总复习

第一章

概论

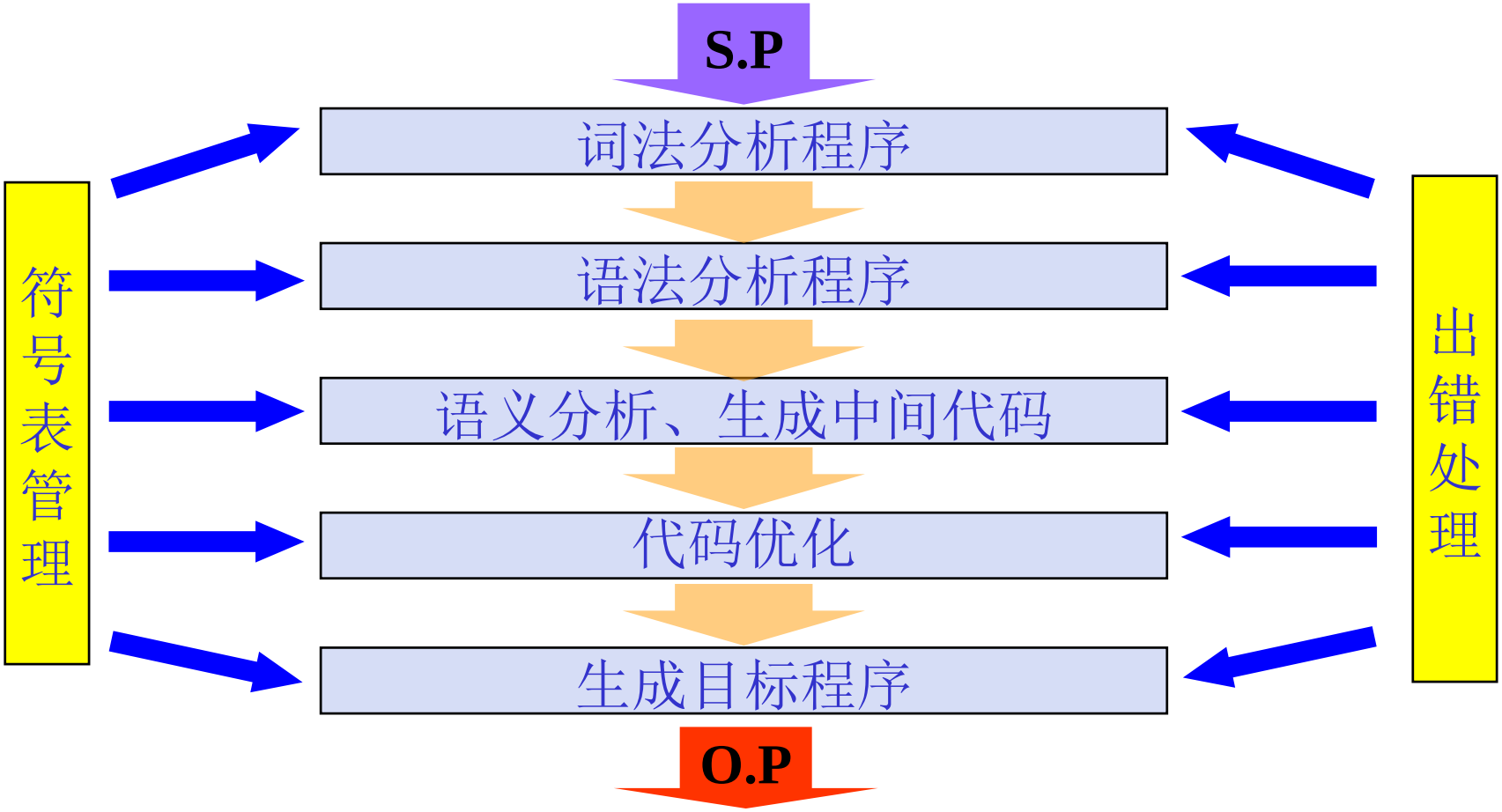
(介绍名词术语、了解编译系统的结构和编译过程）

1.2 编译过程

所谓编译过程是指将高级语言程序翻译为等价的目标程序的过程。

习惯上是将编译过程划分为5个基本阶段：

典型的编译程序具有7个逻辑部分



第二章

- 掌握符号串和符号串集合的运算、文法和语言的定义
- 几个重要概念：递归、短语、简单短语和句柄、语法树、文法的二义性、文法的实用限制等。
- 掌握文法的表示：BNF、扩充的BNF范式、语法图。
- 了解文法和语言的分类

第三章和第十一章:词法分析

3.1 词法分析的功能

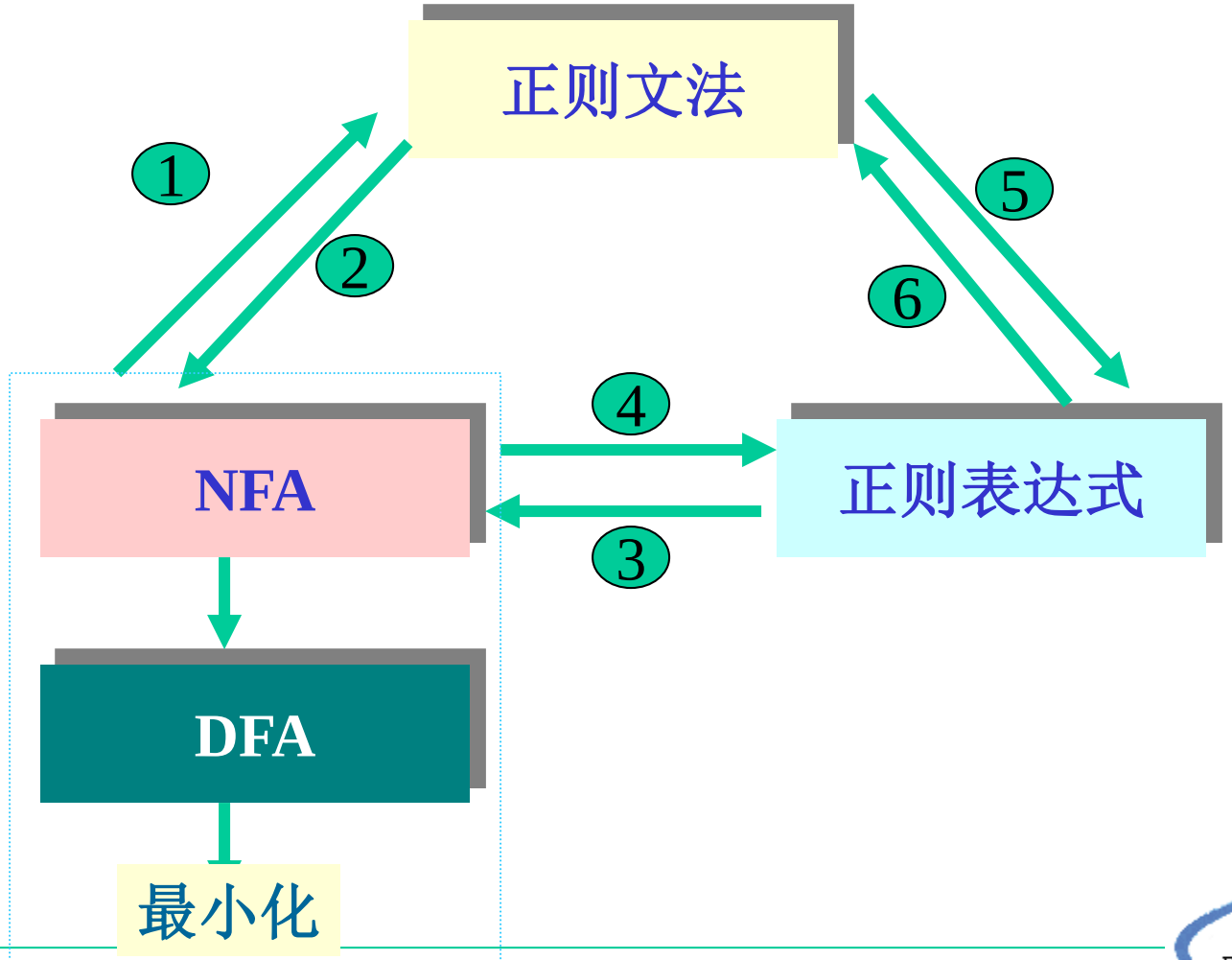
3.2 词法分析程序的设计与实现

- 状态图

11. 词法分析程序的自动生成

- 有穷自动机、LEX

补充



第四、十二章　语法分析

语法分析方法：
- 自顶向下分析法　$Z \overset{+}{\Rightarrow} S$
- 自底向上分析法　$S \overset{+}{\Leftarrow} Z$

$S \in L[Z]$

(一) 自顶向下分析

①概述自顶向下分析的一般过程

存在问题
- 左递归问题 ——— 消除左递归的方法
- 回溯问题 ———
  - 无回溯的条件
  - 改写文法
  - 超前扫描

②两种常用方法:

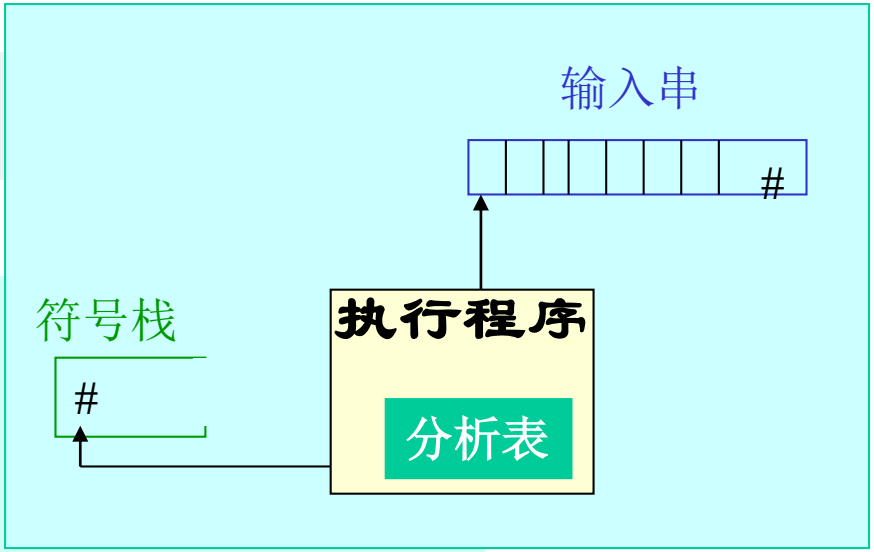
- (1)递归子程序法
  - a)改写文法,消除左递归,回溯
  - b)写递归子程序
- (2)LL(1)分析法
  - LL(1)分析器的逻辑结构及工作过程
  - LL(1)分析表的构造方法
    - 1.构造First集合的算法
    - 2.构造Follow集合的算法
    - 3.构造分析表的算法
  - LL(1)文法的定义以及充分必要条件

12.1 LL分析法

LL－自左向右扫描、自左向右地分析和匹配输入串。

∴ 分析过程表现为最左推导的性质。

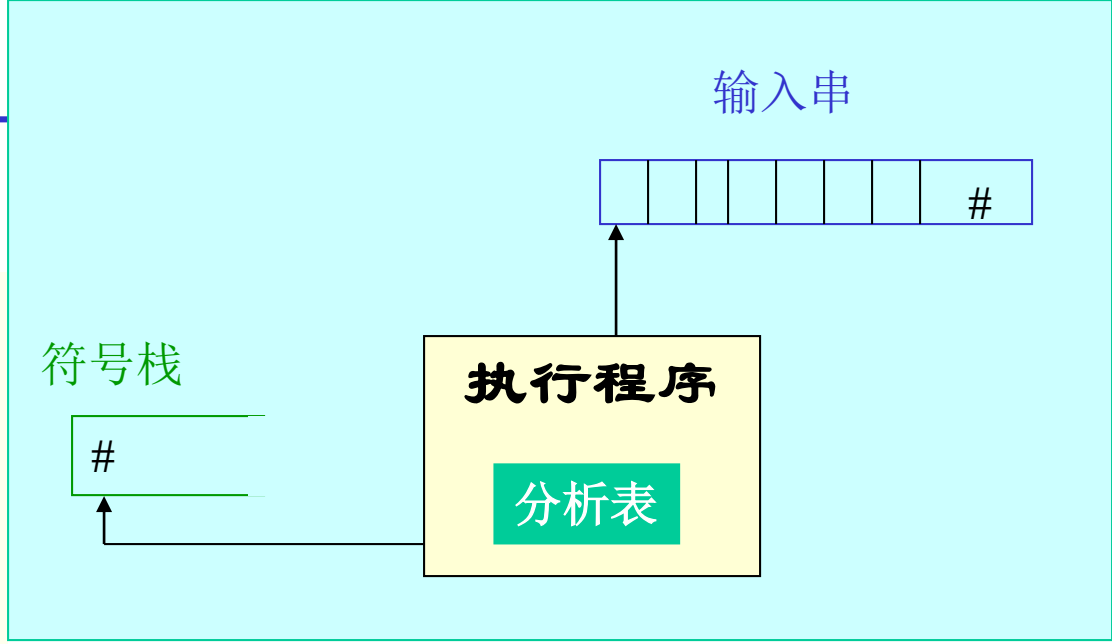
1、LL分析程序构造及分析过程



由三部分组成:

- 分析表
- 执行程序 (总控程序）
- 符号栈 （分析栈）

在实际语言中，每一种语法成分都有确定的左右界符，为了研究问题方便，统一以‘#’表示。



（1）、分析表：二维矩阵M

$$M[A,a]=\begin{cases} A::=\alpha_i & \alpha_i \in V^* \\ \text{或} & A \in V_n \\ \text{error} & a \in V_t \text{ or } \# \end{cases}$$

M[A,a] = A::=α_i

表示当要用A去匹配输入串时，且当前输入符号为a时，可用A的第i个选择去匹配。

即 当$\alpha_i \neq \varepsilon$时，有$\alpha_i \Rightarrow^{*} a\ldots$;

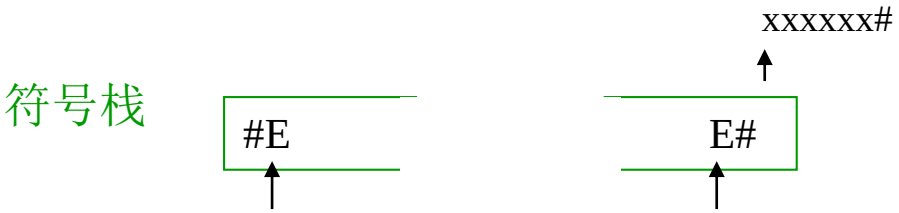
当$\alpha_i = \varepsilon$时，则a为A的后继符号。

M[A,a] = error

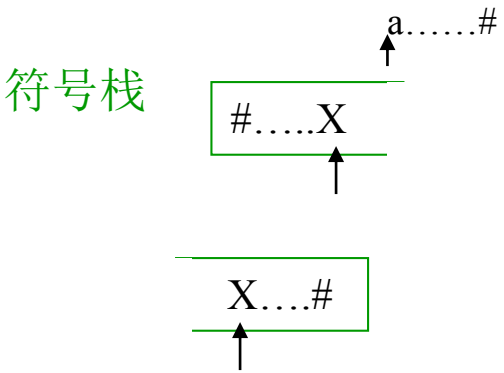
表示当用A去匹配输入串时，若当前输入符号为a，则不能匹配，表示无$A \not\Rightarrow^{*} a\ldots$,或a不是A的后继符号。

（2）符号栈： 有四种情况

- 开始状态



- 工作状态



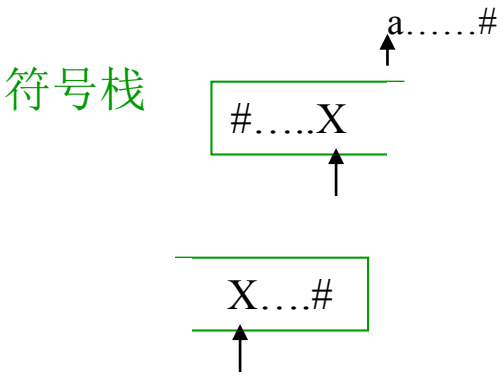
查分析表得：

$X \in V_n$, $M[X,a] = X::=\alpha_i$

$X \overset{+}{\Rightarrow} a\ldots$

$X \in V_t$, $X = a$

- 出错状态



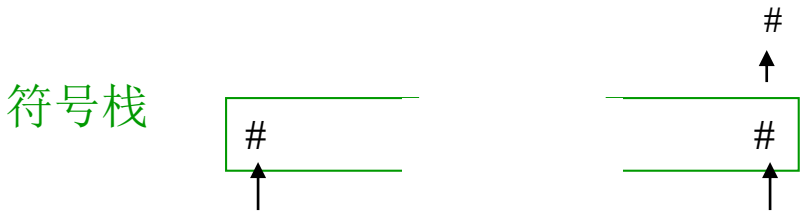
查分析表得：

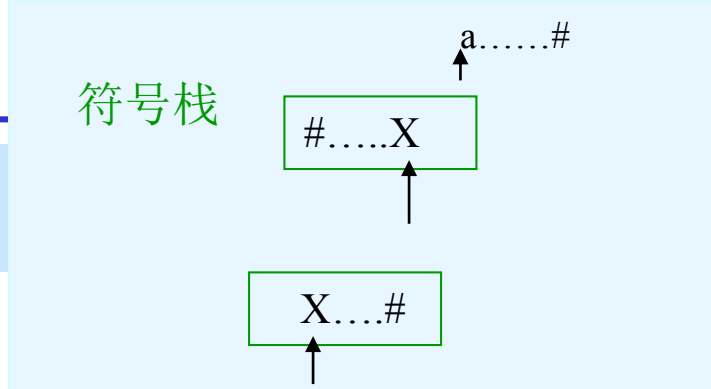
$X \in V_n$, $M[X,a] = error$

无$X \overset{+}{\Rightarrow} a\ldots$

$X \in V_t$, $X \neq a$

- 结束状态





（3）、执行程序

执行程序主要实现如下操作:

1. 把#和文法识别符号E推进栈, 读入下一个符号，重复下述过程直到正常结束或出错。

2. 测定栈顶符号X和当前输入符号a,执行如下操作:

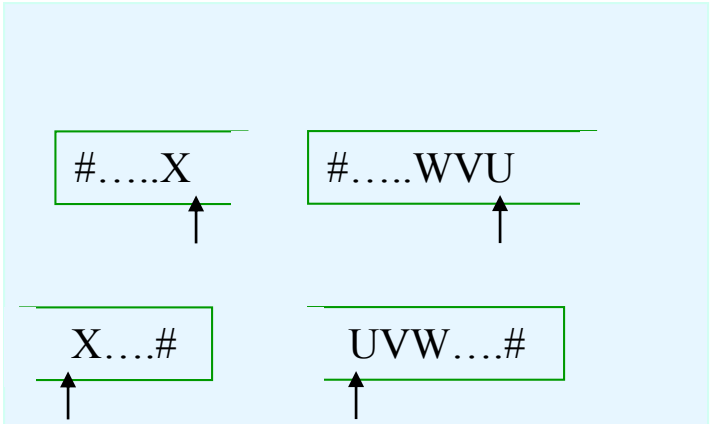
(1) 若X=a=#，分析成功，停止。E匹配输入串成功。
(2) 若X=a≠#，把X推出栈，再读入下一个符号。
(3) 若X∈V_n，查分析表M。

(3) 若X∈V_n，查分析表M。

a) M[X,a] = X∷= UVW
则将X弹出栈，将UVW压入
注：U在栈顶 （最左推导）

b) M[X, a] = error 转出错处理

c) M[X, a] = X::=ε，
— a为X的后继符号
则将X弹出栈 (不读下一符号)
继续分析。



分析表

	i	+	*	(	)	#
E	E::=TE'			E::= TE'		
E'		E'::= +TE'			E'::= ε	E'::= ε
T	T::=FT'			T::= FT'		
T'		T'::= ε	T'::= *FT'		T'::= ε	T'::= ε
F	F::= i			F::= (E)		

★ 注：矩阵元素空白表示Error

3、LL(1)文法

定义：一个文法G，其分析表M不含多重定义入口(即分析表中无二条以上规则)，则称它是一个LL(1)文法.

定理：文法G是LL(1)文法的充分必要条件是：对于G的每一个非终结符A的任意两条规则A::=α|β,下列条件成立：

1、FIRST(α) ∩ FIRST(β) = Φ

2、若β=*=> ε, 则FIRST(α) ∩ FOLLOW(A) = Φ

(二) 自底向上分析

规约过程:

(1)一般过程:　　移进—规约过程

问题:如何寻找句柄

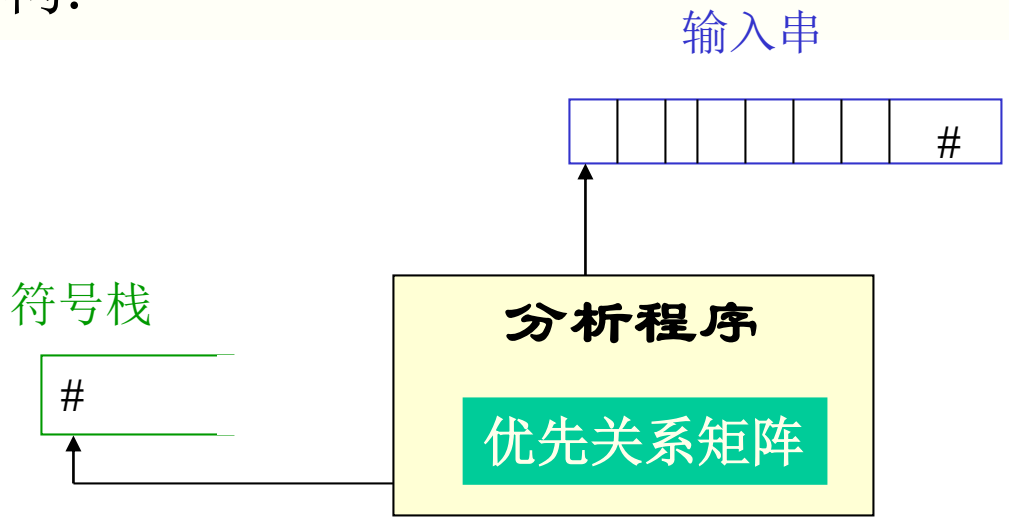
(2)算法:

i)算符优先分析法:

1.分析器的构造,分析过程.

根据算符优先关系矩阵来决定是移进还是规约.

- 分析器结构:



a \ b	+	*	i	(	)	#
+	·>	<·	<·	<·	·>	·>
*	·>	·>	<·	<·	·>	·>
i	·>	·>			·>	·>
(	<·	<·	<·	<·	≐	
)	·>	·>			·>	·>
#	<·	<·	<·	<·		

2.算符优先法的进一步讨论

1.适用的文法类----------引出的算符优先文法的定义
2.优先关系矩阵地构造
3.什么是“句柄”,如何找
有句柄引出的最左素短语的概念.
最左素短语的定理,如何找.

算符优先文法（OPG）的定义

设有一OG文法，如果在任意两个终结符之间，至多只有上述关系中的一种，则称该文法为算符优先文法(OPG)

（2）构造优先关系矩阵

- 求 “ = ” 检查每一条规则，若有U::=…ab…或U::=…aVb…, 则 a=b

- 求 “ < ”、 “ > ”复杂一些，需定义两个集合

$$FIRSTVT(U) = \{b \mid U \overset{+}{\Rightarrow} b\ldots \text{或} U \overset{+}{\Rightarrow} Vb\ldots, b \in V_t, V \in V_n\}$$

$$LASTVT(U) = \{a \mid U \overset{+}{\Rightarrow} \ldots a \text{或} U \overset{+}{\Rightarrow} \ldots aV, a \in V_t, V \in V_n\}$$

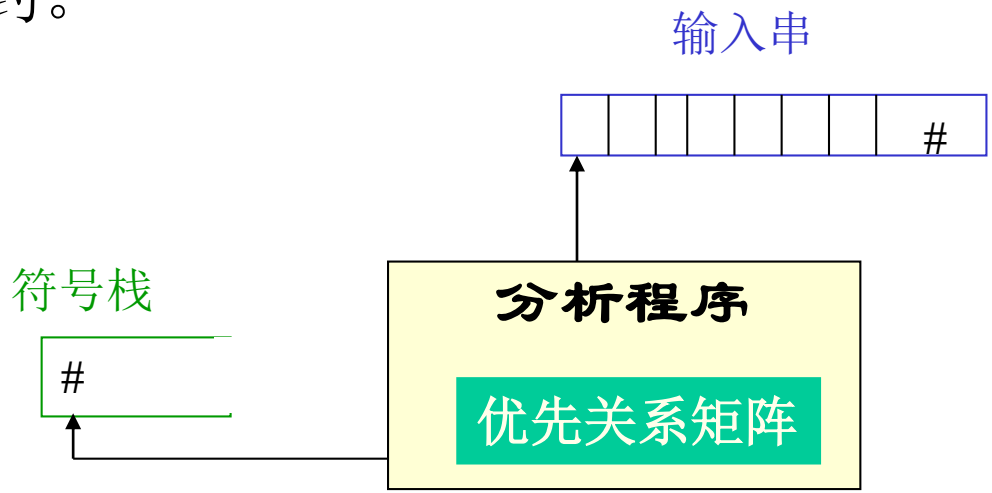
- 求“$\cdot<$”、“$\cdot>$”：

若文法有规则
W∷=...aU... ，对任何b, b∈FIRSTVT(U)
则有：a $\lessdot$ b

若文法有规则
W∷=...Ub... ，对任何a, a∈LASTVT(U)
则有：a $\gtrdot$ b

算符优先分析法的实现：

基本部分是找句型的最左子串（最左素短语）并进行归约。



当栈内终结符的优先级＜＝栈外的终结符的优先级时，移进；栈内终结符的优先级＞栈外的终结符的优先级时，表明找到了素短语的尾，再往前找其头，并进行归约。

ii)LR分析法

1.概述----概念、术语　(活前缀、项目)

- 2.LR分析
  - 1)逻辑结构
    - 状态栈
    - 分析表
      - GOTO表
      - 分析动作表
    - 控制程序
  - 2)分析过程

- 3.如何构造SLR分析表
  - 1)构造DFA
  - 2)由DFA构造分析表

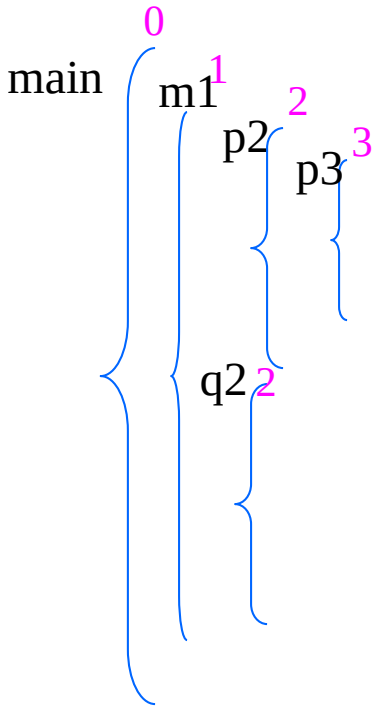
SLR分析

- 最好给出状态图的构造过程。

第五章:符号表管理技术

- 5.1 概述
- 5.2 符号表的组织与内容
- 5.3 非分程序结构语言的符号表组织
- 5.4 分程序结构语言的符号表组织

```
Progiam   mail (…);                                   0
  var
     x, y : real;   i, k: integer;
     name:  array [1…16] of char;
        :
     procedure  M1 (ind:integer);                     1
          var    x : integer
           procedure  P2 (j : real);                  2
                    :
                  procedure   P3;                     3
                      var
                       f : array [1…5] of intrger
                        test1:  boolean;
                        begin
                          :
                       end;     {P3}
              begin
                 :
             end;{p2}
           procedure  q2;                             2
              var   r1,r2 : real;
               begin
                 :
               p2(r1+r2);
                 :
              end;  {q2}
       begin
          :
        P2(x/y);
          :
     end;{M1}
begin
   :
 M1(i+k);
   :
End    {mail}
```



符号表

	name	kind	type	lev	other inf
1	x	var	real	0	
2	y	var	real	0	
3	i	var	int	0	
4	k	var	int	0	
5	name	var	array	0	
6	M_1	proc		0	
7	ind	para	int	1	
8	x	var	int	1	
9	P_2	proc		1	
10	j	para	real	2	
11	P_3	proc		2	
12	f	var	array	3	
13	test1	var	boolean	3	

main

M_1

P_2

P_3

分程序索引表

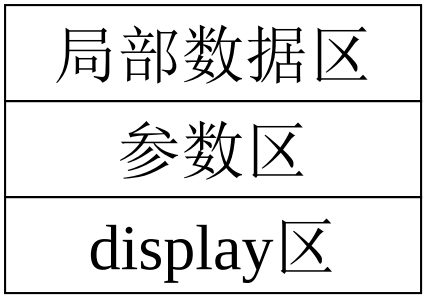
0	1
1	7
2	10
3	12

第六章:运行时的存储组织及管理

- 6.1 概述
- 6.2 静态存储分配
- 6.3 动态存储分配

6.3.1 活动记录

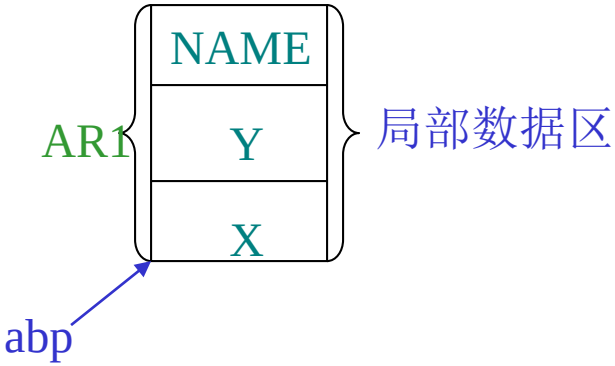
一个典型的活动记录可以分为三部分：



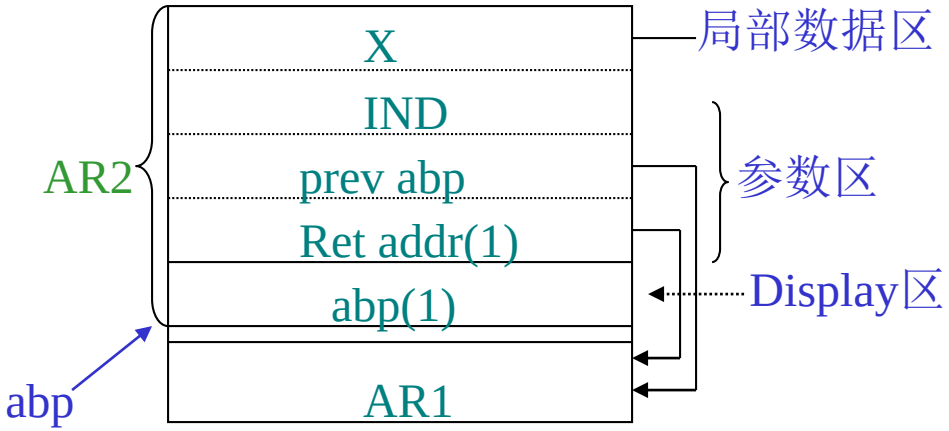
(1)局部数据区：

存放模块中定义的 各个局部变量。

例：下面给出源程序的目标程序运行时，运行栈(数据区栈)的跟踪情况



(a)进入模块1



(b)M1被调用

第七章:源程序的中间形式

- 7.1 波兰表示
- 7.2 N一元表示
- 7.3 抽象机代码

第八章:错误处理

- 8.1 概述
- 8.2 错误分类
- 8.3 错误的诊察和报告
- 8.4 错误处理技术

第九章:语法制导翻译技术

- 5.1 翻译文法(TG)和语法制导翻译
- 5.2 属性翻译文法(ATG)
  - 继承属性
  - 综合属性
- 5.3 自顶向下语法制导翻译
  - 翻译文法的自顶向下语法制导翻译
  - 属性文法的自顶向下语法制导翻译
- 5.4 自底向上的语法制导翻译（自学）

第十章:语义分析和代码生成

- 10.1 语义分析的概念
- 10.2 栈式抽象机及其汇编指令
- 10.3 声明的处理
- 10.4 表达式的处理
- 10.5 赋值语句的处理
- 10.6 控制语句的处理
- 10.7 过程调用和返回

第14章:代 码 优 化

总结:

优化分为两大类

- 与机器无关的优化独立于机器的（中间）代码优化
- 与机器有关的优化目标代码上的优化（与具体机器有关）

优化方法的分类2：

- 局部优化技术
  - 指在基本块内进行的优化
  - 例如，局部公共子表达式删除
- 全局优化技术
  - 函数/过程内进行的优化
  - 跨越基本块
  - 例如，全局数据流分析
- 跨函数优化技术
  - 整个程序
  - 例如，跨函数别名分析 ，逃逸分析 等

代码优化

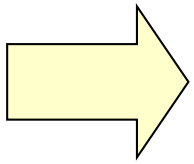
- DAG图
  - 消除局部公共子表达式
  - 从DAG图导出中间代码的启发式算法
- 数据流分析
  - 到达定义分析
  - 活跃变量分析
- 构建冲突图
  - 变量冲突的基本概念
  - 通过活跃变量分析构建（精度不太高的）冲突图

算法14.1 划分基本块

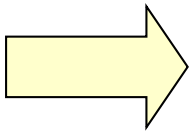
- **输入**：四元式序列
- **输出**：基本块列表，每个四元式仅出现在一个基本块中
- **方法**：
  - 1、**首先确定入口语句（每个基本块的第一条语句）的集合**
    - 规则1：整个语句序列的第一条语句属于入口语句
    - 规则2：任何能由条件/无条件跳转语句转移到的第一条语句属于入口语句
    - 规则3：紧跟在跳转语句之后的第一条语句属于入口语句
  - 2、**每个入口语句直到下一个入口语句，或者程序结束，它们之间的所有语句都属于同一个基本块**

消除局部公共子表达式

```
t1 := - c
t2 := b * t1
t3 := - c
t4 := b * t3
t5 := t2 + t4
a := t5
```



DAG图



```
t1 := - c
t2 := b * t1
t5 := t2 + t2
a := t5
```

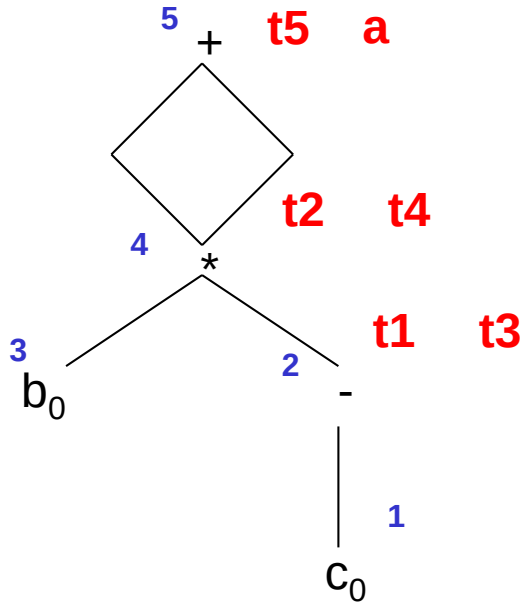
需要两个算法：

1、DAG图的生成算法

2、从DAG图导出代码的算法

建立DAG图，例1 a=b*(-c) + b*(-c)

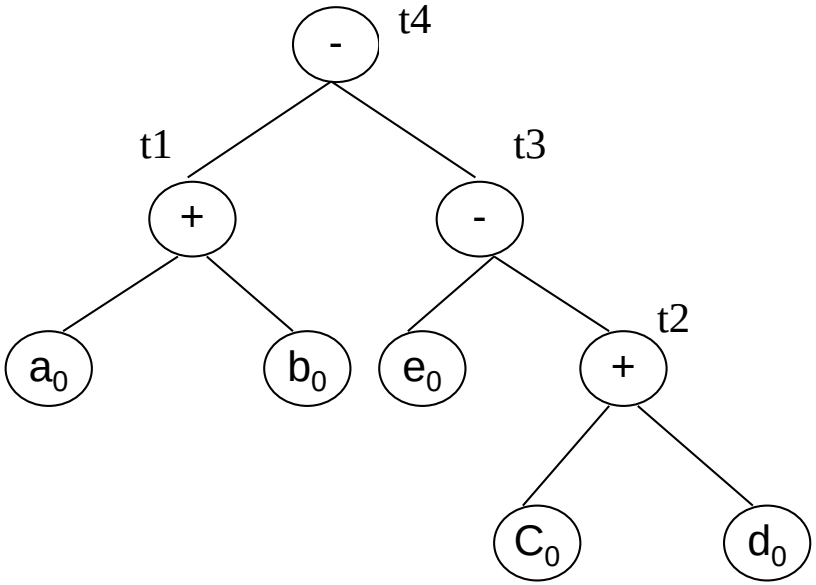
```
t1 := - c
t2 := b * t1
t3 := - c
t4 := b * t3
t5 := t2 + t4
a := t5
```



node(x)

c	1
t1	2
b	3
t2	4
t3	2
t4	4
t5	5
a	5

算法14.2 从DAG导出中间代码的启发式算法



1、初始化一个放置DAG图中间节点的队列

2、如果DAG图中还有中间节点未进入队列，则执行步骤3，否则执行步骤5。

3、选取一个尚未进入队列，但其**所有父节点均已进入队列**的中间节点n，将其加入队列；或选取**没有父节点的中间节点**，将其加入队列

4、如果n的最左子节点符合步骤3的条件，将其加入队列；并沿着当前节点的最左边，循环访问其**最左子节点**，最左子节点的最左子节点等，将符合步骤3条件的中间节点依次加入队列；如果出现不符合步骤3条件的最左子节点，执行步骤2。

5、将中间节点队列逆序输出，便得到中间节点的计算顺序，将其整理成中间代码序列

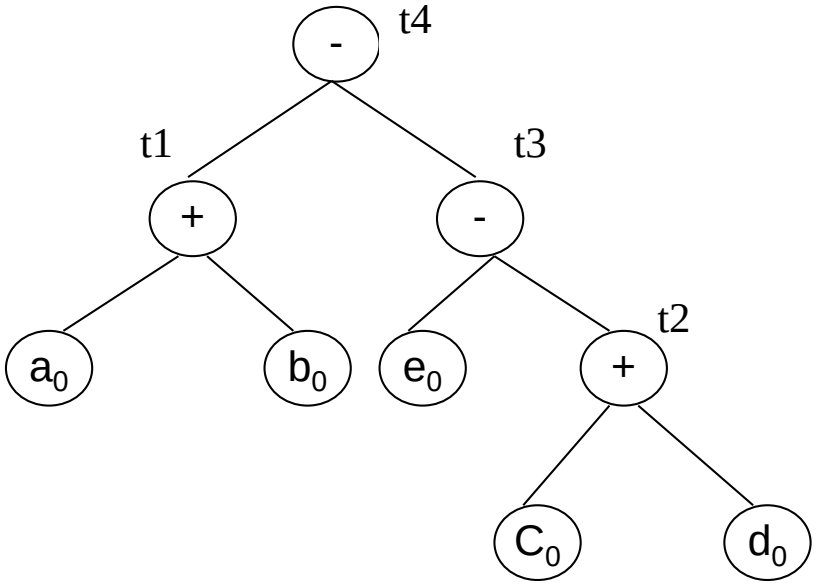
中间节点队列：

t4	t1	t3	t2

中间节点队列：

t4	t1	t3	t2

5、将中间节点队列逆序输出，便得到中间节点的计算顺序，将其整理成中间代码序列



$t2 = c + d$

$t3 = e - t2$

$t1 = a + b$

$t4 = t1 - t3$

数据流分析方程

- 考察在程序的某个执行点的数据流信息。

- out[S] = gen[S] ∪ (in[S] – kill[S])
  - S代表某条语句（也可以是基本块，或者语句集合，或者基本块[illegible]等）
  - out[S]代表在该语句[illegible]到的数据流信息
  - gen[S]代表该[illegible]
  - in[S]代表进入[illegible]
  - kill[S]代表该[illegible]

含义：当执行控制流通过某一条语句时，在该语句**末尾得到的数据流信息**等于该语句**本身产生**的数据流信息，**合并进入该语句时的数据流信息减去**该语句**注销的数据流信息**后的数据流信息。

到达定义（reaching definition）分析

- 通过到达定义分析，希望知道：
  - 在程序的某个静态点p，例如某条中间代码之前或者之后，某个变量可能出现的值都是在哪里被定义的？
- 在p处对该变量的引用，取得的值是否在d处定义？
  - **如果从定义点d出发，存在一条路径达到p，并且在该路径上，不存在对该变量的其他定义语句，则认为“变量的定义点d到达静态点p”**
  - **如果路径上存在对该变量的其他赋值语句，那么路径上的前一个定义点就被路径上的后一个定义点“杀死”，或者消除了**

14.5.2 活跃变量分析（Live-variable Analysis）

- 达到定义分析是沿着流图路径的，有的数据流分析是方向计算的
- **活跃变量分析：**
  - **了解变量x在某个执行点p是活跃的**
    - **变量x的值在p点或沿着从p出发的某条路经中会被使用，则称x在p点是活跃的。**
  - 通过活跃变量分析，可以了解到某个变量x在程序的某个点上是否活跃，或者从该点出发的某条路径上是否会被使用。如果存在被使用的可能，x在该程序点上便是活跃的，否则就是非活跃，或者死的。

与到达定义分析的区别

活跃变量分析：

$in[B] = use[B] \cup (out[B] - def[B])$

到达定义分析：

$out[B] = gen[B] \cup (in[B] - kill[B])$

- 采用use[B]代表当前基本块新生成的数据流信息（用了）
- 采用def[B]代表当前基本块消除的数据流信息 （定义的）
- 采用in[B]而不是out[B]来计算当前基本块中的数据流信息
- 采用out[B]而不是in[B]来计算其它基本块汇集到当前基本块的数据流信息
- 在汇集数据流信息时，考虑的是后继基本块而不是前驱基本块

14.5.3 定义-使用链、网和冲突图

- **冲突图：**其节点是待分配全局寄存器的变量，当**两个变量中的一个变量在另一个变量定义（赋值）处是活跃的**，它们之间便有一条边连接。

十五章 代码生成

- 微处理器体系结构基础知识
- 基本块和流图的划分与建立方法
- 全局寄存器分配算法
  - 引用计数
  - 图着色
- 临时寄存器池管理方法与指令选择

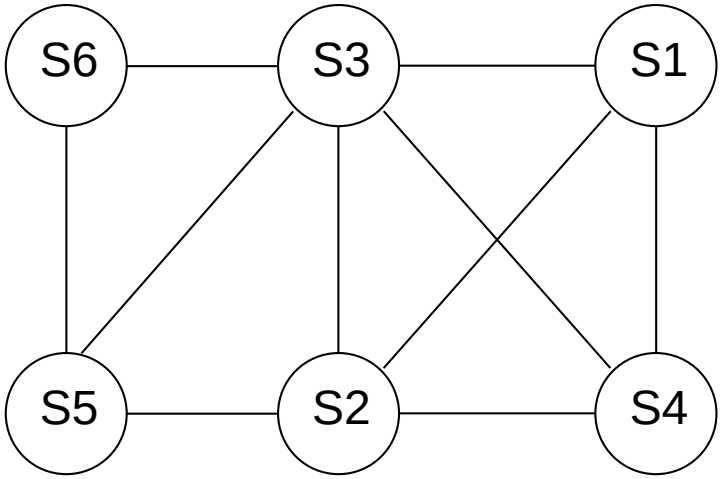
15.4.1.2 图着色算法

- 一种简化的图着色算法
  - 步骤：
  - 1、通过数据流分析，构建变量的冲突图
  - 2、如果可供分配k个全局寄存器，那么我们就尝试用k种颜色给该冲突图着色

算法15.2 一种启发式图着色算法

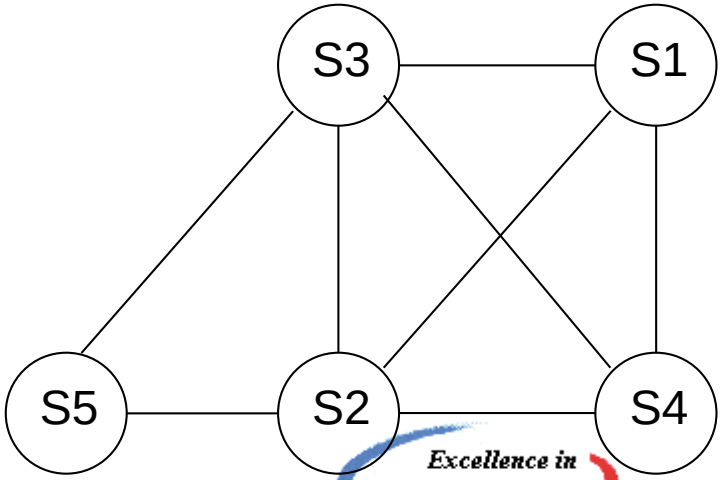
- 冲突图G
  - 寄存器数目为K
    - 假设 K=3

G

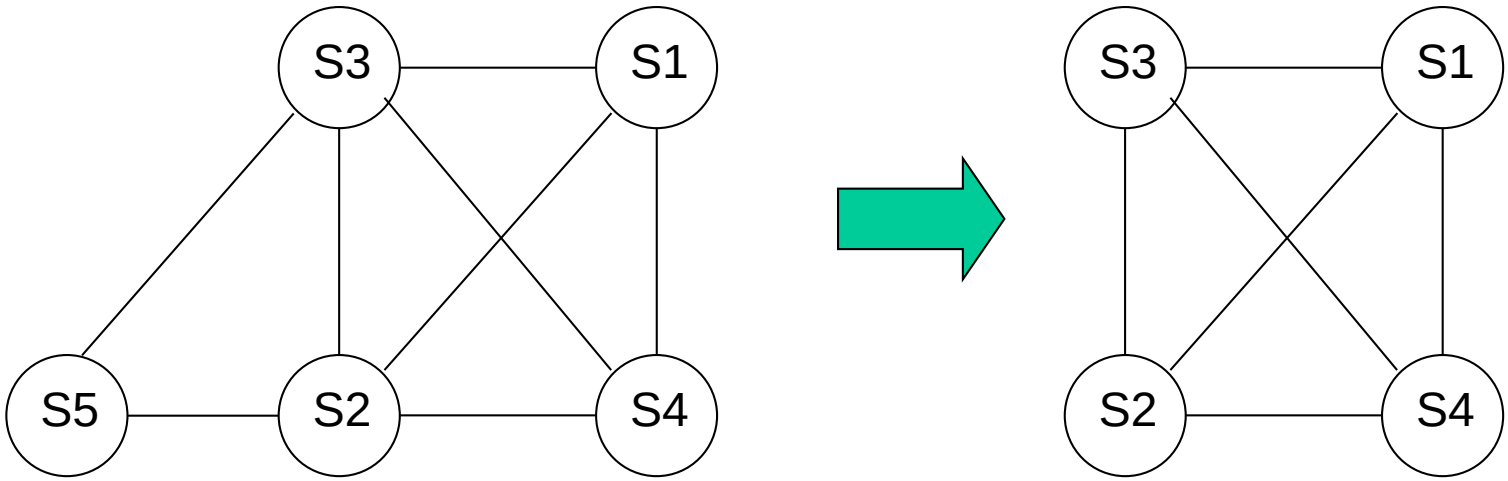


- **步骤1**、找到第一个连接边数目小于K的结点，将它从图G中移走，形成图G’

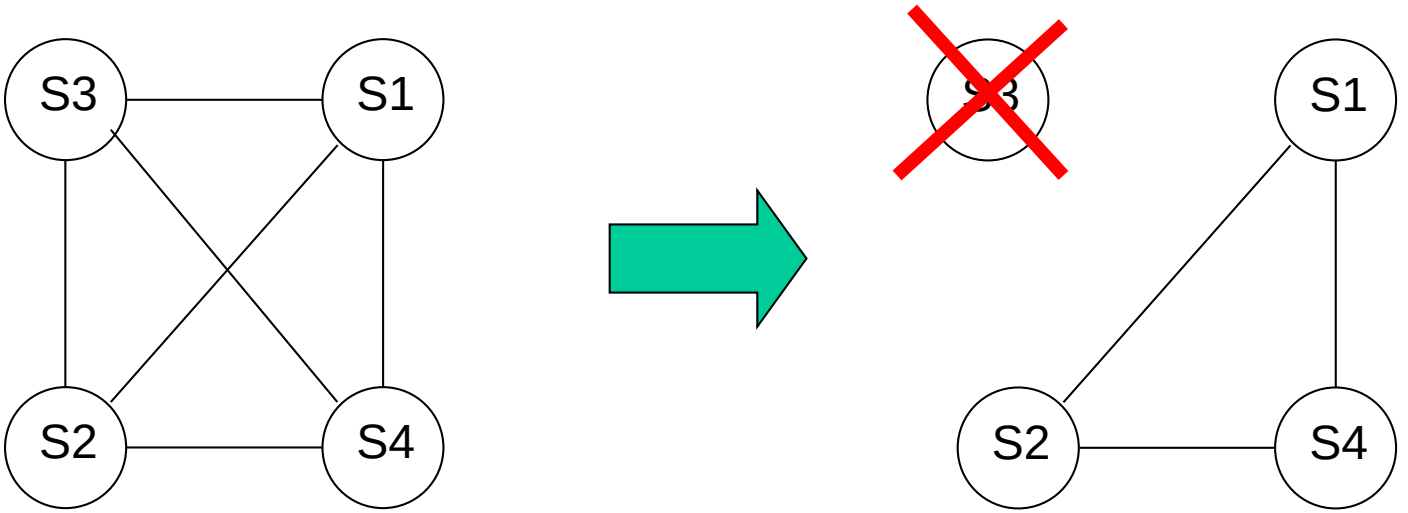
G’



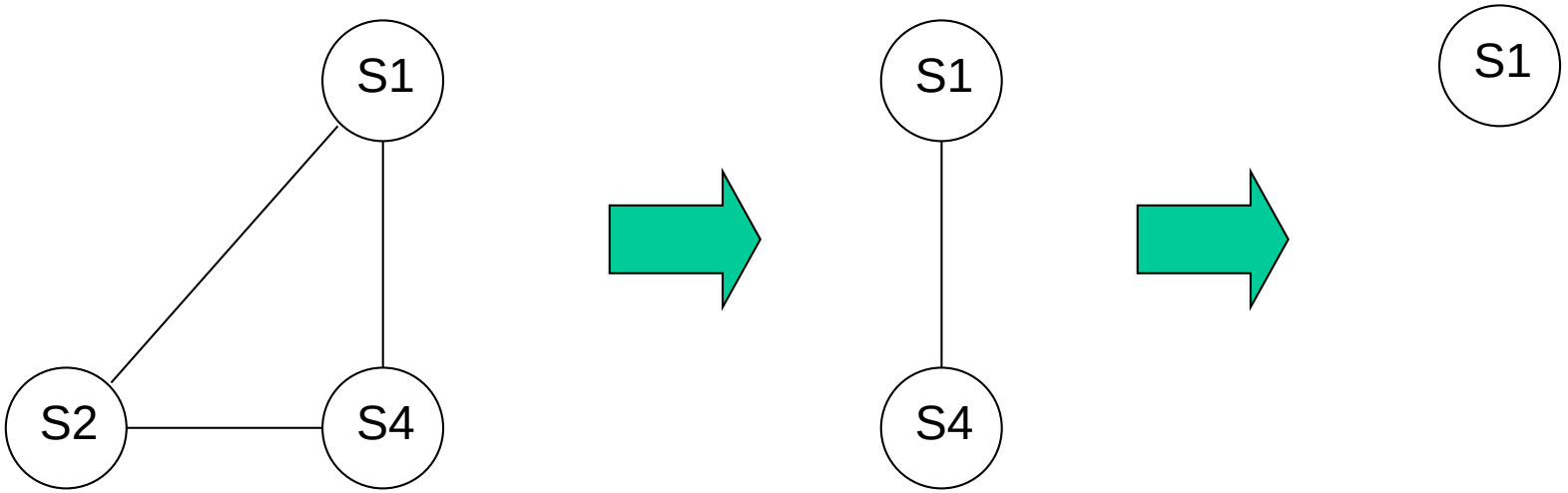
- **步骤2**、重复步骤1，直到无法再从G'中移走结点



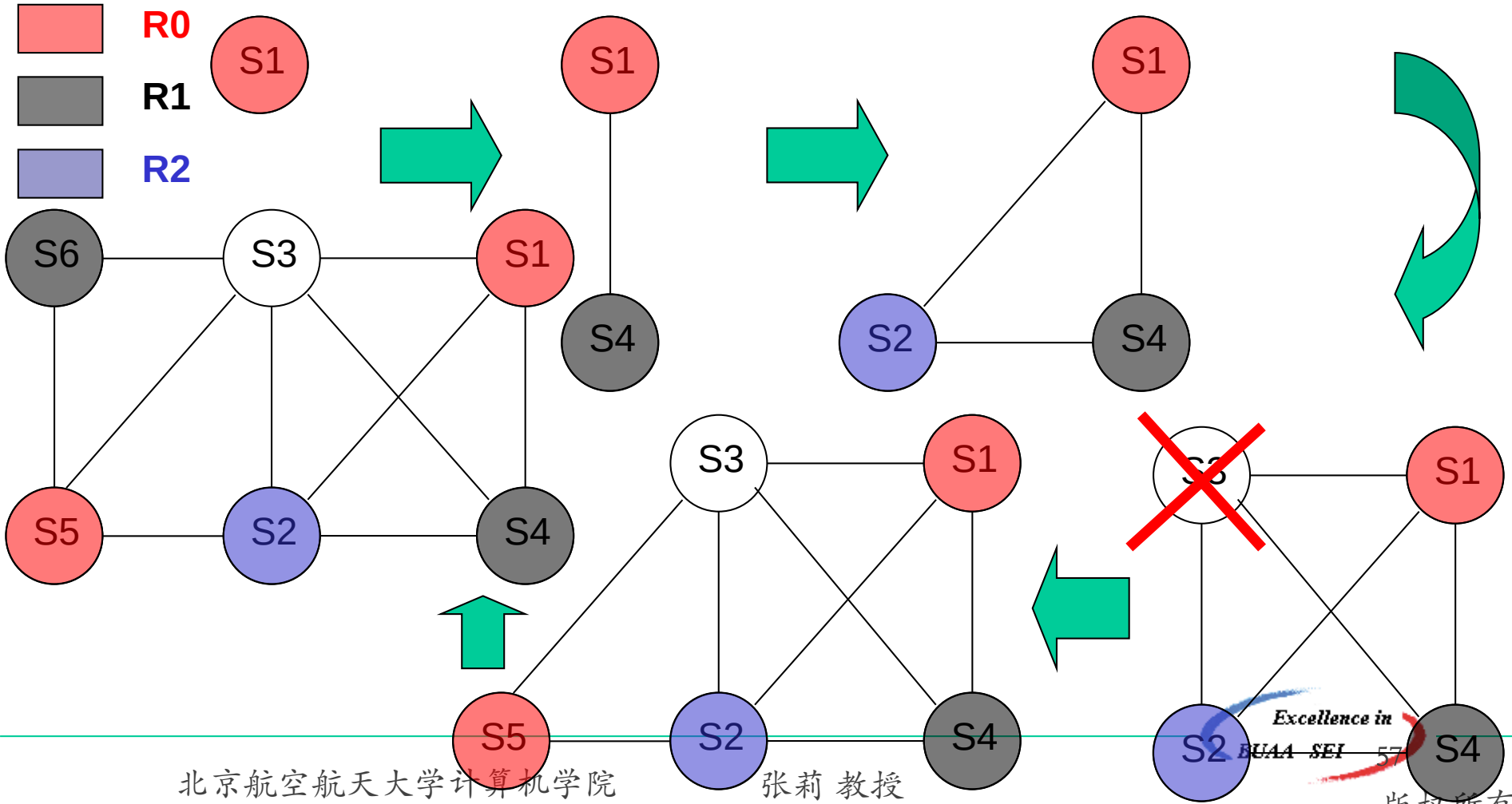
- **步骤3**、在图中选取*适当*的结点，将它记录为“不分配全局寄存器”的结点，并从图中移走



- **步骤4**、重复步骤1~步骤3，直到图中仅剩余1个结点



- 步骤5、给剩余的最后一个结点选取一种颜色，然后按照结点被移走的顺序，反向将结点和边添加进去，并依次给新加入的结点选取颜色。（保证有链接边的结点着不同的颜色）



第十六章 编译程序生成方法和工具

- 自编译
- 自展
- 交叉编译

- 认真复习，有重点，
- 概念清楚、例题自己会做、习题会做。

- 考过了，还会考
- 没有考过的，也会考。