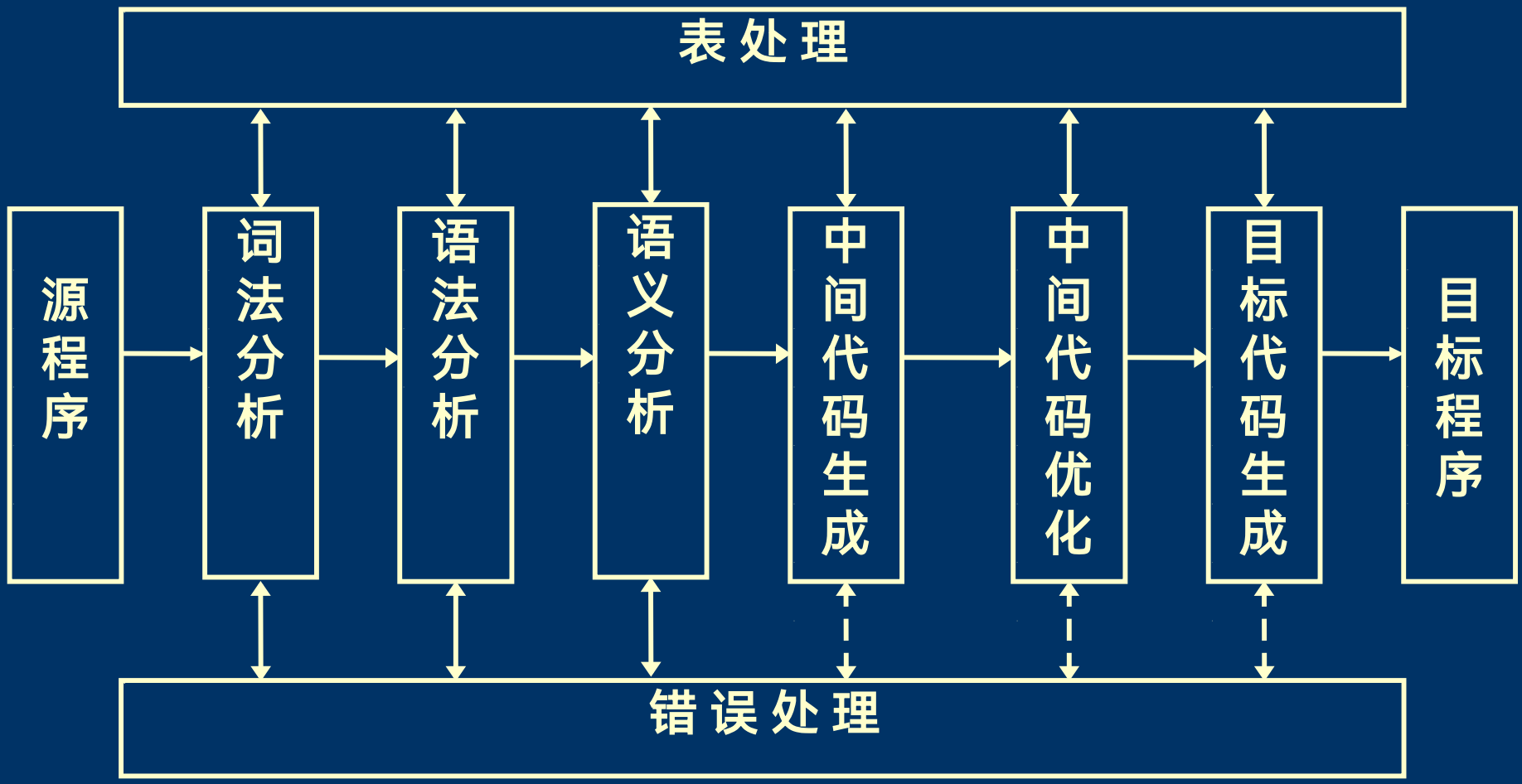
# 总 结

试卷类型：
选择5题 每题2分
简答计算等10题 每题5分
大题4题

题目分布（按书中章节）：
2、 25分
4-5、30分
6、 10分
7-8、15分
9、 5分
10、 5分
剩余10分出卷老师自行分配

# 绪论

- 编译器 、解释器
- 编译过程 、各过程的功能
- 遍数


表处理
源程序
词法分析
语法分析
语义分析
中间代码生成
中间代码优化
目标代码生成
目标程序
错误处理

# 词法分析

- 一个核心：词法分析器的设计
- 两个工具：正则表达式和自动机
- 三个转换算法：NFA到DFA，自动机的极小化，正则表达式和自动机的互相转换

单词：是指语言中具有独立含义的最小的语义单位。
正则表达式缺乏对称性字符串的表达能力

例如：设字母表Σ={0，1}，求二进制数字集合且为2的倍数。
所有Σ上定义的串的正则表达式为（1|0）*
则二进制数表示为1（1|0）*|0
其中能被二整除的表示为1（1|0）*0|0

确定有限自动机DFA为一个五元组$(\sum,S,S_0,f,Z)$

对于两个DFA M1和M2，若有L(M1)=L(M2)则称M1和M2等价

例如：用自动机描述被3整除的数。

NFA和DFA的区别

一个状态的不同输出边可以标有相同符号
允许有多个开始状态
允许有空边

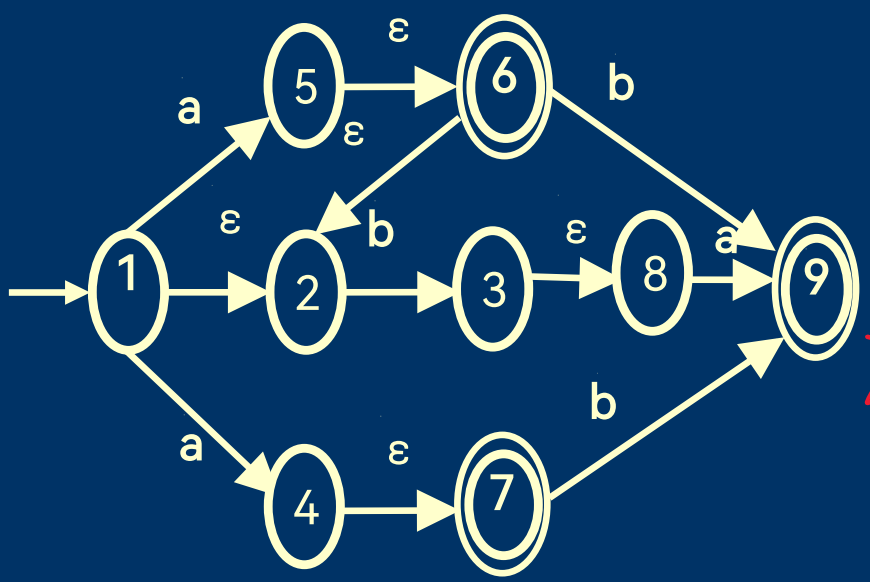
## NFA到DFA转化：

已知 A： NFA, 构造 A’:DFA

令A’的初始状态为$I_0$’=ε_CLOSURE({$S_1$,$S_2$,…$S_k$}),其中$S_1$…$S_k$是A的全部初始状态。

若I={$S_1$,…,$S_m$}是A’的一个状态，a∈∑，则定义f’(I, a)=Ia将Ia加入S’，重复该过程，直到S’不产生新状态。

若I’={$S_1$,…,$S_n$}是A’的一个状态,且存在一个$S_i$是A的终止状态，则令I’为A’的终止状态。




| | a | b |
|---|---|---|
| +{1,2} | {2,4,5,6,7} | {3,8} |
| -{2,4,5,6,7} | {} | {3,8,9} |
| {3,8} | {9} | {} |
| -{3,8,9} | {9} | {} |
| -{9} | {} | {} |

## 定义：等价状态

设DFA M 的两个状态$S_1$和$S_2$，如果对任意输入的符号串x，从$S_1$和$S_2$出发，总是同时到达接受状态或拒绝状态中，则称$S_1$和$S_2$是等价的。

## 定义：最小(最简)自动机

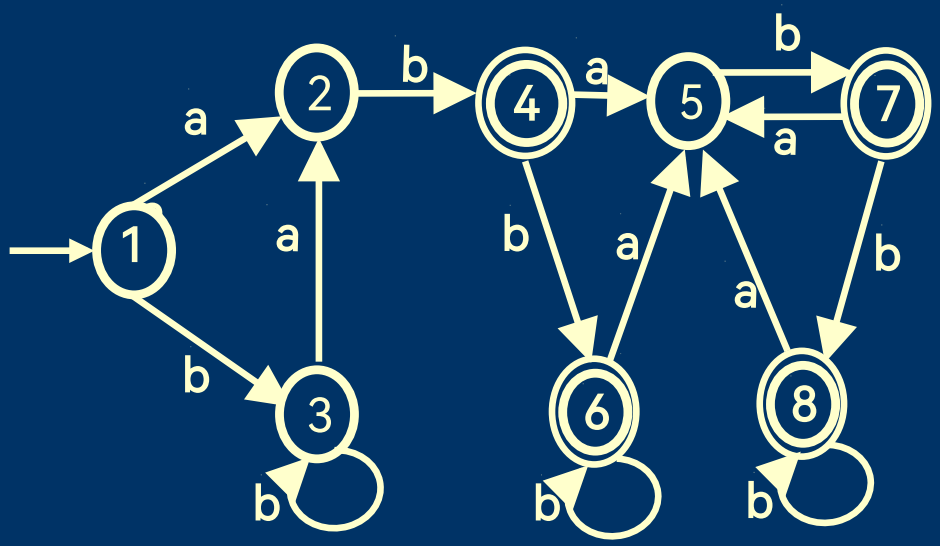
如果DFA M 没有无关状态，也没有等价状态，则称M 为最小自动机。

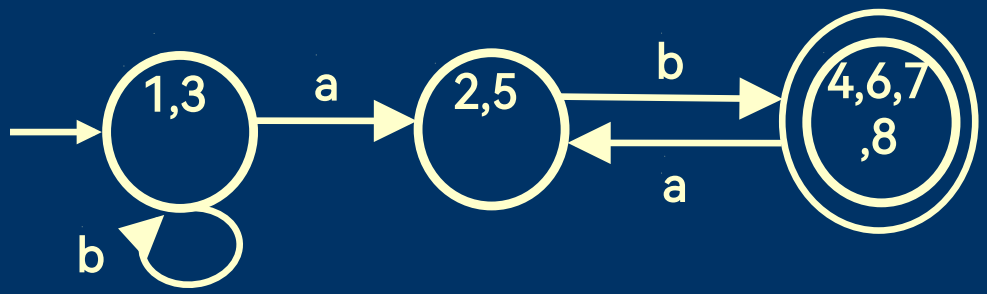
## 状态分离法

终止状态为一组，非终止状态为一组

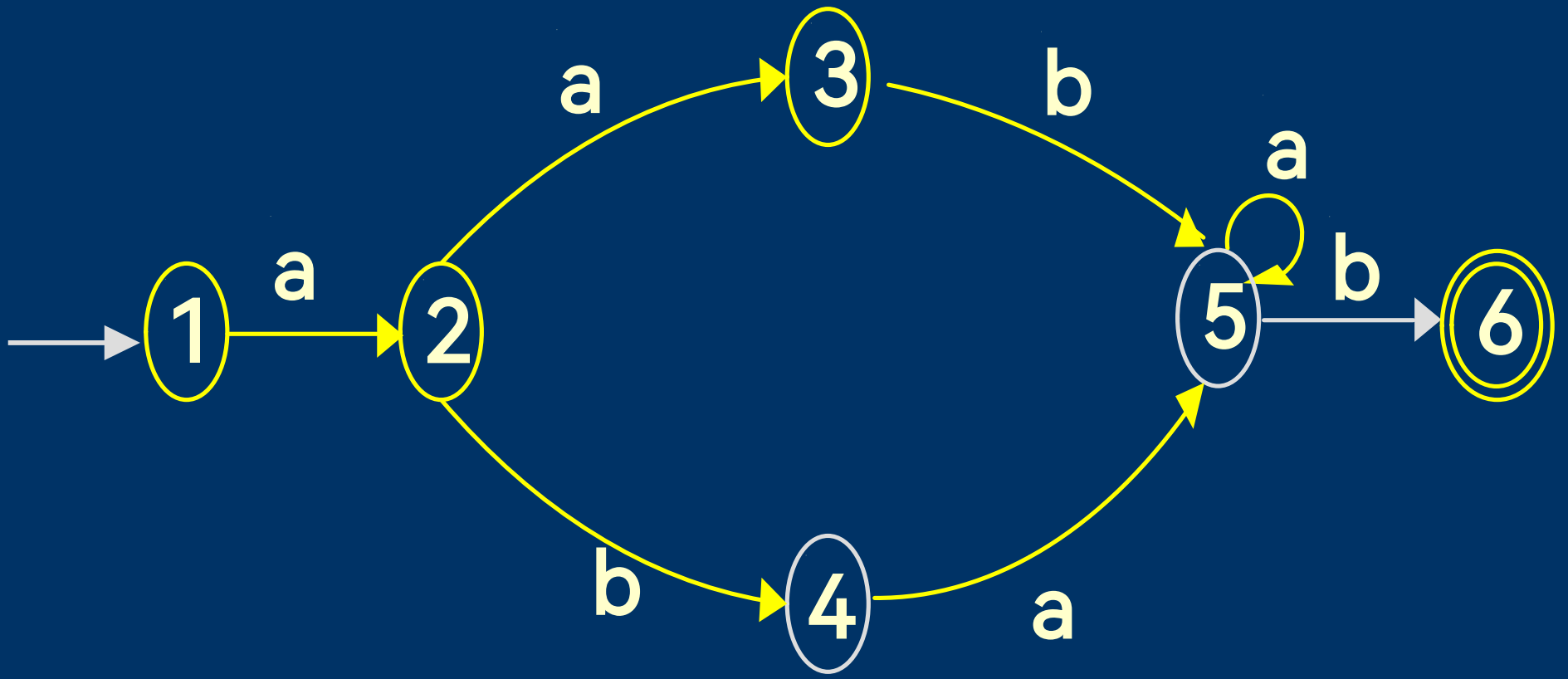
对每一组进行分离，若每组中的元素映射到不同的组，则表示他们不等价，就可以划分出来。

重复2，知道没有新组产生，此时每组中的状态都为等价状态。

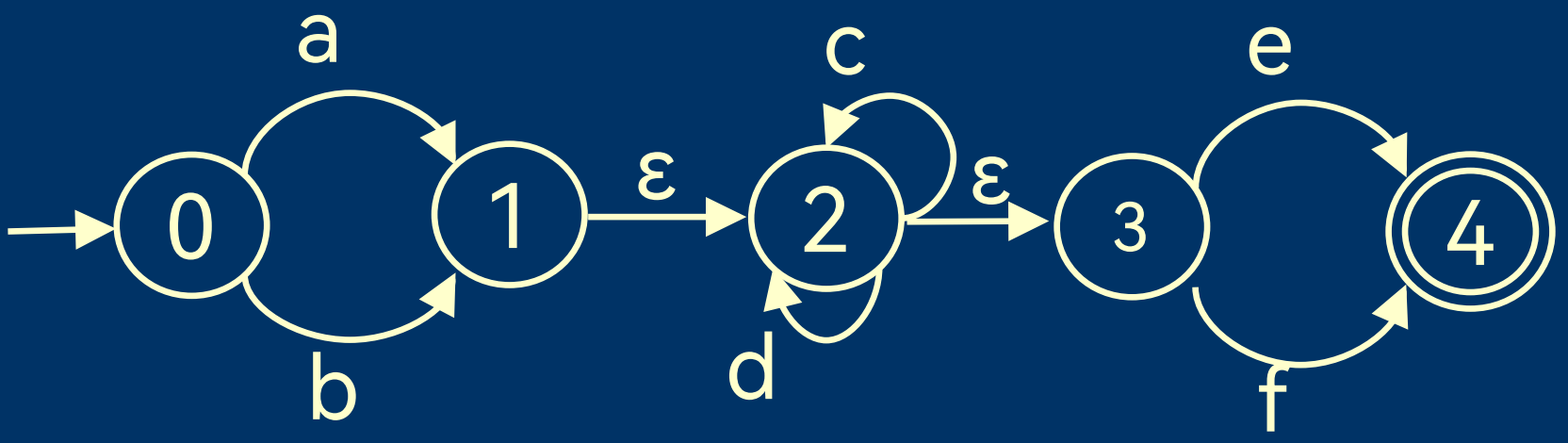


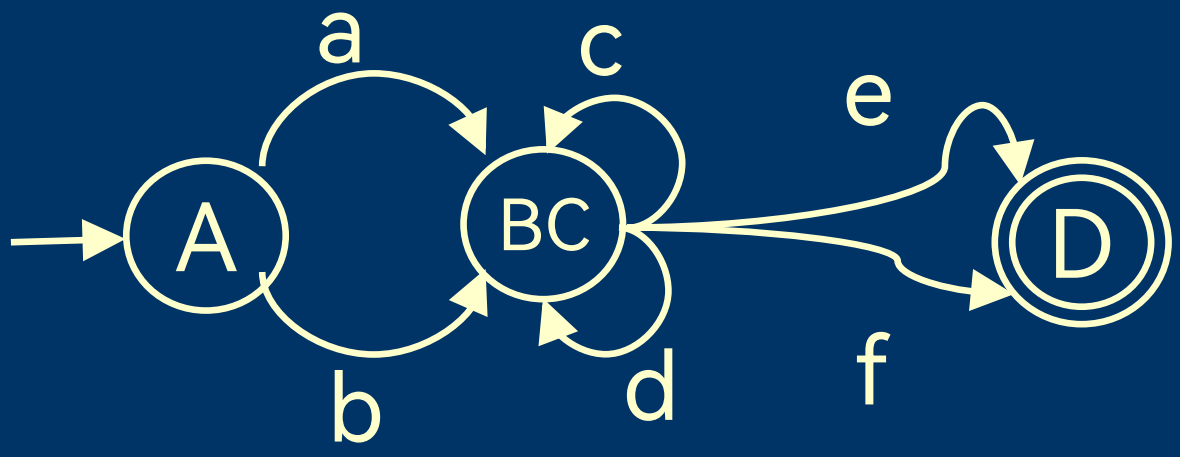



# DFA到正则表达式




- 给出一个正则表达式(a|b)(c|d)*(e|f)



关于Token的结构没有统一的规定，但至少包括两部分内容：

单词的类型（词法信息）

单词的内容（语法信息）

| 类型 | 内容 |
| --- | --- |

| 标识符索引表 | |
| --- | --- |
| 1 | |
| 2 | |
| 3 | |
| … | … |
| n | |

| 常量索引表 | |
| --- | --- |
| 1 | |
| 2 | |
| 3 | |
| … | … |
| n | |

| 保留字、特殊符号表 | |
| --- | --- |
| 3 | while |
| 4 | if |
| 5 | % |
| … | … |
| 56 | } |

# 语法分析

基本概念：

- 文法、文法分类
- 上下文无关文法
- 语法分析树、二义性、推导、归约、短语简单短语、句柄。
- 文法分析：First、Follow、Predict

语法分析方法：

- 自顶向下
- 自底向上

句型：设有文法G，如果有$S \Rightarrow^* \beta$，则称符号串β为G的句型。我们用SF(G)表示文法G的所有句型的集合

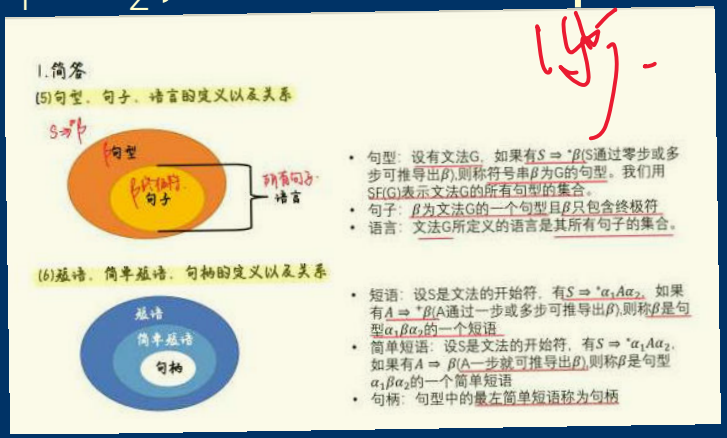
句子：设β为文法G的一个句型，且β只包含终极符，则称β为G的句子

语言：$L(G)=\{ u \mid S \Rightarrow^+ u, u \in V_T^* \}$。文法G所定义的语言是其所有句子的集合

短语：设S是文法的开始符，有$S \Rightarrow^* \alpha_1 A \alpha_2$，如果有$A \Rightarrow^+ \beta$ 则称β是句型$\alpha_1\beta\alpha_2$的一个短语。

简单短语：设S是文法的开始符，有$S \Rightarrow^* \alpha_1 A \alpha_2$，如果有$A \Rightarrow \beta$ 则称β是句型$\alpha_1\beta\alpha_2$的一个简单短语。

句柄：句型中的最左简单短语称为句柄。




最左（右）推导

左（右）句型

每个句子都有相应的最右和最左推导（但对句型此结论不成立）

语法树 文法的二义性判定

文法G定义为四元组($V_T$,$V_N$,S,P)

0型文法: 也称为短语文法，其产生式具有形式：$\alpha \rightarrow \beta$

1型文法: 也称为上下文有关文法。$\alpha A_1 \beta \rightarrow \alpha A_2 \beta$

2型文法: 也称为上下文无关文法。$A \rightarrow \beta$

3型文法: 也称为正则文法。$A \rightarrow a$ ， $A \rightarrow aB$

描述能力：0>1>2>3

处理难度：0>1>2>3

正则文法到DFA构造方法:

令$S=V_N \cup \{k\}$； $\sum=V_T$ ； $S_0=_Z$; $F=\{k\}$;

转换函数f 如果有$X \rightarrow aY$ 则构建$f(X,a)=Y$

如果有$X \rightarrow a$ 则构建$f(X,a)=k$ k是终止状态

DFA到正则文法构造方法:

令$Z'=\{A|A \in Z$，A没有输出边$\}$; $S=S_0$; $V_N=S-Z'$; $V_T=\sum$;

规则P:

若有$f(X,a)=Y$，且$Y \notin Z'$，则构建$X \rightarrow aY$;

若有$f(X,a)=Y$，且$Y \in Z$ ，则构建$X \rightarrow a$;

# 自顶向下分析方法

- 思想
- 关键问题
- 两种分析方法：[条件、分析过程]
  - 递归下降方法
  - LL（1）分析方法
- 等价变换：消除左递归、左公共前缀

提取公共前缀：$A\to\delta\beta_1|\delta\beta_2|\dots|\delta\beta_n|\gamma_1|\gamma_2|\dots|\gamma_m$

$A\to\delta A'|\gamma_1|\gamma_2|\dots|\gamma_m$ $A'\to\beta_1|\beta_2|\dots|\beta_n$

**消除直接左递归**：$A\to A\alpha|\beta$ 即有$A\Rightarrow\beta\alpha^*$

$A\to\beta A'$ $A'\to\alpha A'|\varepsilon$

First集是对一个串来定义的

Follow集是对一个非终极符定义的，并且递归求解得出的

Predict集也叫预测符集，代表用当前规则进行推倒可能推出的首个终极符

LL(1)文法定义：

对于文法G中任一非终极符A，其任意两个产生式$A\to\alpha$和$A\to\beta$，都要满足下面条件：

$$Predict(A\to\alpha)\cap Predict(A\to\beta)=\varnothing$$

满足这一条件的文法称为LL(1)文法。

**分析栈 输入流 LL(1)分析表 驱动程序(替换、匹配、成功、出错)**

递归下降法是对$V_N$构建函数，照着规则和Predict集写就可以

# 自底向上分析方法

- 思想
- 关键问题
- 分析方法:

  LR(0)、 SLR(1)、 LR(1)、LALR(1)、简单优先分析方法
- 比较:

  状态数、展望符、分析能力、应用

## 增加拓广产生式

自底向上的语法分析就是一个找句柄的过程

简单优先方法：引入≅、⋖、⋗三种优先关系

X ≅ Y： 当且仅当存在一个产生式A→…XY…

X ⋖Y： 当且仅当存在一个产生式A→…XB…并有B ⇒ +Y…

X ⋗Y： 当且仅当存在一个产生式A→…BC…并有B ⇒ +…X, C ⇒ *Y…

特别假设对输入流结束标志'#'有：X ⋗#；对符号栈栈底标志'#'有：# ⋖X；其中X为文法中任意符号

## LR方法总体思想

**符号栈** #α 有终极符也有非终极符

**输入流** β# 终极符

假如要分析的串没有语法错误，则αβ一定是文法的一个句型

LR方法的主要思想是，从输入流依此把符号移入符号栈，直至栈顶出现一个句柄；之后对句柄进行归约，直至栈顶不出现句柄；重复上述过程，直至最终归约为一个开始符，且输入流为空。

规范句型：用最右推导导出的句型

规范前缀：若存在规范句型αβ，且β是终极符串或空串，则称α为规范前缀。

可归前缀：若α是含句柄的规范前缀，且句柄在α的最右端

派生定理：

开始符产生式的右部是可归前缀。

如果$\alpha_1 A \alpha_2$是可归前缀，且$A \to \beta$是产生式，则$\alpha_1\beta$也是可归前缀。

找句柄的过程就是找可归前缀的过程，找到可归前缀就对其句柄进行规约

LR(0)项目是带一个圆点'•'的产生式。每个项目都标志分析时的某一状态。

项目集的投影：假设IS是LR(0)项目集，则称下面$IS_{(X)}$ 为IS关于X的投影集：$IS_{(X)} = \{A \to \alpha X \bullet \beta \mid A \to \alpha \bullet X\beta \in IS, X \in (V_T \cup V_N)\}$

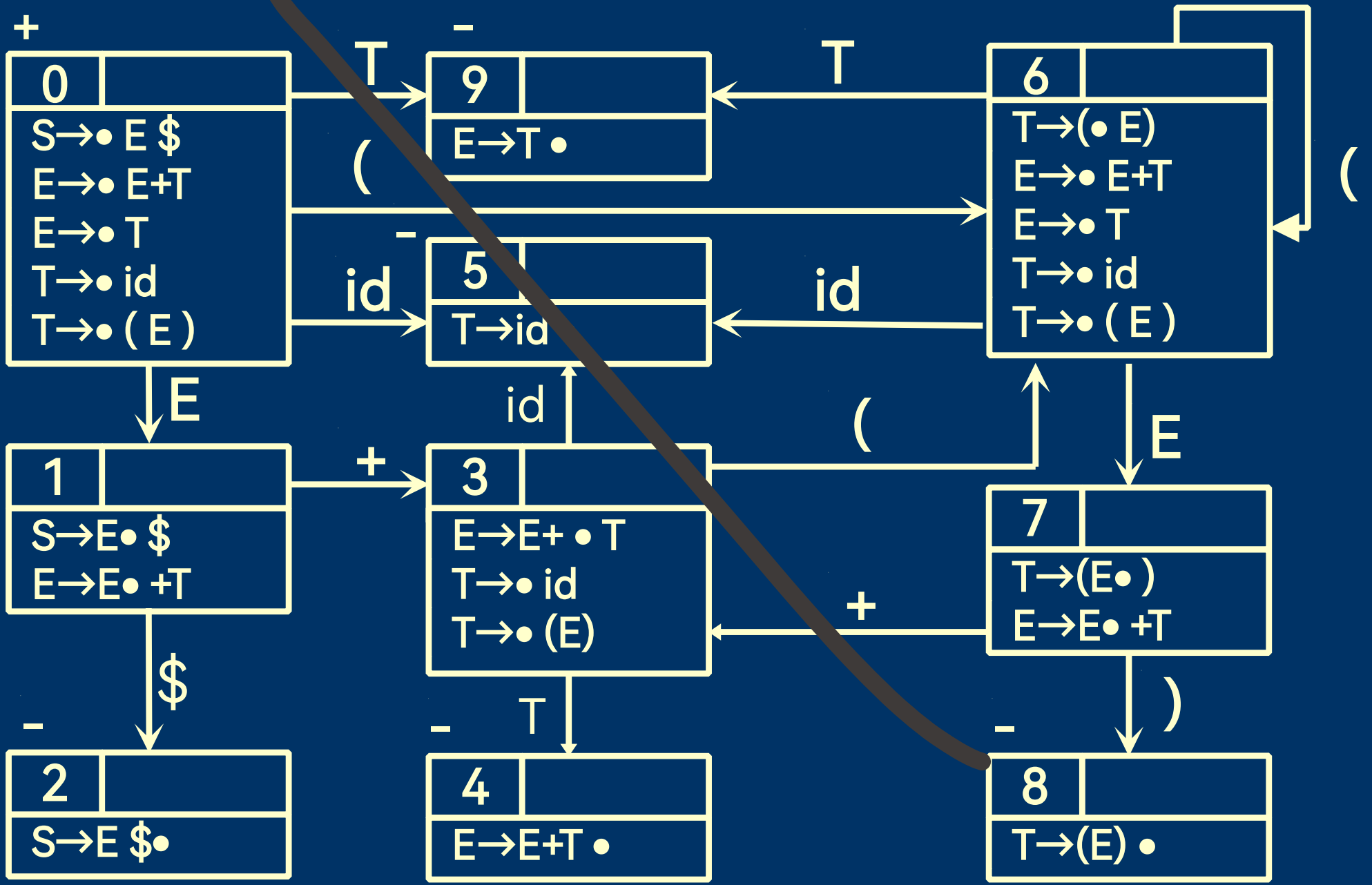
项目集的闭包：假设IS是LR(0)项目集，则称下面CLOSURE(IS)为IS的闭包集：

$CLOSURE(IS) = IS \cup \{A \to \bullet\pi \mid Y \to \beta \bullet A\eta \in CLOSURE(IS)\ \ A \to \pi$是产生式 $\}$

设IS为LR(0)的项目集，X是语法符号，则定义GO(IS, X)=$CLOSURE(IS_{(X)})$

设有文法G(S):

S → E $　　E → E + T　　E → T　T → id|( E )



## LR方法包含的主要部件

两个栈：状态栈 (可归前缀自动机)　符号栈

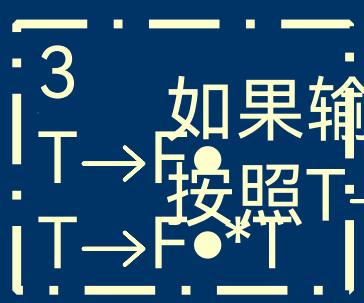
输入串

根据LR分析表：

分析动作(action)【移入(输入流移入)、规约、成功、失败】

转向状态(goto)【规约后状态的变化】

LR(0)不考虑输入串的头符，存在有大量的归约/移入、规约/规约冲突，没有充分的利用输入流信息，因此有了SLR(1)方法

对于一个状态 

如果输入流的头符是属于follow(T)就按照T→F进行规约

LR(0)只看分析栈的内容，不考虑当前输入符，SLR(1)考虑输入符，用follow集来解决冲突，因此SLR(1)要比LR(0)分析能力强。

$Z \rightarrow B^1aB^2bB^3c$

$B \rightarrow d$

SLR(1)归约时向前看一个符号，但是不区分语法符号的不同出现。上述文法中，B出现了三次，很显然$B^1$的后继符只能是a，$B^2$的后继符只能是b，$B^3$的后继符只能是c，而Follow(B)={a,b,c}，用SLR(1)就失去了精度，因此引出了LR(1)方法

LR(1)的基本思想是对非终极符的每个不同出现求其后继符，而不是给每个非终极符求其统一的后继符，我们称其为展望符集。

CLOSURE(IS)= IS∪{[A→•β,a]|

[B→$\alpha_1$•A$\alpha_2$,b]∈CLOSURE(IS),

A→β是产生式,a∈First($\alpha_2$b)} 重点！！！！！！！！！

初始项目集$IS_0$=CLOSURE(Z→•α,#)

核心的记忆点：展望符是什么？用当前规则进行规约，规约成一个非终极符，则可能出现在这个非终极符后面的第一个终极符是什么，用其与输入流进行对比，判断是否进行规约

LR(1)分析方法的可归状态机中包含的状态数过多，为了降低其状态数，以合并同心状态为基础，介绍了LALR(1)方法。

项目的心：假设[A→α•β, b]是LR(1)项目，则称其中的LR(0)项目部分A→α•β为该项目的心。

状态的心：设S是LR(1)状态机的一个状态，则S的所有项目心之和称为状态心，并表示为Core(S)。

同心状态：如果LR(1)状态机中的两个状态具有相同的心，则称它们为同心状态。

LALR(1)方法可能产生归约/归约冲突，可能造成延迟发现错误，不会产生移入/规约冲突。

LR(0)< S <LALR(1)

从功能上看，各种语法分析方法的分析能力从小到大依次为：

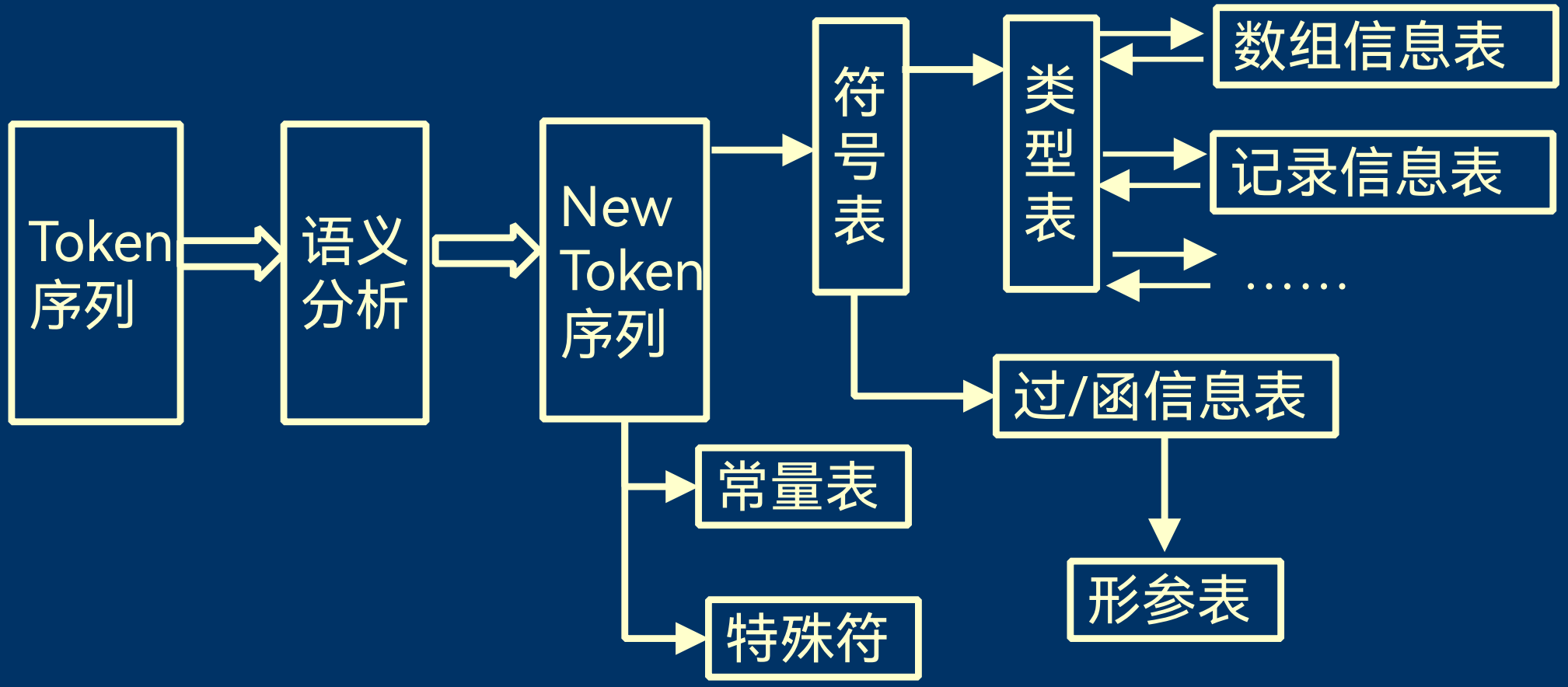
LR(0)<SLR(1)<LALR(1)<LR(1)

从状态数方面看，各种语法分析方法的状态数有如下关系：

LR(0)=SLR(1)=LALR(1)<LR(1)。

# 语义分析

- 语义检查的内容
- 标识符的语义表示
- 类型的语义表示
- 抽象地址
- 符号表的建立和管理




标识符：

常量标识符；

类型标识符；

变量标识符（实在变量，形参变量；值引用型；地址引用型）；

过函标识符（实在过函，形式过函）；

域名标识符；

符号表中必须包括：
种类信息 类型信息

常量标识符

| Name | Kind | TypePtr | Value |
|---|---|---|---|

类型标识符

| Name | Kind | TypePtr |
|---|---|---|

变量标识符

| Name | Kind | TypePtr | Access | Level | Off | Value |
|---|---|---|---|---|---|---|

域名标识符

| Nam | Typeptr | Kind | Off | HostType |
|---|---|---|---|---|

过程/函数标识符

| Nam | TypePtr | Kind | Level | off | Para | Class | Code | Size | Forwar |
|---|---|---|---|---|---|---|---|---|---|

类型可以分成下面几大类：
标准的类型：整形、实型、bool、字符类型， 这是标准的数据类型
自定义的数据类型，子界类型，枚举类型
结构数据类型：数组，集合，记录
特殊的指针类型，文件类型

| | Size | Kind |
|---|---|---|
| intPtr→ | IntSize | intTy |
| boolPtr→ | BoolSize | boolTy |
| charPtr→ | CharSize | charTy |
| realPtr→ | RealSize | realTy |

枚举类型

| Size | Kind | ElemList |
|---|---|---|

| Nam | Value |
|---|---|

结构体和联合体

| Size | Kind | RecBody |
|---|---|---|

| Name | Typeptr | Off | link |
|---|---|---|---|

指针类型

| Size | Kind | BaseType |
|---|---|---|

数组类型

| Size | Kind | Low | Up | ElemType |
|---|---|---|---|---|
| ArraySize | ArrayTy | | | TypePtr |

地址分配原则：静态分配、动态分配

| 层数 | 偏移 |
| --- | --- |

层数主要是针对嵌套式语言,表示的是某个函数所处的嵌套定义层数
偏移是针对过程活动记录的一个相对偏移量

符号表存储的是标识符的语义信息
声明性出现：
<标识符，a> 创建标识符a所对应的符号表，将token转化为<标识符，指针（符号表中创建的项）>
使用性出现：
<标识符，a> 到符号表中查找a，找到则将token转化为<标识符，指针（符号表中查找到的项）>
标识符的作用域：是指某标识符可以有效使用的范围，局部化区是允许含有声明的最小程序单位。

符号表中重要的检查：变量是否重复声明，标识符的使用有无声明

**一个是创建符号表，一个是管理符号表，属于同时进行的。**

**符号表局部化处理的本质：** 在程序的某一个点P上，判断符号表的哪些信息是有效的

**因此有以下原则：**

- 每进入一个局部化区，记录本层符号表的首地址
- 遇到声明性标识符时，构造其语义字，查本层的符号表，检查是否有重名，有则出错，否则就把其语义字填到符号表里。
- 遇到使用性出现，查符号表，如果查到则读取其语义字，否则出现语义错误。
- 退出一个局部化区，'作废'本层的符号表。

**主要有：删除法、驻留法**

# 中间代码的生成

- 中间代码生成的目的（优化、移植）
- 语法制导方法
- 中间代码生成：

  表达式的中间代码

  复杂变量的中间代码

  语句的中间代码

  过函声明的中间代码

语义信息的提取与保存:

四元式: (算符op，$ARG_1$，$ARG_2$，运算结果RESULT)

1、指向相应符号表的指针

2、把对应分量的语义信息放在此处

语义栈Sem及其操作：在语法制导生成中间代码的过程中，要用到一个语义栈，把相关的分析程序的一些中间结果都要存放到语义栈中。

表达式的中间代码生成

【1】E→T 空　【2】E→E+T #inc 【3】E→E-T #dec

【4】T→F 空　【5】T→T*F #MUL 【6】T→T/F # Dev

【7】F→(E) 空　【8】F→i#Push

V→id #push 将id压入语义栈 【标识符变量】

V→*V #pushA 取sem(s-1)，取其址压入【指针变量】

V→V.id #dom 【域选择变量】

下标变量：V→V[E] #Addnext
#Addnext中要执行的动作的思想：
1.取出E的值
2.-数组下界
3.*size
4.+初始地址

S → V:= E #Assig# (ASSIG，Right （t），-，Left )
函数调用的中间代码：
**(VarACT值参/ValACT变参,$t_1$,$Offset_1$,$Size_1$)……**
**(CALL , <f> ,true/false ,Result)**
goto L的中间代码(JMP,-,-,L)
标号L:（Label,-,-,L）
条件语句if E then s1 else s2
E的四元式;（then,E,-,-）; S1的四元式 (else,-,-,-);S2的四元式 (endif,-,-,-)

While语句的中间代码结构while （E） do S

（WHILE，—，—，—）

E 的中间代码

（DO，E . FORM，—，—）

S的中间代码

(ENDWHILE，—，—，—）

过程函数声明的中间代码

(ENTRY,Q,-,-)

（ENDPROC，—，—，—）或

（ENDFUNC，—，—，—）

# 中间代码优化

- 优化的目标、要求、对象
- 基本块的划分
- 优化方法：全局、局部
- 常表达式优化
- 公共表达式(局部)优化
- 循环不变表达式外提

基本块是指程序的一组顺序执行的语句序列，其中只有一个出口和一个入口。
入口：基本块的第一条语句；
出口：基本块的最后一条语句；
具体划分方法略

## 常表达式节省

常表达式：在编译过程中能够计算出常量值的表达式
处理思想：针对每个基本块，如果一个多元式的两个分量的值已知，则计算其值，并删掉相应的中间代码。
常量定值表ConstDef：(Var,Val)

## 公共表达式节省

值编码优化方法的主要思想:
　对中间代码中出现的每个分量（常量和变量）确定一个值编码，使得具有相同值的分量编码值相等.

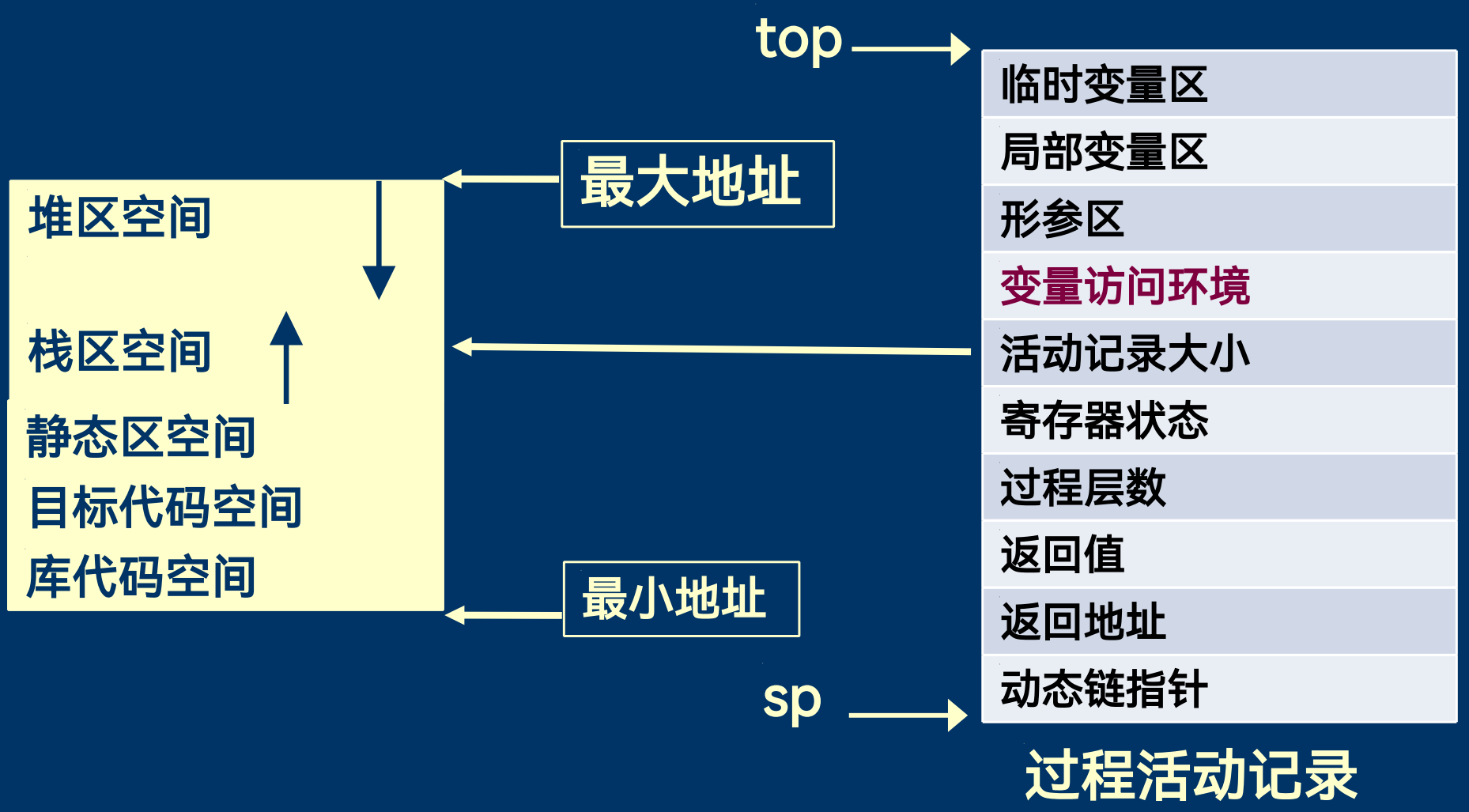
## 循环不变式外提

循环不变式:如果一个表达式E在一个循环中不改变其值，则称它为该循环的不变表达式。主要是检查循环内的变量是否被定值

# 运行时的存储空间管理

- 存储结构、各区的存储分配的特点、分配对象和方法
- AR的结构、内容
- 调用链、动态链、声明链、变量访问环境的含义及相互间的关系。
- 变量访问环境的实现方法：

  静态链、局部Display表


top
临时变量区
局部变量区
形参区
变量访问环境
活动记录大小
寄存器状态
过程层数
返回值
返回地址
动态链指针
sp
过程活动记录
最大地址
堆区空间
栈区空间
静态区空间
目标代码空间
库代码空间
最小地址

**动态链：**

如果有调用链CallChain(S)=(M,…,R, S)，

则它对应的动态链为：

DynamicChain=[AR(M),…,AR(R),AR(S)]

**LiveAR（LAR）：**

一个过程S在动态链中可有多个AR，但其中只有最新AR(S)是可访问的，称此AR(S)为S的活跃活动记录，并记为LiveAR(S)，简写为LAR（S）。

**变量访问环境VarVisitEnv：**在动态链中，对应当前函数的声明链的活跃活动记录

**局部display表和静态链**

# 目标代码生成

- 目标代码形式
- 单寄存器的目标代码结构
  - 表达式操作的目标代码
  - 赋值的目标代码
  - 标号和跳转的目标代码
  - 过程的传参、调用、声明、调用结束