程序=数据结构+逻辑结构

存储结构：线性，链接，哈希

制导翻译

正确判断：语法（合法），语义（合理）

词法lexical，语法，句法syntax，语义semantic

源程序（词法，句法分析）-正确程序（句法制导翻译）-中间代码-

链接：静态（运行前，包含所需库），动态（运行时，链接使用函数）

优化：源程序，数据结构，汇编级

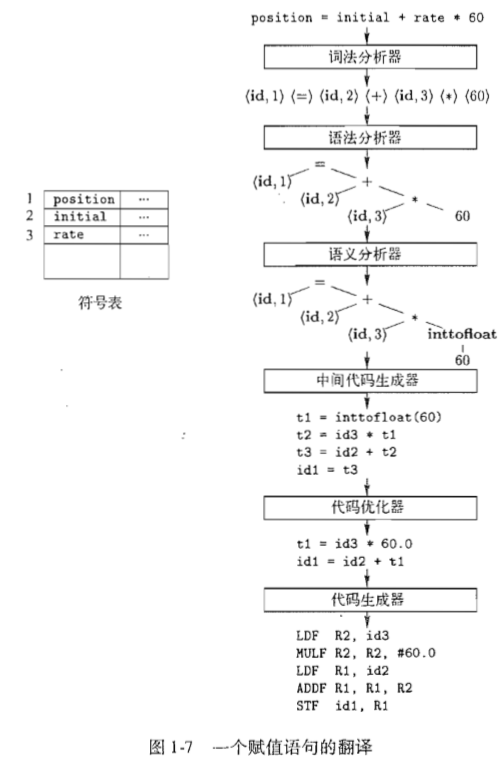
编译器：附带错误信息报告的将源程序变为目标程序的程序（正确性判定，翻译）

Syntax树：算数成分为叶子，运算符为父节点 Parse树：源成分为叶子节点

错误类型：语法，语义（空指针错误）

程序 外部特征：输入、输出 内部特征：数据结构、算法逻辑

词法分析器parse：上下文无关文法（输入）、词法分析程序



语言语法表示

语言结构：自下而上，自上而下

最小成分：普通、转义字符

基本成分：单词、字符序列（空串：零字符字符串）

完整成分：句子、单词序列

笛卡尔：

单词集合：、（以为核心的闭包、正闭包）

句子集合：

文法grammar：

描述语法对象结构的无二义的**产生式**规则

Formal production rules describing the construction of syntax elements.

关联：条件-动作、原因-结果

文法概念：核心、支撑、原子

语法树syntax：语法分析的结果

Parse树：语法分析的细化

开始符：树的根节点，文法核心概念（通过产生式集合P形成产生式）

非终结符：产生式左部，需继续细化（非终结符集合、尖括号/大写字母）

终结符：产生树的叶节点，为原子结构/概念（终结符集合、小写字母）

文法描述且仅描述唯一核心概念（开始符），此概念的支撑概念形成产生树，无法细分的原子概念为终结符

无用产生式：

1. 同义反复（直接/间接环）
2. 与无用非终结符（与开始符无关、使用关联树检测）相关联
3. 无法推出终结符序列

好的文法：删除无用产生式、删除产生式（提高推导效率）

产生式：直接、间接（推导）产生式（使用**迭代**方式，找到稳定解）

**产生式优化**：设计包含所有直接产生、找右部包含中元素的非终结符使用替代到稳定、扩展产生式、删除产生式（非开始符）、存在引入新开始符

判定实例合法：

最左、最右、混合

推导Derivation：从**开始符**开始，对非终结符找相关联产生式，使用右部代替左部

预测分析Predictive：对子串预处理，判断其包含终结符情况，提高推导效率

归约Reduction：从**句子实例**开始，对终结符找相关联产生式，使用左替代右

句型：开始符推导的终结/非终结符序列

随机推导/归约：轮盘赌方式（根据概论选择句型）

**最左**（右）归约是最右（左）推导的逆过程（横线推导、双竖线归约）

集合表示：枚举、解析（模板与相应条件）

递归：终止条件、分解方法、推出无限

**文法类型**：

0型/无限制：产生式左部存在非终结符（图灵机）

1型/长度增加：产生式右部长度左部长度，除、**上下文有关**文法/描述语义（线性自动机）

2型：产生式左部非终结符必推出右部终结/非终结符序列、**上下文无关**/描述语法结构（分层/带栈/下堆有限自动机PDA、函数嵌套形成多层自动机）

嵌入式文法：右部非终结符嵌在两终结符中间

3型：产生式右部为（1终结符+1非终结符）/ 1终结符**（向）右/左线性正规文法**

给出分析树、寻找模板

**对称法**：找对称轴、对称特性、证实结果、嵌入式文法

**逐步求精**（分解部分独立）：自顶向下、自左向右等（将右部视为整体递归）

**等价法**：产生式两边具有相同特性

**电路状态转换图**：关心**奇偶状态**的项作为节点、绘制电路转换图、确定开始/结束状态、为每个节点命名并写产生式（终止处）

题目要求做初态全0为终态经过的路径

随机选择：选择终态、与全1异或得到初态

产生式个数：

**Hybrid方法**：要求开始/终止为特定字符、设置开始/终止边并与电路图链接

**有限自动机FA**：关心二进制转十进制整除问题、绘制有限自动机、包含左/右线性描述

二义性：一个句子有多种产生树

词法分析lexical analysis：

Token：单词、类别、内部码（数据关联/弱）

指向符号表中位置的指针设为内部码

正规表达式RE：、单词（组合）模板

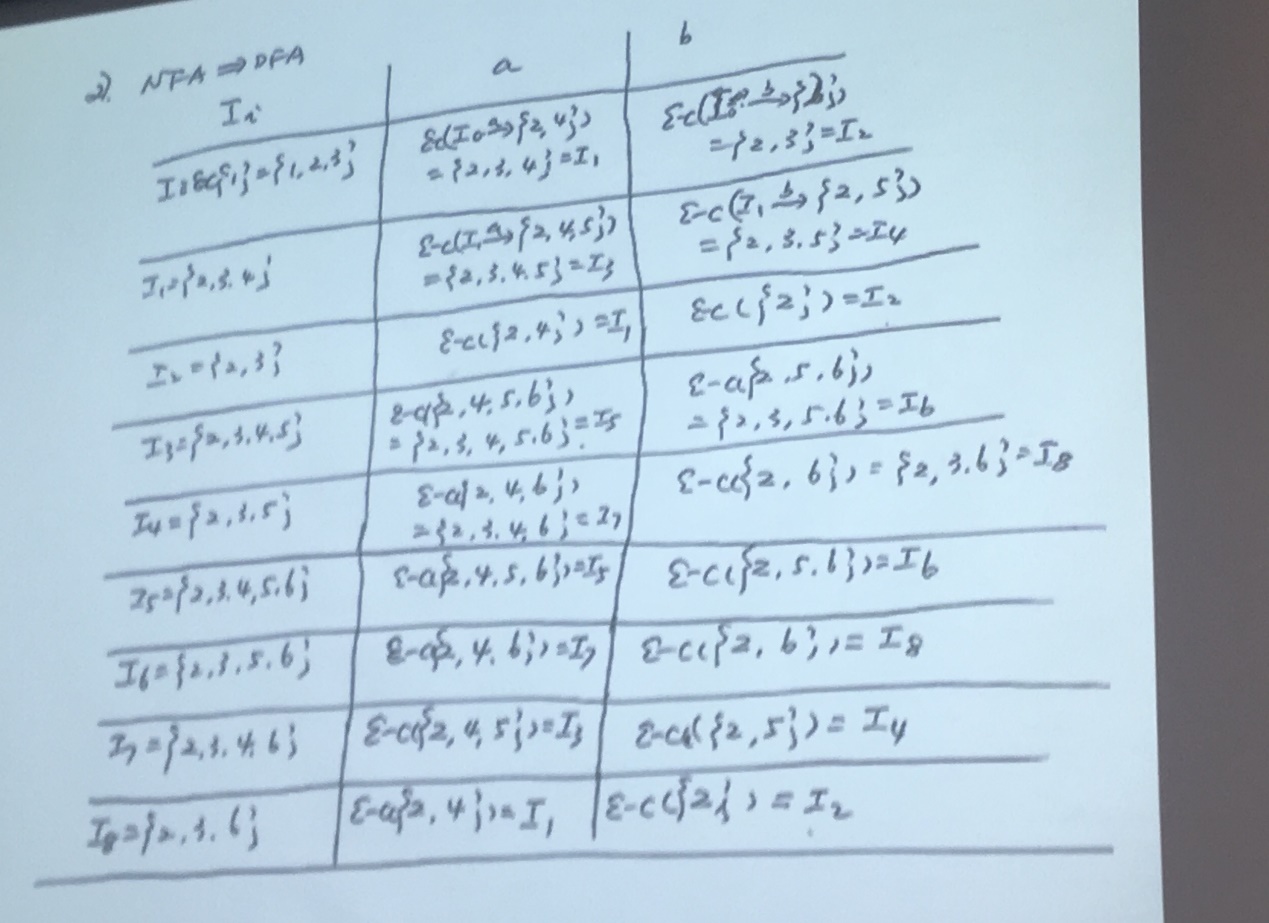
**FA**：（S：状态集合、S0：开始状态、F：终态、：标记、move：边）

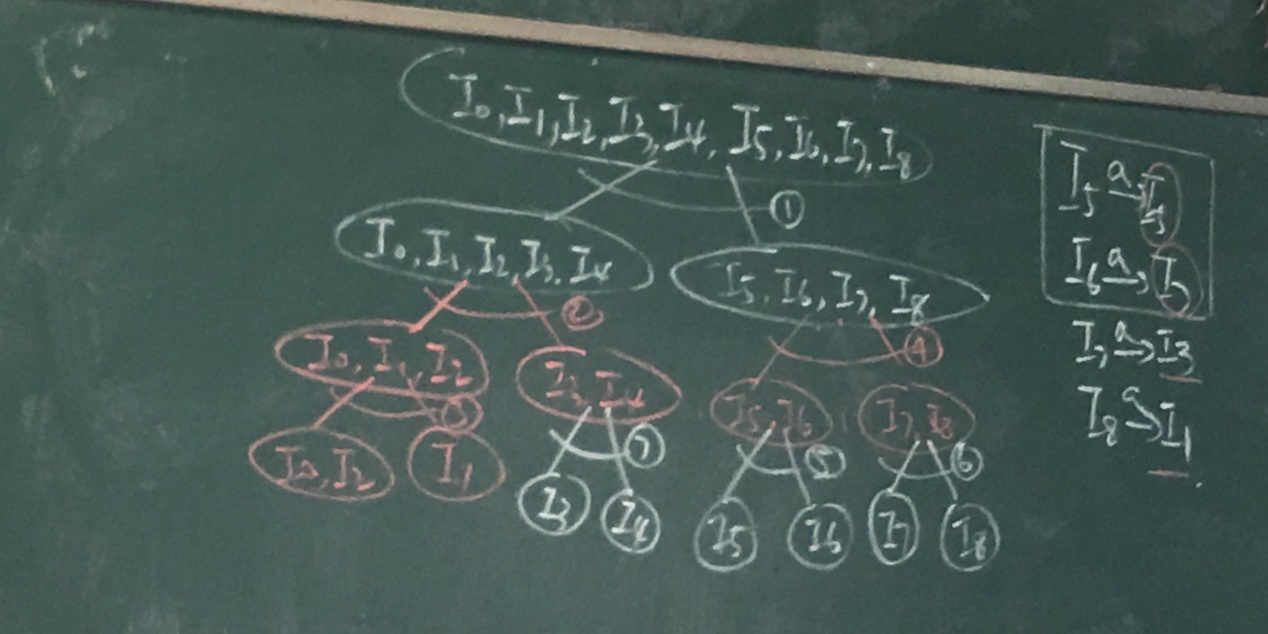
DFA：不允许边、至多允许唯一后续状态

NFA：允许边、允许多后续状态

NFA转DFA：闭包、子集构造、表格驱动、**终态**与原终态交集非空（状态数：）

ECD：聚类（clustering自下而上）、分类（自上而下、多次二分类解决多分类）





状态等价：强（后续标记相同、递归强等价）、弱（相同标记边到达所属叶子节点相同）Open/Close表：工作用表（将做处理状态）、记忆用表（已做处理状态）

Thomson算法：基本自动机、利用边运算符组合（加点、中转后缀、链接各自动机）

最小化DFA：终态/非终态、逐渐分解、直至叶子节点状态**弱**等价（注意回溯），原DFA初终态为代表

右线性文法-NFA：状态数=非终结符数+1（状态间通过产生式边连接）

RE正规表达式-DFA：逐步求精、基于FA、基于RE结构树

程序：硬编码（FA写入程序）、软编码（提供事件驱动表）

实验中：化简DFA时化简非终态，所有终态分开为不同叶子节点（防止歧义）

语法分析Syntax Analysis：

依据CFG上下文无关文法进行语法分析

**递归下降分析**：存在公共左因子试探（引起回溯）、左递归（无限环）

**预测分析法**：预处理（提取**最大公共左因子**、消除左递归）、构造预测分析表

**消除左递归**：顺着使用圈设置优先级，低优先级者代换（间接左递归）、变为右递归（直接）、混合左递归（由内而外，从左而右）

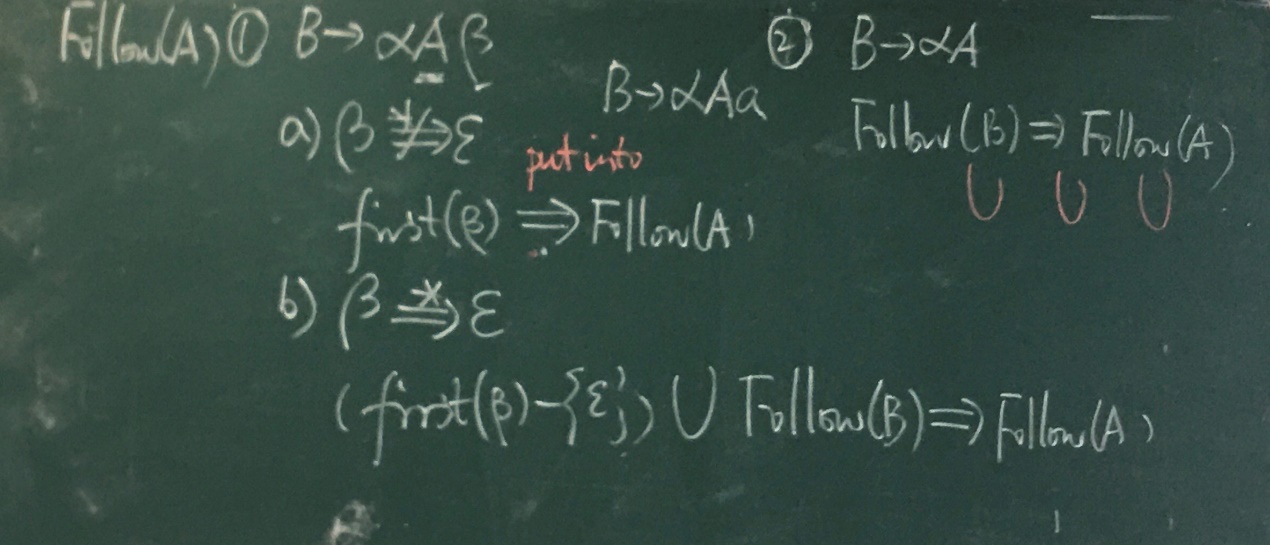
预测分析表PPT：栈顶出现非终结符y，读头下的符号x（PPT[y, x]相应产生式推导），从右向左压栈，栈顶为终结符时匹配读头，构建语法分析树（二义性: 格中元素个数>1）

**PPT构造**：First填非产生式，Follow填产生式

Follow(A)：句型中在A后的终结符，寻找相关产生式（右部出现A）、解决产生式应用

**LL(1)**（左线性，Leftmost，读头一个）：无左递归/公共左因子（multiple entries）

FA变为RE：消除中间态、观察每一个终态



数据结构：数据带、PPT、语法栈

程序步骤：初始化（栈，读头）、查表、匹配Matching

通过CFG构造**分层FA**：构造最上层小的FA、通过边连接相关FA（NFA）、新引入开始符S

LR item命名：层次（产生式）、位次（dot 表示分析到的位置 可规约、已经项目）

状态间扩展：分层放入、。后非终结符扩展 状态内扩展：移点（闭包）构造LR DFA

栈：状态栈+符号栈

SR冲突：first(移进)项存在于follow(规约)

RR冲突：follow(规约)交集不为空

PPT：ACTION+GOTO、非终结符+终结符S、可规约项填为R

SLR(1)：无法解决二义文法SR、RR冲突（shift-reduction）

LR(1) item：LR item+预测符（可规约串后将出现符号）

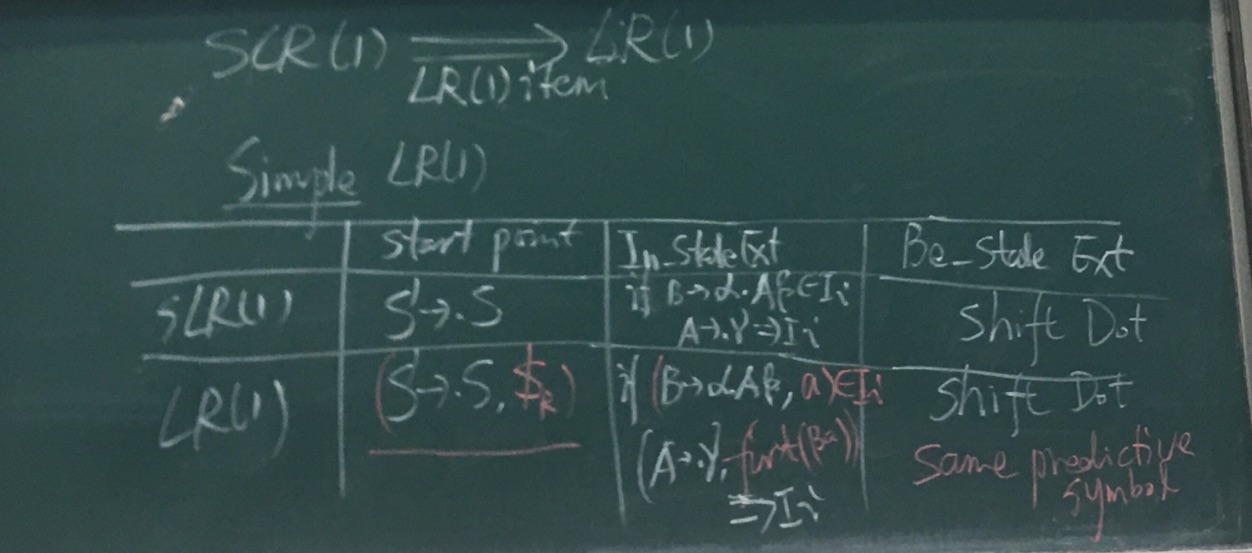
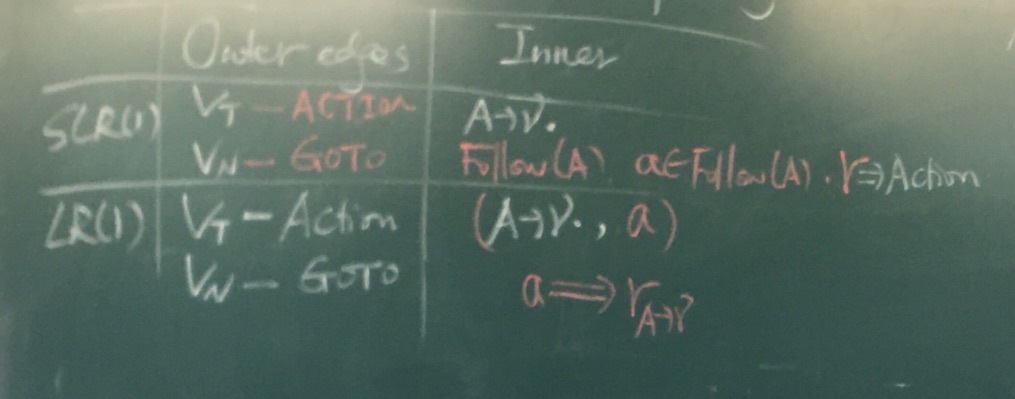
SLR(1)均为LR(1)文法：预测符属于FOLLOW集

状态间扩展：判断预测符 状态内扩展：移点（预测符不变）

SR冲突：可规约项目预测符**包含**移进项目点后**终结符**

RR冲突：可规约项预测符交集不为空

栈与分析表项数目增加，MORE STATES

LALR(1)：合并LR(1)同芯状态、后与SLR状态数同（芯：状态内LR item集合）

合并时可能出现RR冲突

判断一个文法，构造SLR/LR/LALR分析表，无冲突则为相应文法

规定优先级及结合规则（附加条件），解决歧义性冲突（删除相应S/R）

第一个产生式左部为开始符、越靠近开始符的运算符优先级越低

语义分析Syntax-Directed

语义规则：符号串的属性（名，值）、特征

结合规则：语法制导定义，规则放**在产生式外右侧列出**，规约时计算（SDD）

转换规则，分部存放，规则**嵌在产生式中**（TS）

继承属性：从兄长、父节点从左继承（不允许出现在SDD中）

加标注分析树：将符号加上属性名、自下而上归约计算（语法制导）

分析后加标注：构造分析树、属性名指派、属性值计算（自上而下）

类型检查Type Checking

静态检查：运行之前（类型、控制流while、一致性、命名）

数据类型：指示数据取值范围（基本、组合）

类型表达式：int a[10]（a:array(0~9, int))；int a(int, int)（a:intXint->int）

运行时环境Runtime Environment

运行时检查：数据存储空间分配

自顶向下：逻辑/数据结构相似性，递归

Procedure：A(a: integer, Var b:float)

Function：f(a: integer):integer

变量：全局（static仅此文件有效），局部（static同一函数多次执行间通信，一次初始化），块内（仅在函数块中有效for）

Caller-Callee关联参数：返回位置，参数个数，参数，返回值

**Max Stack map**结合树：根为由OS调用的main函数，显示调用关系

总体布局stack layout：结合树树根到叶子最长路径

程序状态：记录数据对象的集合（静态，动态：栈、堆）

活动记录AR：关联、Callee数据

C活动记录：[Caller **S**tack **P**ointer、函数返回地址、参数个数、参数(形名, 实值)]、[数据(局部, 内部临时)、函数返回值]caller-callee间关系数据+callee自身内部数据

回调函数Call-back function：函数参数为函数

符号表：全局（全局变量、函数名），局部（符号表per函数，分层命名）

中间代码生成Intermediate Code Generation

语法制导：将语法制导翻译以code中间代码形式展现

中间代码表示：图（分析树，语法树，有向无环图），表达式

表达式：中缀C（**三地址码** 两操作数+结果）、前缀 汇编（**四元式** 操作码、两操作数、结果），后缀

TAC：if(x<>y) goto L (j<>, x, y,<L>)

简单量表达式：逻辑常量、变量、相关表达式、包含**简单数据类型**的表达式（非数组指针）

变址访问：MOV AX[BX], CX，计算地址偏移量

C变为TAC：使用SDD，增加.code属性（三地址码语句）

Row-major按行优先：address(a[I, j])=addrA+[(i-l1)\*(u2-l2+1)+(j-l2)]\*w=

addrA-(l1\*n+l2)\*w+(i\*n+j)\*w

**短路代码**：复杂条件表达式由部分得知全体

**标号回填**Back patching：语句序号、转移目标不确定(goto 0)、记住包含goto 0、并链转移目标相同、转移目标确定回填（改写文法）

E.TC/FC：条件为真/假含goto0语句序号

C.chain/S.chain：记住需回填的语句序号

并链：转移语句相同 回填：转移语句可知