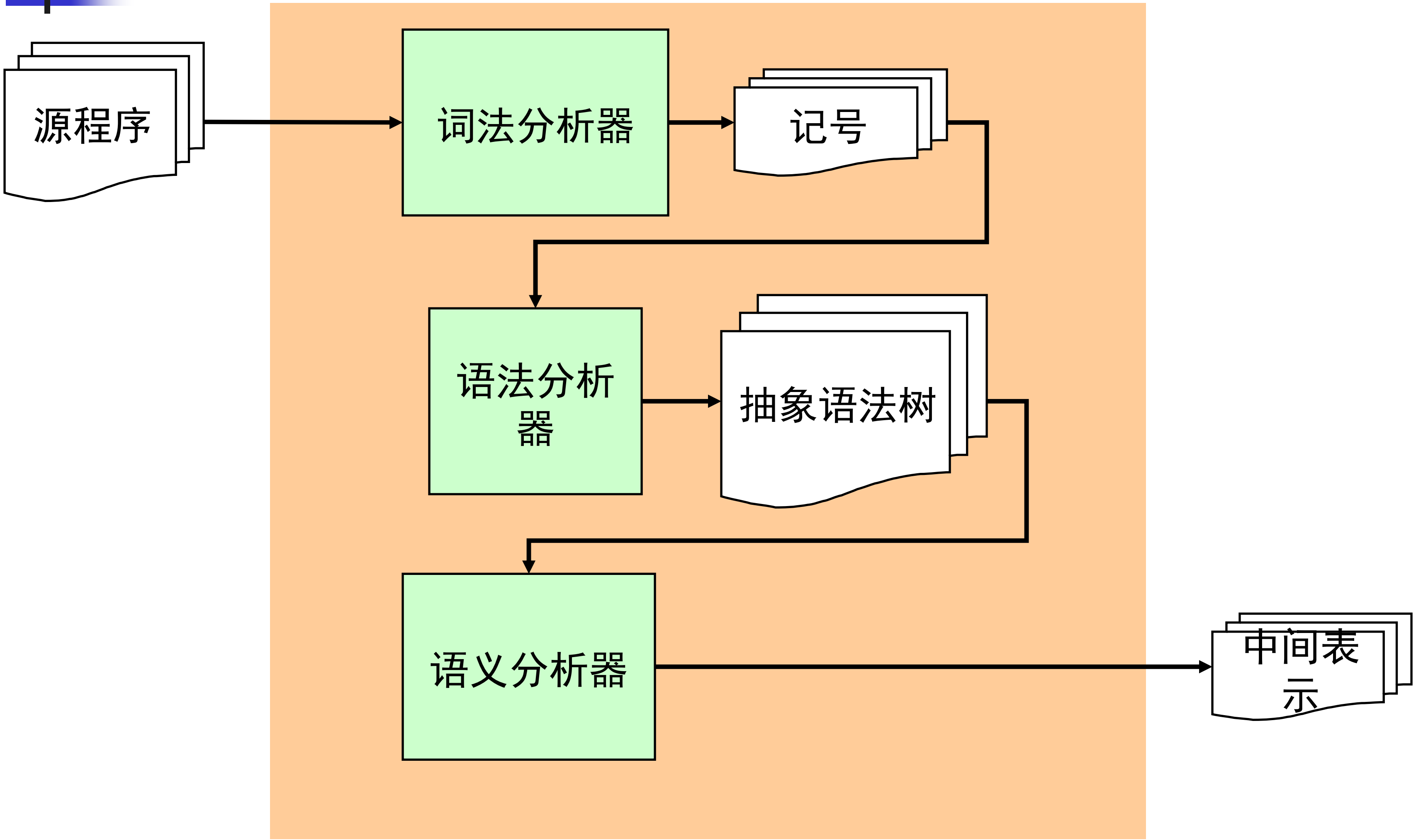
语法分析：
LL(1)分析算法

编译原理

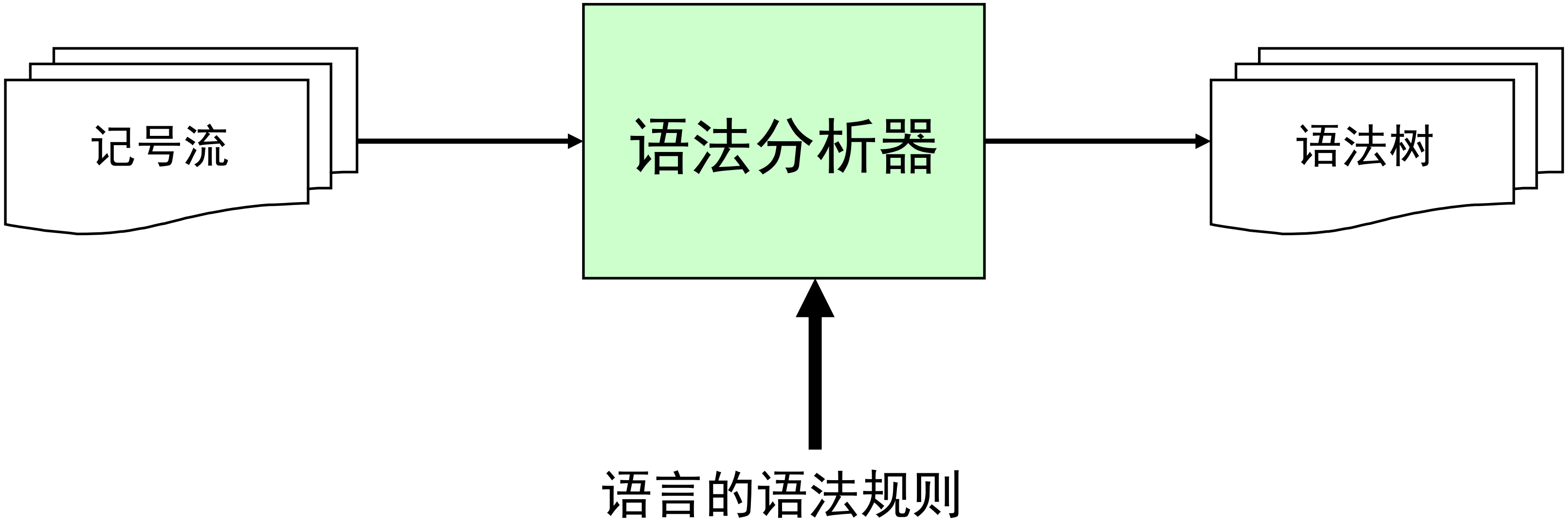
华保健

bjhua@ustc.edu.cn

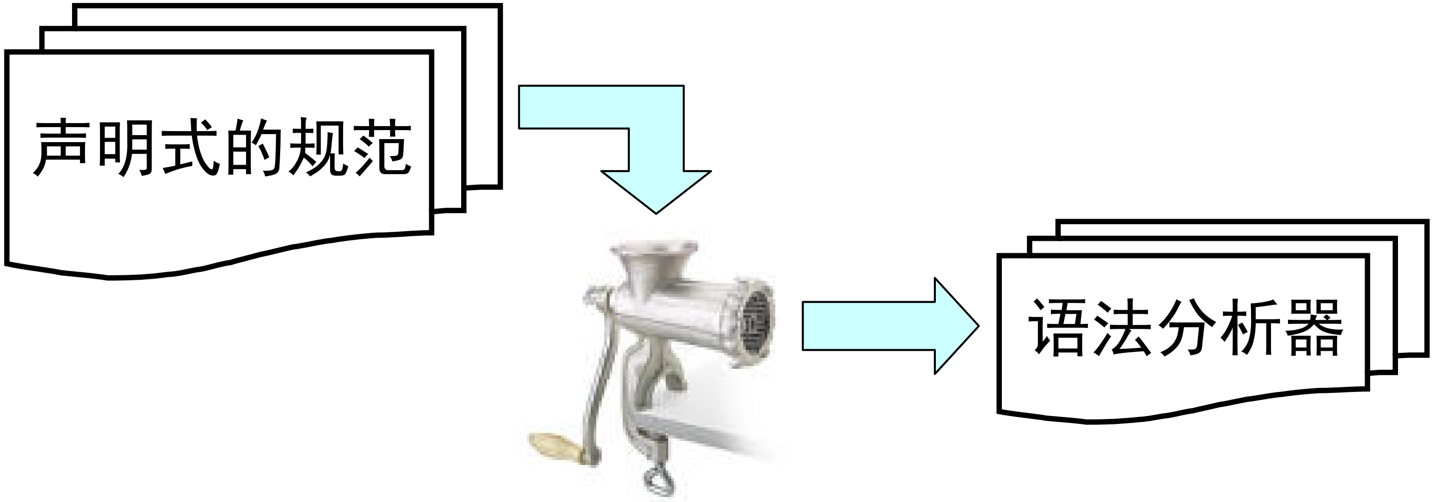
前端



语法分析器的任务



自动生成



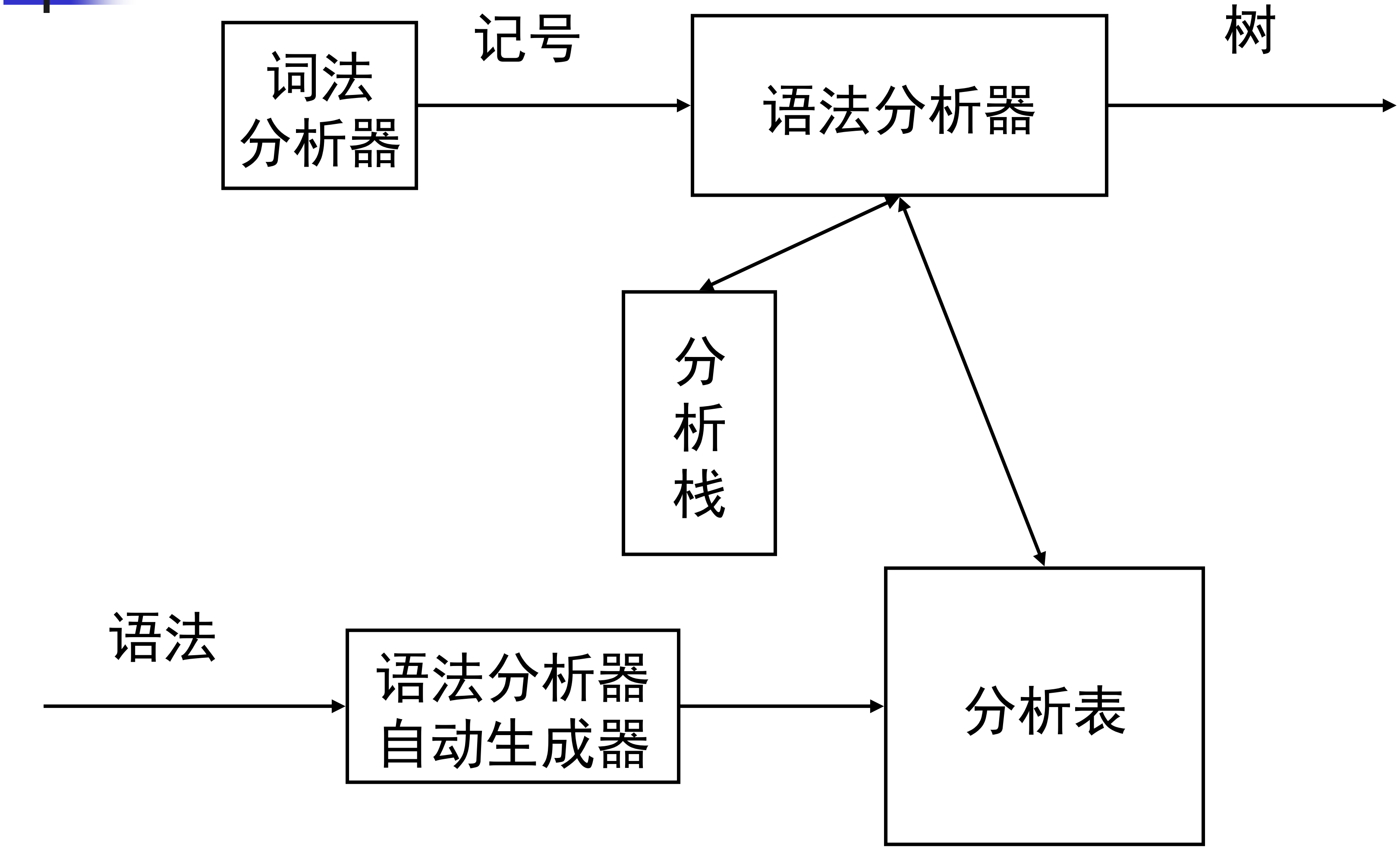
声明式的规范

语法分析器

LL(1)分析算法

- 从左（L）向右读入程序，最左（L）推导，采用一个（1）前看符号
  - 分析高效（线性时间）
  - 错误定位和诊断信息准确
  - 有很多开源或商业的生成工具
    - ANTLR，。。。
- 算法基本思想：
  - 表驱动的分析算法

表驱动的LL分析器架构


词法
分析器
记号
语法分析器
树
分
析
栈
语法
语法分析器
自动生成器
分析表

回顾：自顶向下分析算法

```
tokens[];     // all tokens
i=0;
stack = [S]   // S是开始符号
while (stack != [])
   if (stack[top] is a terminal t)
      if (t==tokens[i++])
         pop();
      else ~~backtrack();~~ error(…)
   else if (stack[top] is a nonterminal T)
      pop(); push(the ~~next~~ right hand side of T)
                      correct
```

```
0: S -> N V N
1: N -> s
2:      | t
3:      | g
4:      | w
5: V -> e
6:      | d
```

分析这个句子

g d w

N\T	s	t	g	w	e	d
S	0	0	0	0		
N	1	2	3	4		
V					5	6

```
0: S -> N V N
1: N -> s
2:      | t
3:      | g
4:      | w
5: V -> e
6:      | d
```

```
tokens[];    // all tokens
i=0;
stack = [S]  // S是开始符号
while (stack != [])
  if (stack[top] is a terminal t)
    if (t==tokens[i++])
      pop();
    else ~~backtrack();~~ error(…)
  else if (stack[top] is a nonterminal T)
    pop(); push(the ~~next~~ right hand side of T)
                     correct  table[N, T]
```

分析这个句子

g d w

FIRST集

```
// 定义：
// FIRST(N) = 从非终结符N开始推
//            导得出的句子开头的
//            所有可能终结符集合

// 计算公式（第一个版本，近似！）：
对 N -> a …
FIRST(N) ∪= {a}

对 N -> M …
FIRST(N) ∪= FIRST(M)
```

```
0: S -> N V N
1: N -> s
2:    | t
3:    | g
4:    | w
5: V -> e
6:    | d
```

推导这个句子

```
g d w
```

FIRST集的不动点算法

```
foreach (nonterminal N)
  FIRST(N) = {}

while(some set is changing)
  foreach (production p: N->β1 … βn)
    if (β1== a …)
      FIRST(N) ∪= {a}
    if (β1== M …)
      FIRST(N) ∪= FIRST(M)
```

```
0: S -> N V N
1: N -> s
2:      | t
3:      | g
4:      | w
5: V -> e
6:      | d
```

N\FIRST	0	1	2	3	4	5
S	{}					
N	{}					
V	{}					

把FIRST集推广到任意串上

```
FIRST_S(β1 … βn) =
   FIRST(N),     if β1 == N;
   {a},          if β1 == a.
// 在右侧产生式上标记这个FIRST_S集合
```

```
0: S -> N V N
1: N -> s
2:      | t
3:      | g
4:      | w
5: V -> e
6:      | d
```

N\FIRST	
S	{s, t, g, w}
N	{s, t, g, w}
V	{e, d}

构造LL(1)分析表

N\T	s	t	g	w	e	d
S	0	0	0	0		
N	1	2	3	4		
V					5	6

```
              {s,t,g,w}
0: S -> N V N
1: N -> s {s}
2:      | t {t}
3:      | g {g}
4:      | w {w}
5: V -> e {e}
6:      | d {d}
```

N\FIRST	
S	{s, t, g, w}
N	{s, t, g, w}
V	{e, d}

LL(1)分析表中的冲突

N\T	s	t	g	w	e	d
S	0	0	0	0		
N	1	2	3	4,5		
V					5	6

冲突检测：

对N的两条产生式规则N->β 和N->γ，要求FIRST_S(β) ∩ FIRST_S(γ) ={}。

N\FIRST	
S	{s, t, g, w}
N	{s, t, g, w}
V	{e, d}

```
                {s,t,g,w}
0: S -> N V N
1: N -> s {s}
2:    | t {t}
3:    | g {g}
4:    | w {w}
5:    | w V  {w}
6: V -> e {e}
7:    | d {d}
```

一般条件下的LL(1)分析表构造

```
Z -> d
   | X Y Z
Y -> c
   |
X -> Y
   | a
```

- 首先研究右侧的例子：
  - FIRST_S(X Y Z)?
    - 一般情况下需要知道某个非终结符是否可以推出空串
    - NULLABLE
  - 并且一般需要知道在某个非终结符后面跟着什么符号
    - 跟随集FOLLOW

NULLABLE集合

- 归纳定义：
- 非终结符X属于集合NULLABLE，当且仅当：
  - 基本情况：
    - X ->
  - 归纳情况:
    - X -> Y1 … Yn
      - Y1, …, Yn 是n个非终结符，且都属于NULLABLE集

NULLABLE集合算法

```
NULLABLE = {};

while (NULLABLE is still changing)
  foreach (production p: X-> β )
    if ( β == ε)
      NULLABLE ∪= {X}
    if ( β == Y1 … Yn)
      if (Y1 ∈NULLABLE && … && Yn ∈NULLABLE)
        NULLABLE ∪= {X}
```

示例

```
NULLABLE = {};

while (NULLABLE is still changing)
  foreach (production p: X-> β)
    if (β == ε)
      NULLABLE ∪= {X}
    if (β == Y1 … Yn)
      if (Y1 ∈NULLABLE && … && Yn ∈NULLABLE)
        NULLABLE ∪= {X}
```

```
Z -> d
   | X Y Z
Y -> c
   |
X -> Y
   | a
```

FIRST集合的完整计算公式

- 基于归纳的计算规则：
  - 基本情况：
    - X -> a
      - FIRST (X) ∪= {a}
  - 归纳情况：
    - X -> Y1 Y2 … Yn
      - FIRST (X) ∪= FIRST(Y1)
      - if Y1∈NULLABLE, FIRST (X) ∪= FIRST(Y2)
      - if Y1,Y2 ∈NULLABLE, FIRST(X) ∪= FIRST(Y3)
      - …

FIRST集的不动点算法

```
foreach (nonterminal N)
  FIRST(N) = {}

while(some set is changing)
  foreach (production p: N->β1 … βn)
    foreach (βi from β1 upto βn)
      if (βi== a …)
        FIRST(N) ∪= {a}
        break
      if (βi== M …)
        FIRST(N) ∪= FIRST(M)
        if (M is not in NULLABLE)
          break;
```

FIRST集计算示例

```
foreach (nonterminal N)
  FIRST(N) = {}

while(some set is changing)
  foreach (production p: N->β1 … βn)
    foreach (βi from β1 upto βn)
      if (βi== a …)
        FIRST(N) ∪= {a}
        break
      if (βi== M …)
        FIRST(N) ∪= FIRST(M)
        if (M is not in NULLABLE)
          break;
```

```
Z -> d
   | X Y Z
Y -> c
   |
X -> Y
   | a
```

N\FIRST	0	1	2
Z	{}		
Y	{}		
X	{}		

FOLLOW集的不动点算法

```
foreach (nonterminal N)
  FOLLOW(N) = {}

while(some set is changing)
  foreach (production p: N->β1 … βn)
    temp = FOLLOW(N)
    foreach (βi from βn downto β1)  // 逆序!
      if (βi== a …)
        temp = {a}
      if (βi== M …)
        FOLLOW(M) ∪= temp
        if (M is not NULLABLE)
          temp = FIRST(M)
        else temp ∪= FIRST(M)
```

FOLLOW集计算示例

```
0: Z -> d
1:    | X Y Z
2: Y -> c
3:    |
4: X -> Y
5:    | a
```

NULLABLE = {X, Y}

	X	Y	Z
FIRST	{a, c}	{c}	{a, c, d}

N\FOLLOW	0	1	2
Z	{}		
Y	{}		
X	{}		

计算FIRST_S集合

```
foreach (production p)
  FIRST_S(p) = {}

calculte_FIRST_S(production p: N->β1 … βn)
  foreach (βi from β1 to βn)
    if (βi== a …)
      FIRST_S(p) ∪= {a}
      return;
    if (βi== M …)
      FIRST_S(p) ∪= FIRST(M)
        if (M is not NULLABLE)
          return;
  FIRST_S(p) ∪= FOLLOW(N)
```

示例：构造FIRST_S集

```
0: Z -> d
1:    | X Y Z
2: Y -> c
3:    |
4: X -> Y
5:    | a
```

NULLABLE = {X, Y}

	X	Y	Z
FIRST	{a, c}	{c}	{a, c, d}
FOLLOW	{a, c, d}	{a, c, d}	{}

	0	1	2	3	4	5
FIRST_S	{d}	{a, c, d}	{c}	{a, c, d}	{c, a, d}	{a}

示例：构造LL(1)分析表

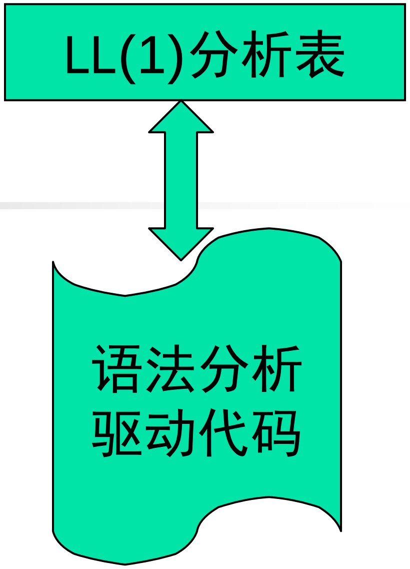
```
0: Z -> d
1:    | X Y Z
2: Y -> c
3:    |
4: X -> Y
5:    | a
```

	a	c	d
Z	1	1	0, 1
Y	3	2, 3	3
X	4, 5	4	4

	0	1	2	3	4	5
FIRST_S	{d}	{a, c, d}	{c}	{a, c, d}	{c, a, d}	{a}

LL(1)分析器

```
tokens[];    // all tokens
i=0;
stack = [S]  // S是开始符号
while (stack != [])
  if (stack[top] is a terminal t)
    if (t==tokens[i++])
      pop();
    else error(…);
  else if (stack[top] is a nonterminal T)
    pop()
    push(table[T, tokens[i]])
```



LL(1)分析表

语法分析
驱动代码