# 《词法分析》学案

1. **学习目的**

能力获取：掌握构造词法分析器所需的基本理论、具体算法和开发工具。

1. 描述源语言词法结构的能力——设计正则表达式。学习要点：要理解词法分析本质上的研究对象是符号串集合，从而理解正则表达式的设计初衷。
2. 用有限自动机描述词法结构、进行词法分析的能力——设计NFA、DFA，理解其运转机制。
3. 衔接上述两个数学模型、构造完整词法分析器的能力——正则表达式🡪NFA的算法，NFA🡪DFA的算法，DFA最小化算法、模拟DFA运转的算法。
4. 借助自动生成工具Lex构造词法分析器的能力。

**关键词：符号串集合，单词本质是符号串集合，正则表达式的本质是符号串集合的运算，词法分析本质是处理符号串集合。**

**最重要的算法：子集构造法，重点理解由问题逐步导出算法的过程。**

**知识点的关联：Thompson构造法利用了语法制导翻译，子集构造法和DFA最小化方法都是按序枚举无穷多的符号串。**

1. **学习内容**

**第一部分：词法描述——正则表达式**

1. 基本概念：
2. 了解词法分析器的角色，语法分析器的功能函数，返回下一个单词。
3. 理解基本术语，词素——源程序中出现的符号串，单词——词素的分类，模式——单词和词素间的桥梁。理解模式形式化表达的难点，包含无穷词素的单词。
4. 理解词法分析的本质，根据模式，将源程序字符流切分，每段是一个词素，属于某类单词。
5. 理解单词属性，词法分析与语法分析的交互（回顾第2章），词法分析返回语法分析单词类别，词素的属性通过全局变量等方式返回语法分析，后续综合阶段会用到。

**重点&难点：理解单词本质上就是符号串集合🡨单词是包含很多词素的类别 & 词素是源程序中的符号串。从而理解词法分析是在处理符号串集合，后续很多概念和算法就好理解了。**

1. 正则表达式
2. 理解基本概念，第2章中已介绍的符号串、符号串集合等概念，理解符号串（集合）的运算、我们又将符号串集合称为语言。
3. 理解前人为什么设计正则表达式这一模型，本质上是为了形式化描述单词（符号串集合）。

**重点&难点：理解正则表达式的产生，思考线索：我们的目标是形式化描述单词，即符号串集合——难点是ID、NUM这些无穷集不可能通过穷举来描述——发现用有穷个简单符号串集合（有穷集）及运算可表达任意复杂的无穷符号串集合（tricky：闭包运算）（通过符号串集合运算例子那两页讲义理解这一点）——将简单符号串集合和符号串集合运算符号化🡪正则表达式**

1. 正则表达式定义和设计
2. 理解正则表达式的定义，一方面，理解上面正则表达式产生的思路，另一方面，理解它也是一种“表达式”，因此定义与熟悉的算术表达式类似，也必然是递归结构：基本规则——简单正则表达式——不可再分；递归规则——正则表达式运算——复杂正则表达式分解为简单正则表达式的运算。
3. 理解正则表达式等价的概念，本质是符号串集合！——表达的符号串集合相等；理解正则表达式运算的性质，还是符号串集合！——符号串（集合）运算的性质。
4. 掌握正规定义的描述方式，很像CFG（正则表达式描述能力包含在上下文无关文法中，本质上可认为是特殊CFG，第4章会介绍）。熟悉正则表达式的设计，熟悉符号简写。
5. 了解不能用正则表达式描述的符号串集合。

**重点&难点：正则表达式的设计，这是只能由人来做的工作，因此只能通过多看一些实例、多加练习来熟悉。一些常见符号串结构的设计，顺序结构——正则表达式连接运算，重复结构——闭包运算，可选结构——或运算。**

**正则表达式无法描述的符号串集合，了解一些典型例子。**

1. Lex程序设计
2. 了解Lex程序结构，类似Yacc的三段式，不同之处，规则段是正则表达式+动作。回忆词法分析与语法分析的交互，“动作”部分进行一些处理后，将单词类别return，词素属性赋予全局变量yylval来返回。
3. 掌握Lex编程基本语法，掌握从磁盘文件而非键盘输入以及输出到磁盘文件而非显示器的方法。

**重点&难点：规则段正则表达式的顺序不是随意的，最长前缀原则——能识别尽量长词素的单词放在前面，词素为其他单词词素前缀的单词放在后面。**

**第一部分：词法分析——有限状态自动机**

1. 状态转换图：
2. 理解状态转换图的意义，正则表达式只是词法结构的静态描述，进行词法分析需要一个动态的机制，状态转换图就是这样一种机制。
3. 理解状态转换图的构成，有向图，顶点——“状态”、词法分析进行到某个时刻，边——“动作”、指出从发出边的状态读入边上终结符的话应转向哪个状态，初态——词法分析的开始状态，终态集合——识别出单词。
4. 理解状态转换图的运行机制——词法分析过程，从初态开始；每一步读入一个符号，根据当前状态的出射边转到下一个状态；当所有符号处理完毕，若停止在某个终态，符号串属于该终态对应单词，否则，不属于此状态转换图可识别的单词。

**重点&难点：理解状态转换图的运行机制。状态本质上对应符号串集合！状态表示词法分析已经进行到特定时刻，从分析开始到此时刻，已经读入处理了从初态到此状态的路径上的符号串，而从初态到此状态可能由很多路径——符号串集合！终态对应单词！**

**状态转换图与正则表达式的对应，根据PPT中示例，观察基本正则表达式和正则表达式运算与状态转换图结构的对应关系，为后面正则表达式与自动机的转换算法打下基础。**

1. 有限自动机
2. 理解非确定和确定有限自动机概念，静态看待它，能描述单词，其动态运行机制又能实现单词识别（词法分析）。确定，状态转换是唯一的，非确定，状态转换不唯一，出于性能考虑，我们倾向于基于前者实现词法分析器。
3. 理解有限自动机数学模型，状态转换图最直观，五元组方便理论推导，状态转换矩阵适合计算机程序实现。
4. 掌握有限自动机的设计。

**重点&难点：运行机制，DFA实际上与状态转换图完全一致，NFA的区别，每一步的“下一状态”都可能有多种情况——符号串识别有多条路径——任何一条到达终态即为接受，所有路径都到达不了终态才是拒绝。**

**设计，和正则表达式、CFG的设计一样，只能人来做，没有算法。一些常见的思路：状态转换方式，从初态开始，分析词素的下一个符号是什么，相应构造状态转换（边）及下一状态，逐步构造出自动机；状态划分法，状态对应符号串集合——将符号串全集划分为几个子集、分别对应自动机的状态——根据词素间的拼接关系构造状态之间的边，结合PPT中示例理解这些常见思路。**

**第三部分：词法分析器构造——转换算法**

1. Thompson构造法
2. 首先理解接下来我们要干什么，最终目标是构造词法分析器，目前已有正则表达式（描述单词、容易设计）和有限自动机（可实现词法分析器，设计困难一些），因此，还缺少从正则表达式到自动机的转换方法。
3. 理解为什么要正则表达式🡪NFA🡪DFA🡪化简🡪模拟DFA，DFA时间复杂度低，应基于它实现词法分析器，但正则表达式🡪DFA复杂，因此以NFA作为桥梁。
4. 理解Thompson算法思想，理解这是一个语法制导翻译方法，再去理解其细节。

**重点&难点：我们要做的是正则表达式🡪NFA，也是一个翻译过程！与第2章讨论的一些翻译工作类似，我们不可能为每个正则表达式设计翻译方法（无穷可能性），还是采用语法制导翻译！为正则表达式每个语法范畴设计好翻译方法就够了！即，基本正则表达式（a、）怎么转换为NFA，正则表达式运算（或、连接、闭包）如何转化为NFA。算法的记忆就很自然了。**

1. NFA转换DFA：**重点理解子集构造法的思想是让DFA和原NFA等价**
2. 首先，如何强调这个算法的重要性都不为过！这是本课程第一个比较复杂的算法，而且是另一个算法——LR分析表构造算法的基础。关键：不要死记硬背，理解其思想！
3. 理解基本运算及其作用，闭包，用来在NFA中计算状态转换的多种情况；闭包和状态转换的状态集版本，用来对NFA非确定性产生的状态集计算状态转换。
4. 理解子集构造法的思想，关键点：NFA🡪DFA最基本要求——两者等价，即两者接受相同的符号串集合，更具体的，任何符号串，输入NFA与输入DFA得到相同的结果。而输入NFA得到的是一个状态集，输入DFA得到一个状态，这两者等价。因此算法总体思路是通过NFA识别符号串得到状态子集来构造DFA状态。
5. 理解NFA的模拟算法，每处理一个输入符号对状态集进行状态迁移和闭包——可看作在进行子集构造法，假想有等价的DFA，动态构造此次词法分析涉及的DFA状态——Lazy方法。
6. 理解Lex工具的实现，正则表达式🡪NFA🡪DFA，DFA模拟程序即词法分析器，理解最长前缀原则。

**重点&难点：“子集构造”具体如何进行？穷举所有符号串进行子集构造是不可行的——由简单到复杂的顺序枚举符号串输入NFA，得到状态子集构造DFA状态。首先是长度为0的符号串，输入DFA得到初态，输入NFA得到初态的闭包T——计算它就得到DFA初态A；然后求A对每个终结符a的边的目标状态(A, a)（长度为1的符号串）——计算\_closure((T, a))！不断重复此过程（将更长的符号串输入NFA和DFA），构造出DFA更多状态和边，直到收敛。**

1. DFA最小化

前面已经解决了词法分析器构造行不行的问题，进一步考虑好不好的问题。

1. 理解区分的概念，首先搞清谁区分谁——一个符号串w区分两个状态s、t，所有符号串都不可区分的状态——词法分析中作用相同——可合并。
2. 理解算法思想，不是进行合并，而是反过来进行分裂，因此初始时是从状态全集开始，如果某个集合中状态可被符号串区分，则分裂，直到不可再分裂为止，最终每个状态集合并为单一状态。

**重点&难点：弄清区分概念，从两个状态开始，对符号串进行识别，不是说最终得到两个不同状态就是区分，而是要一终态、一非终态才是区分。**

**理解分裂方法，类似子集构造法，我们不可能穷举所有符号串来区分状态，而是按顺序枚举符号串。首先还是，它能区分的只有终态与非终态（识别的效果是原地不动），因此初始时状态全集分裂为终态集和非终态集。然后对每个状态集对每个符号进行状态迁移（利用更长的符号串进行区分），如果结果状态落在多个状态集中，就将原状态集相应拆分，直至所有集合不可拆分为止。**