则**：典型步骤是反设语言正则，获得泵长度p，选择特定字符串使任意切分都导致违反语言性质，从而矛盾证明语言不是正则的。注意泵引理只是正则性的必要条件，不是充分条件——不能证明某语言正则，只能证明非正则。
- **正则语言的封闭性质**：熟记**正则语言**对多种运算的**封闭性**。封闭运算包括：并、交、补、差、连接（串联）、Kleene星（闭包）、正闭包、逆（倒置字符串）、同态像及其逆像等。需要能举例说明，如L_1, L_2是正则语言，则$L_1 \cup L_2$、$L_1 \cap L_2$、L_1^*等仍是正则语言；而对非正则语言进行这些运算可能产生非正则结果。要特别注意正则语言类对交和补也是封闭的（与上下文无关语言的对比，后者对交和补不封闭）。典型考点是给出某些语言运算组合，判断结果是否一定正则，并要求说明理由（可通过构造相应自动机或利用已有封闭性结论）。
- **正则语言的判定算法**：了解针对正则语言的一系列**判定算法**，即存在有效算法判定某些性质：例如**成员判定**（给定DFA/NFA和字符串，能在线性时间判定字符串是否被接受）、**空语言判定**（检查自动机是否有从初态到终态的可达路径）、**有限性判定**（检查自动机有无循环路径可回到终态）、**等价判定**（判断两DFA是否识别相同语言，可通过构造差异自动机或直接最小化比较状态集合）。特别重要的是**DFA最小化**算法，它既用于等价判定又用于减少状态：常用的方法是**区分状态的划分算法**（不断分裂合并等价状态集合）或基于**Myhill-Nerode定理**判断状态可区分的准则。掌握*最小DFA存在且唯一*（同构意义下）的结论，理解等价状态概念以及如何合并。
- **典型题型：证明题**：利用泵引理证明某语言非正则（如$L = a^n b^n$非正则的经典证明）。**闭包运算题**：给出若干语言关系，问某运算结果是否为正则语言（如"正则语言对逆串运算封闭吗？"等，需要回答是/否并说明依据）。**构造题**：根据需要构造自动机实现集合运算，如构造识别$L_1 \cap L_2$的自动机（笛卡尔积法）。**算法题**：DFA最小化过程（给一个DFA图或转移表，要求划分等价类得到最简状态图），或判定两机等价（可转为让学生找出区别两机语言的字符串）。**快问快答**式小题：列举正则语言封闭的运算；给出两个正则表达式判断其表示语言是否相同等。考查重点在灵活运用正则性质解决问题，如利用封闭性和已知正则语言构造新语言证明其正则，或通过反例论证非封闭情况。
- **常见易错点**：运用泵引理时选取错误的字符串或切分方式，导致证明逻辑不严谨。需要注意$s = xyz$拆分时条件（$|xy| \geq p$, $|y| > 0$）以及在所有可能切分下都应推出矛盾。关于封闭性，学生易混淆上下文无关语言的封闭性质，将CFL的不封闭运算误用于正则语言（如错误地认为正则语言对交不封闭，其实正

则是封闭的）。DFA最小化时，易错在**初步划分**终态/非终态集合，以及迭代划分过程中遗漏某些对。例如不区分不同不可达状态与可达状态的合并，或合并了本不应合并的状态。等价判断中，常见错误是只对比两机的语言描述而未严格算法验证；**应以严格算法或构造证明**两者语言一致或给出反例串。记忆结论时，也需避免张冠李戴，例如误记"正则语言对补不封闭"这类与事实相反的说法。

- **与其他章节联系**：正则语言的性质是对第2-4章内容的深化和总结。本章泵引理与第7章的**上下文无关语言泵引理**形成类比，均用于判定非属于某语言类。本章的封闭性和判定算法概念在第7章对CFL、以及第9章不可判定性中形成对照：正则语言类性质良好，很多判定问题算法存在，而对更高级语言类则有的变得不可判定。DFA最小化与等价判定思想也用于更高层模型的研究，比如判定两个CFG是否等价是不可判定的（这体现了随着模型能力提升，某些问题变得更困难乃至无解）。因此，通过正则性质的学习，学生既掌握了解决实际问题的方法，也领会了计算理论中不同语言类别的差异。

第6章 上下文无关文法与下推自动机 (Context-Free Grammar & PDA)

- **上下文无关文法 (CFG)**：CFG定义与文法一般定义类似，但**产生式左部严格为单一非终结符**。形式定义：$G=(V,T,P,S)$，每条产生式形如$A\to\alpha$，其中$A\in V$，$\alpha\in(V\cup T)^*$。CFG生成的语言称为上下文无关语言 (CFL)。典型示例：文法G：$S\to 0S1\mid\epsilon$生成语言$0^n1^n\mid n\geq 0$。理解CFG推导串的过程：左侧变量可不考虑上下文独立替换为右侧串。掌握**语法分析树**(Parse Tree)概念：是表示推导过程的树形结构，根节点为开始符号，每个内部节点标记为一个变量，叶节点依次构成推导生成的终结符串。语法树形象反映字符串的层次结构和文法的递归性质。
- **句法分析与歧义**：推导过程存在不同策略，如**最左推导**和**最右推导**，分别每步替换最左或最右的非终结符。对于每个CFG，可以通过画语法树检查某字符串的推导是否唯一；如果存在某个字符串可以有两棵不同的语法分析树（等价地，两种不同的推导方式）生成，则文法被称为**二义性文法（歧义文法）**。**定义:** 若$\exists w\in L(G)$有两种不同的解析树，则G是二义的，否则为无二义。歧义文法会导致解析的多重解释，如经典例子算术表达式文法在缺乏运算符优先级规则时是二义的。应能判断简单文法是否二义，例如通过找到产生二义的字符串对比两种解析树。需要注意**二义性是文法的性质，不是语言的固有性质**：有些语言存在无二义文法，但也有语言本身**固有二义**无法找到无二义文法生成。
- **下推自动机 (PDA)**：PDA是一种增加了**栈存储**的自动机模型，用于识别CFL。形式定义：$P=(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$，其中$Q$为有限状态集，$\Sigma$为输入字母表，$\Gamma$为栈字母表，$q_0$为初始状态，$Z_0$为初始栈底符号，$F$为接受状态集（针对**终态接受**定义；另一种等价方式是**空栈接受**）。转移函数$\delta: Q\times(\Sigma\cup\epsilon)\times\Gamma\to 2^{Q\times\Gamma^*}$，即根据当前状态、当前读入符号（可为空）和栈顶符号，转移到新状态并可能更改栈顶（出栈入栈若干符号）。PDA的运行通过读串和栈操作结合，当输入读完且满足接受条件（终态$\in F$或栈空）则接受。
- **CFG与PDA的等价性**：核心理论结果：**上下文无关语言当且仅当存在一台下推自动机可以识别它**。也就是说，CFG生成的语言类正好是PDA可接受的语言类。这提供了两种不同视角理解CFL：一种是生成式（文法），一种是识别式（自动机）。需要掌握**等价性的构造证明**思路：（*自CFG到PDA*）给定CFG可构造等价PDA：采用**非确定性**策略，遇到栈顶变量时推展所有可能产生式右部符号入栈，模拟文法推导过程，栈空接受即可识别文法语言；（*自PDA到CFG*）给定PDA则可构造等价CFG：非正式思路是在PDA状态对之间用变量表示从一个状态经某栈变化到另一状态能生成的字符串，从而提炼出产生式。理解由于PDA具有非确定性，CFG往往对应非确定的分析过程，这也是某些CFL无法由确定性PDA（DPDA）识别的原因。课程可能强调**确定性下推自动机**(DPDA)的概念及其识别的**DCFL**是CFL的真子集，而且DPDA识别的语言都是无二义的，常用于实际编译中的语法分析。
- **典型题型：构造文法**：根据给定语言描述构造CFG（如平衡括号，回文奇偶分割等）。**等价转换**：如"给定CFG构造等价PDA"或反之（通常考查简单文法/PDA，例如$L=a^nb^n|n\geq 0$的文法与PDA互构）。**推导及语法树**：给出文法和字符串，要求给出推导过程或画出语法树；若文法可能二义，要求给出一字符串的两棵不同语法树以证明。**设计PDA**：根据语言要求设计下推自动机，包括状态-栈转移图或形式化七元组描述。例如常考"设计PDA识别$0^n1^n|n\geq 0$"或者"识别形如wcw^R的语言的PDA"等。**术语解释**：如"何为上下文无关文法"，"下推自动机如何工作"，"二义性如何判定"等。考试重点在于**CFG与PDA两种模型的转换和应用**：能否灵活使用文法规则和栈操作实现对特定模式语言的生成和识别。

- **常见易错点**：CFG构造中容易犯规则设计不完备或多余的错误：例如生成某类串时漏掉递归规则或多写导致语言扩大。设计文法要注意避免不必要的二义性（虽然不要求一定无二义，但有时可优化）。绘制语法树时，学生常把终结符误放在内部节点或将非终结符当叶节点。判断文法二义性有难度，易错判断包括以为"左递归文法就是二义的"（其实左递归可有等价无左递归文法，不等于二义）等。PDA设计误区：状态与栈配合不当，例如遗忘在处理完输入后还需检查栈空或状态接受；对需要匹配多个符号的情况，入栈/出栈顺序混乱。特别是处理嵌套结构时，栈符号的选择和含义容易搞错。转换CFG到PDA时，推入栈的串顺序经常写反（应注意文法右部α的最后一个符号先入栈，这样栈顶顺序与推导顺序一致）。从PDA构造CFG时，容易遗漏状态对的组合，导致文法不完整。需牢记PDA两种接受方式等价且可相互转换；有时学生混淆了"终态接受"和"空栈接受"的区别，答题时应根据题目指定采用一致的接受方式。
- **与其他章节联系**：本章是对第3章CFG和第2章自动机概念的拓展——引入栈使自动机有了记忆能力，与CFG的递归规则正对应。**语法分析树**揭示了文法推导的结构，与编译原理中的语法树概念一致，也是在下一章讨论范式转换和算法的基础。**二义性**问题在理论上联系到第7章某些CFL性质（如无法判定文法是否二义），在实用上预示了解决方案（如限定使用LL(1)/LR(1)无二义文法）。**CFG-PDA等价定理**对应于正则文法-FA等价（第5章）的一种提升，对应的证明技巧也会用于更复杂模型的比较。通过CFG和PDA模型，埋下了通向图灵机（第8章）的伏笔：可以看到，即使有了栈，PDA仍无法识别一些更复杂语言（如$a^n b^n c^n$），这引出对更强大计算模型的需求。

第7章 上下文无关文法的性质 (Properties of Context-Free Languages)

- **文法范式转换**：掌握CFG的标准化形式，尤其是**乔姆斯基范式**(Chomsky Normal Form, CNF)。**Chomsky范式**要求产生式要么形如$A \to BC$（两个非终结符）要么形如$A \to a$（单个终结符），且允许$S \to \epsilon$作为特例。考试关注**文法转换步骤**：消除ϵ产生式、消除单一变量产生式、消除无用符号，然后将产生式化为CNF形式。需要能将一般CFG转换为等价的CNF文法，并理解这种规范形式在证明和算法（如CYK算法）中的用途（简化推导长度的上界）。另有**格雷巴赫范式**(Greibach Normal Form, GNF)可简单了解，其特点是每条产生式右侧以终结符开头，但一般不重点考核。
- **上下文无关语言的泵引理**：类似正则情况，CFL也有**抽象的泵引理**性质。**泵引理表述**：存在一个泵长度p，对任何在语言中的长串（$|s| \geq p$），都可拆分为5段$s = uvwxy$，满足条件：$|vwx| \leq p$（中间三段总长不超p）、$|vx| > 0$（可抽出的部分非空），且对任意$i \geq 0$有$uv^i wx^i y \in L$。这个引理同样用于**证明语言非CFL**。考题一般让证明一些复杂语言（如$a^n b^n c^n$或$a^i b^j c^k \mid i = j$或$j = k$等）不是CFL，需要正确应用泵引理：关键是巧选字符串并论证抽泵会破坏字符串之间的关联关系。要注意CFL泵引理和正则泵引理的区别：CFL的被抽段是两段(v和x可分别抽，多用于证明涉及两段相关性的语言非上下文无关)。
- **上下文无关语言的封闭性**：记牢**CFL的封闭运算**性质。CFL在**并、串连接、Kleene闭包**等运算下是封闭的，这些可以通过文法或PDA构造证明：如L_1, L_2是CFL，则$L_1 \cup L_2$也是CFL（可构造选择文法规则或PDA增加新初态ϵ转移到两个子机）；串连接和闭包的封闭性证明类似。CFL对**同态像、逆同态**以及**替换**（将符号替换为语言）的运算也封闭。但需要注意CFL**一般不对交、补运算封闭**：两个CFL的交集不一定是CFL，一个CFL的补集也不一定是CFL。例如$L_1 = a^n b^n c^m, L_2 = a^m b^n c^n$ 均为CFL，但$L_1 \cap L_2 = a^n b^n c^n$ 已知不是CFL，就是交集不封闭的反例。对于补集，由于交和补在所有语言类中满足德摩根律，可知如果对补不封闭则对交也不封闭。**特例**：CFL**与正则语言交**仍是CFL，可通过PDA和DFA构造同步运行判定串属不属于来证明（利用正则语言约束栈操作）。这一特性常用于证明某语言非CFL：若怀疑L不是CFL，可尝试找一个正则语言R使得$L \cap R$容易证明非CFL，从而印证L非CFL。
- **上下文无关语言的判定算法**：与正则类比，CFL也有一些判定问题可解：**判空**问题（给定CFG，能判定其语言是否为空集）是可判定的，通过找能产生终结符串的变量集迭代计算可解；**判有限**问题（判定CFG生成的语言是否有限）也是可判定的，可通过检测文法是否存在循环生成无限长串来实现；**成员判定**问题（给定字符串是否属于CFG语言）在多项式时间可解，典型算法是**CYK算法**（需CFG为CNF）在$O(n^3)$时间判定，或利用PDA模拟在指数时间内判定（更优化的Earley算法等也是多项式）。这意味着CFL是可以有效解析的（这点在编译实践中十分重要）。另一方面，**等价性**问题（判定两CFG产生相同语

言）是不可判定的，**二义性判定**也被认为不可判定——这些结果通常不要求详证但可能以结论形式出现。考试可能要求了解**CYK算法**的填表思路或复杂度，但具体实现步骤一般不深究。

- **典型题型：文法转换题**：将给定CFG化为等价的Chomsky范式文法（步骤包括增加新开始符号处理ϵ，消除ϵ产生式、单一产生式、无用符号，最终写出CNF产生式集）。**泵引理证明题**：经典题目如证明$a^n b^n c^n$不是CFL或其他需同时保持两段以上对应关系的语言非上下文无关。**封闭性判断题**：判断给定运算对CFL是否封闭，并要求给出证明或反例（如"CFL 对补集运算封闭吗？请证明或举反例"）。**算法应用题**：给出一CFG和字符串，要求用CYK算法判定归类（可能以小规模文法矩阵演示）；或者给出一CFG问其语言为空或有限（考查对文法产生能力的分析，如找不可达/无法终结的变量）。**术语解释**：如"Chomsky范式是什么"，，"上下文无关语言对哪些运算封闭"等。综合题可能涉及比较正则语言与CFL的异同、层级包含关系等。
- **常见易错点**：范式转换时步骤顺序和细节易出错：比如**消除ϵ产生式**后忘记处理空串情况导致与原语言不等价，**消除单一产生式**时替换不完全。转换CNF特别要小心新引入的变量符号和规则正确性，学生有时遗漏某些替代，或在替换终结符为变量时不小心引入了无用规则。应用CFL泵引理时，比正则情况更复杂，常见错误是没有覆盖所有情况（如v和x可能在串不同部分，需要分类讨论）。要明确每一步推理，否则容易漏掉一种可能切分使论证不严谨。闭包性质判断易错在混淆正则和上下文无关的闭包运算，如错认为CFL对交封闭。反例构造需要语言既是两个CFL的交又已知非CFL，这对基础薄弱同学有难度，易举出错误示例。算法判定方面，容易误以为CFG等价或二义问题可判定，需要记住这些是**不可判定问题**。在CYK算法应用中，漏考虑某些划分使判定失误，也是常见问题。
- **与其他章节联系**：本章CFL性质与第5章正则性质对比鲜明。正则语言在封闭性和判定问题上更强，而CFL有所欠缺，例如交和补的不封闭以及某些问题不可判定，预示了更强模型（类型1、0语言）的复杂性。**泵引理**在正则与上下文无关两类中都有体现，方法类比但细节不同。**文法范式**的转换和判定算法为下一步理解第9章的图灵可判定性埋下基础：CFG判定算法的存在体现了CFL类问题是可计算的；而第9章很多更复杂的问题（如CFG等价）就超出了算法能力范围。实践上，本章知识与编译原理课程相关：无二义的CFL和DPDA对应语法分析算法（LL、LR分析），CNF和CYK算法对应语法分析程序设计等。掌握CFL性质不仅有助于理论考试，在工程上也有意义。

第8章 图灵机 (Turing Machine, TM) – *要点提炼*

- **图灵机模型**： 图灵机是更强大的计算模型，由**无限纸带**（既做输入又可读写、无限延伸）、**读写头**（可左右移动）、**有限控制**（状态集合）组成。形式化定义通常为七元组$(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$，其中$\Sigma$是输入符号集（不含空白符），$\Gamma$是带符号集（包含$\Sigma$及空白符␣），$\delta$是转移函数（可根据当前状态和读到符号决定写符号、移动方向、下一个状态）。图灵机每一步可以读写符号并移动头位置，实现对输入的**算法式操作**。图灵机接受一种**"半无限"计算能力**：既能无限循环（对拒绝情况可能不终止），也能无限使用存储（靠纸带）。
- **确定型 vs 非确定型**：了解**非确定性图灵机**的模型差异：转移函数δ可返回多个后继配置。但重要结论是**确定型图灵机和非确定型图灵机在判定能力上是等价的**（非确定性不能增加可判定语言的集合，尽管可能影响时间复杂度）。考试可能不会深究证明，只需了解**所有非确定型TM都有等价的确定型TM**。另外，**多带图灵机**（多条带并行操作）等变种虽然方便设计算法，但**在计算能力上与单带TM等价**；这一点反映了图灵机模型的鲁棒性：只要允许多项式时间模拟，不改变可计算问题的类别。
- **图灵可计算与丘奇-图灵论题**：理解**算法**的正式化含义**：丘奇-图灵论题**提出任何现实可实现的计算过程都不超出图灵机的计算能力。即图灵机被认为是**算法的精确定义**。该论题不是严格定理，但被广泛接受。由此，"可计算的"通常等价于"存在某个图灵机可计算"。这奠定了计算理论基础：所有现代编程语言或计算模型（RAM机、Lambda演算等）在**计算泛化能力**上是等价的。考题可能涉及此概念的理解，例如填空"丘奇-图灵论题认为：___"。
- **可判定和可识别语言**：区分**图灵可判定语言**（又称递归语言）与**图灵可识别语言**（又称半可判定/递归可枚举语言）。**可判定语言**是存在一个图灵机总能在有限步骤内接受或拒绝的语言（即有决策算法解决的判定问题）；**可识别语言**则是存在一个图灵机能接受所有在语言内的输入，但对不在语言内的输入可能不终止（只要对属于的能识别出来即可）。明确关系：可判定 ⊂ 可识别 ⊂ 所有语言。考试经常考查"半

可判定”概念，比如问：“为什么说图灵机可以识别一些语言但无法判定？”对应回答应提及**对于某些问题TM会无限循环**。常见例子：图灵机可以识别“程序是否会停机”相关语言，但不能决定其否定情况。

- **典型关键术语**：*配置*（configuration，TM在某状态下带上内容和头位置的完整描述）、*停机*（halt，TM进入接受/拒绝状态或无规则可用）、*决定（判定）*（decide，算法保证停机给出yes/no答案）。**枚举器**(Enumerator)的概念也可能涉及：它是一种输出型图灵机，列举语言中所有字符串。需要知道**类型-0文法**（短语结构文法）产生的正是**可识别语言**，对应图灵机模型。
- **考试提示**：第8章通常以概念理解为主，考试多以**填空或选择**形式出现，如填出“图灵机的基本组成部分有_____”，判断“非确定型图灵机是否比确定型更强大”（答案：否，等价），或问“什么是图灵可判定语言/半可判定语言”。也可能要求简单描述**图灵机工作过程**：例如给一个简单的TM示意图，问其功能或输出。由于篇幅限制，一般不会要求手工设计复杂TM，但基本设计思路要懂，比如“设计TM识别语言$a^n b^n | n \geq 0$”可能作为理解题。关注概念准确表述，如“无限循环”与“拒绝”要区别对待。

第9章 可判定性 (Decidability) – *要点提炼*

- **判定问题与不可判定性**：形式化一个**判定问题**为语言L，若存在图灵机判定L则称该问题**可判定**（decidable），否则**不可判定**。**不可判定问题**的存在是计算理论的重要结果。要理解经典不可判定问题示例，如**停机问题**：判定任意给定的图灵机M对输入w是否最终停机（接受或拒绝）。**著名结论：停机问题是不可判定的**。进一步，**接受问题**$A_{TM} = \langle M, w \rangle \mid M$接受$w$也是不可判定的（但$A_{TM}$是可识别的，因为可模拟$M$接受则接受，而不接受可能不终止）。理解**不可判定性的证明方法**：主要有**对角线法**和**归约法**。对角线法通过构造“对角线”表格论证某些语言（如停机问题的补集）不在可判定集合内。归约法则是实践中常用证明**：如果已知问题A不可判定，将A归约(reduce)到新问题B，**使B可判定会推导出A可判定的矛盾，则说明B不可判定。归约可以是**映射归约**（函数映射输入转换）或**图灵归约**（借助B的判定器作为子程序）。
- **典型不可判定问题**：除了停机问题$HALT_{TM}$，考核时还应了解其它经典不可判定问题：如**TM语言为空集的问题**($E_{TM} = \langle M \rangle \mid L(M) = \emptyset$)、**两TM语言是否相等**($EQ_{TM} = \langle M_1, M_2 \rangle \mid L(M_1) = L(M_2)$)**等都不可判定**。更接近之前章节的例子有：“两个CFG是否等价”也是不可判定的；“给定CFG的语言是否为无限集”其实**可判定**（因为可分析循环产生式），容易混淆需小心。**Rice定理**通常作为一般性结论给出：任何关于图灵机所识别语言的非平凡性质都不可判定，这涵盖了许多具体不可判定问题（考试可能不会让学生表述定理，但理解实例足够）。需要牢记**可识别但不可判定**的问题典型代表就是A_{TM}（接受问题），而它的对偶“不接受问题”不可识别。亦可能考问**semi-decidable**概念：半可判定语言指机器只对属于的情况有正确停机接受，对不属于的可能不停止。
- **归约证明技巧**：考试可能要求证明某问题不可判定，通常采用**已知不可判定的问题**（如A_{TM}或$HALT_{TM}$）归约到待证问题。例如证明“E_{TM}不可判定”可以假设有判定器R判定任何TM语言是否空，然后构造另一台机利用R来解A_{TM}，从而矛盾。对此要写清**归约构造**和**推理逻辑**（假如B可判定则A可判定但已知A不可判定，所以B不可判定）。通常不要求写形式化证明，但归约过程的关键细节要明确。
- **典型关键术语**：*判定(decide)*、*识别(recognize)*、*不可判定(undecidable)*、*归约(reduction)*、*A_{TM}接受语言*、*$HALT_{TM}$停机语言*、*问题的编码$\langle M \rangle$*。学生应能解释“不可判定”的含义，以及举出至少一个不可判定问题例子。**映射归约**与**Oracle（调用）归约**在课件中提及，了解映射归约是函数f使$x \in A \Leftrightarrow f(x) \in B$，Oracle归约是允许在过程中调用假设的判定黑箱来解决子问题。
- **考试提示**：本章重在概念理解和基本论证。常见题型有**判断题/选择题**：例如“判断：存在图灵机可以判定任意给定两TM是否等价”（答案：错误）。**简答题**：解释为什么停机问题不可判定，可从对角线法角度简述或者归约角度说明。**证明题**（较难）：给出一个新问题，让学生尝试归约证明其不可判定（通常会提示从已知问题归约）。**填空题**：诸如“A_{TM}是不可判定但___的”（答案：可识别），“Rice定理表明凡是关于图灵机识别语言的___性质都不可判定”。对于基础较弱的同学，需重点掌握列举的经典不可判定问题及其意义，理解可判定与可识别的区别。

第10章 计算复杂性 (Computational Complexity) – *要点提炼*

- **时间复杂度与P类**：计算复杂性关注算法（图灵机）所需的时间/空间随输入规模增长的度量。重点是**多项式时间复杂度**概念：如果存在k次多项式$p(n)$界定某问题求解所需步骤，则认为是**多项式时间可解**。**P类**是所有能在多项式时间由确定型图灵机判定的问题集合，被视为"高效可计算"问题类。理解大O符号描述渐进上界的意义，例如$O(n^2)$表示复杂度随n的平方增长。考试可能要求区分多项式级别和指数级别增长的差异（多项式时间一般可接受，指数时间则规模稍大就不可行）。常见P类问题例子：有向图连通（PATH）、素数判定(PRIMES)等都已知属于P类。
- **NP类与NP完全**： **NP类**是指能由非确定型图灵机在多项式时间判定的问题集合，等价于**存在"多项式长度证明（证书）"可供确定型机在多项式时间验证**的问题集合。初学可将NP理解为"一猜就能在多项式时间检查"的问题，比如旅行商问题猜一个路线可在多项式时间计算总长验证是否解。了解NP问题举例：可满足性(SAT)、哈密顿回路等。这些问题目前不知道是否在P类中。**NP-complete (NP完全)问题是NP中"最难"的一组，形象描述为："NP中结成团伙，互相规约，共存共荣"的问题集，每个NP完全问题都可以相互多项式归约**。正式定义：L是NP完全，当且仅当$L \in \mathrm{NP}$，且对于任何$A \in \mathrm{NP}$，A多项式时间归约到L。**Cook-Levin定理**证明了第一个NP完全问题**：命题可满足性问题 (SAT) 是NP完全**。其简化版本3-SAT也NP完全，作为种子繁衍出数以千计的著名NP完全问题。应理解**NP完全问题的重要性**：如果找到其中任何一个属于P类，则$P = NP$；反之，只要相信某个NP问题确实比多项式难，那$P \neq NP$且NP完全问题都不可能在多项式时间解决。
- **NP难与P vs NP**： **NP-hard (NP难)指不要求在NP内但至少不比NP问题更容易**的问题集合。NP难问题可能比NP完全还要难（比如停机问题属于NP-hard但不在NP）。NP-hard定义通常：任意NP问题可多项式归约到它，但它本身未必是判定型（可能是优化型问题或者更高复杂度问题）。**P vs NP猜想**是计算机科学最大的未解难题之一：目前猜想**P类不等于NP类**，即不存在多项式算法解决所有NP问题，但尚无证明。考试可能以客观题考："人们普遍认为P ___ NP"（填"≠"）。需要知道如果某NP完全问题被证明有多项式算法，则推得$P = NP$，这将颠覆整个复杂性理论。
- **典型关键术语**： *多项式时间*、*指数时间*、*类P*、*类NP*、*NPC问题*、*NP-hard*、*归约*、*Cook-Levin定理*。考题喜欢问"NP问题有什么特点"（需答非确定机多项式时间，或给出验证式定义），或"举例一个NP完全问题"。**归约证明**在复杂性里用于证明某问题NP难：如已知3-SAT NP完全，归约到新问题B以证明B也是NP完全。学生应能理解归约在判定不可行性上的作用。
- **考试提示**：复杂性部分通常考察基础概念理解，不深挖证明细节。**选择题/判断题**：如"判断：NPC问题都是NP问题"（答案：对）。**填空题**：如"第一个被证明的NP完全问题是___"（答案：SAT）；"若发现一个NP完全问题有多项式算法，则**__"（答案：P=NP）。**简答题**：可能问"何为P类问题，何为NP类问题，两者关系如何"。基础较弱同学应侧重记忆定义和意义，不必详记证明过程。复杂性部分虽在课程末尾，但至少要拿到基础分，例如明确P和NP的定义区别、知道NP完全的概念和一个例子（SAT或旅行商问题等），了解P≠NP尚未解决且意义深远。