语法分析
简单的LR方法

徐 伟

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2025年09月28日

本节提纲

- 简单的LR方法（简称SLR）
  - 活前缀
  - 识别活前缀的DFA/NFA
  - SLR算法

句柄(Handles)

- **句型的句柄（可归约串）**
  - 该句型中和某产生式右部匹配的子串，并且
  - 把它归约成该产生式左部的非终结符，代表了最右推导的逆过程的一步

$$S \to aABe$$

$$A \to Abc \mid b$$

$$B \to d$$

$$S \Rightarrow_{rm} aABe \Rightarrow_{rm} aAde \Rightarrow_{rm} aAbcde \Rightarrow_{rm} abbcde$$

  - 句柄的右边仅含终结符
  - 如果文法二义，那么句柄可能不唯一

LR语法分析器的格局

- **LR语法分析的每一步都形成一个格局config**

$(s_0X_1s_1X_2s_2\ldots X_ms_m,\quad a_ia_{i+1}\ldots a_n\$)$

栈的内容 **尚未处理的输入**

  - 代表最右句型$X_1X_2\ldots X_ma_ia_{i+1}\ldots a_n$
  - $X_1X_2\ldots X_m$是最右句型的一个前缀

LR语法分析器的格局

- **LR语法分析的每一步都形成一个格局config**

  $(s_0X_1s_1X_2s_2\ldots X_ms_m,\quad a_ia_{i+1}\ldots a_n\$)$

  栈的内容　　尚未处理的输入

  - 代表最右句型$X_1X_2\ldots X_ma_ia_{i+1}\ldots a_n$
  - $X_1X_2\ldots X_m$是最右句型的一个前缀
  - 每一个前缀都对应一个状态，因此，找出所有可能在栈里出现的前缀，就可以确定所有的状态

LR语法分析器的格局

- **LR语法分析的每一步都形成一个格局config**

  $(s_0X_1s_1X_2s_2\ldots X_ms_m,\quad a_ia_{i+1}\ldots a_n\$)$

  栈的内容　　尚未处理的输入

  - 代表最右句型$X_1X_2\ldots X_ma_ia_{i+1}\ldots a_n$
  - $X_1X_2\ldots X_m$是最右句型的一个前缀
  - 每一个前缀都对应一个状态，因此，找出所有可能在栈里出现的前缀，就可以确定所有的状态
  - 状态之间的转换 <==> 前缀之间的转换

LR语法分析器的格局

- **LR语法分析的每一步都形成一个格局config**

  $(s_0X_1s_1X_2s_2\ldots X_ms_m,\quad a_ia_{i+1}\ldots a_n\$)$

  栈的内容　　尚未处理的输入

  - 代表最右句型$X_1X_2\ldots X_ma_ia_{i+1}\ldots a_n$
  - $X_1X_2\ldots X_m$是最右句型的一个前缀
  - 每一个前缀都对应一个状态，因此，找出所有可能在栈里出现的前缀，就可以确定所有的状态
  - 状态之间的转换 <==> 前缀之间的转换
  - 在栈顶为s，下一个字符为a的格局下，前缀为p
    - 何时移进？当p包含句柄的一部分且存在p′= pa
    - 何时归约？当p包含整个句柄时

LR分析: 基本概念

- **活前缀或可行前缀（viable prefix）:**
  - 最右句型的前缀，该前缀不超过最右句柄的右端

$$S \Rightarrow^*_{rm} \gamma A w \Rightarrow_{rm} \gamma \beta w$$

  - $\gamma\beta$的任何前缀（包括ε和$\gamma\beta$本身）都是活前缀
  - 都出现在栈顶

活前缀

$S \rightarrow aABe$
$A \rightarrow Abc \mid b$
$B \rightarrow d$

栈中可能出现的串：

a
ab
aA
aAb
$aAbc$
aAd
aAB
$aABe$
S

活前缀：
最右句型的前缀，该前缀不超过最右句柄的右端

$$S \Rightarrow^*_{rm} \gamma A w \Rightarrow_{rm} \gamma \beta w$$

$\gamma\beta$的任何前缀（包括ε和$\gamma\beta$本身）都是一个活前缀。

活前缀与句柄的关系

$S \to aABe$
$A \to Abc \mid b$
$B \to d$

栈中可能出现的串：

a
ab ← 出现句柄（对应$A \to b$）
aA
aAb
$aAbc$ ← 出现句柄（对应$A \to Abc$）
aAd ← 出现句柄（对应$B \to d$）
aAB
$aABe$ ← 出现句柄（对应$S \to aABe$）
S

- 活前缀已含有句柄，表明产生式A→β的右部β已出现在栈顶。

活前缀与句柄的关系

$S \rightarrow aABe$
$A \rightarrow Abc \mid b$
$B \rightarrow d$

栈中可能出现的串：

a
ab
aA ← 出现产生式$A \rightarrow Abc$右端的一部分，期望从输入串中看到bc
aAb ← 出现产生式A→Abc右端的一部分，期望从输入串中看到c
$aAbc$
aAd
aAB ← 出现产生式S→aABe的右端一部分，期望从输入串中看到e
$aABe$
S

- 活前缀已含有句柄，表明产生式A→β的右部β已出现在栈顶。
- 活前缀只含句柄的一部分符号如β_1表明$A \rightarrow \beta_1\beta_2$的右部子串$\beta_1$已出现在栈顶，当前期待从输入串中看到$\beta_2$推出的符号。

LR分析方法的特点

- 栈中的文法符号总是形成一个活前缀
- 分析表的转移函数本质上是识别活前缀的DFA

下表蓝色部分构成识别活前缀DFA的状态转换表

状态	动作						转移		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3

LR分析方法的特点

- 栈中的文法符号总是形成一个活前缀
- 分析表的转移函数本质上是识别活前缀的DFA
- 栈顶的状态符号包含确定句柄所需的一切信息

栈	输　入	动　作
0	id * id + id $	移进
…	…	…
0 *T* 2 * 7	id + id $	移进
0 *T* 2 * 7 id 5	+ id $	按*F* → id归约
0 *T* 2 * 7 *F* 10	+ id $	按*T* → *T***F*归约

LR分析方法的特点

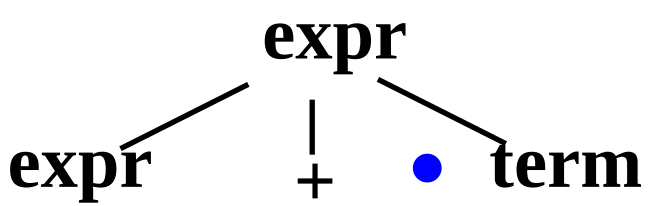
- 栈中的文法符号总是形成一个活前缀
- 分析表的转移函数本质上是识别活前缀的DFA
- 栈顶的状态符号包含确定句柄所需的一切信息
- 是已知的最一般的无回溯的移进-归约方法
- 能分析的文法类是预测分析法能分析的文法类的真超集
- 能及时发现语法错误
- 手工构造分析表的工作量太大

本节提纲

- 简单的LR方法（简称SLR）
  - 活前缀
  - 识别活前缀的DFA/NFA
  - SLR算法

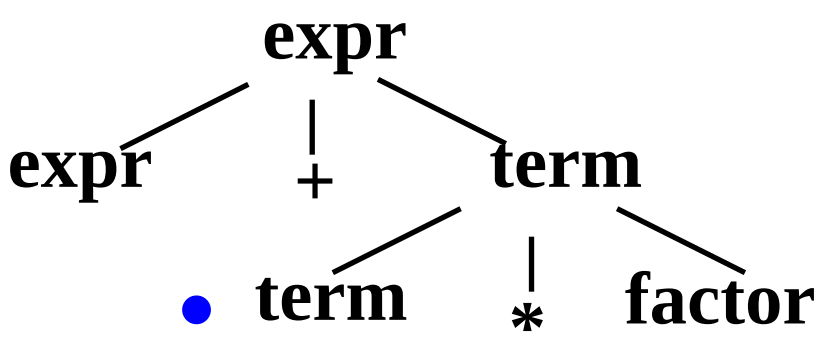
SLR分析表的构造

- **SLR (Simple LR)**
- **LR(0)项目（简称项目）**
  - 在右部的某个地方加点的产生式
  - 加点的目的是用来表示分析过程中的状态



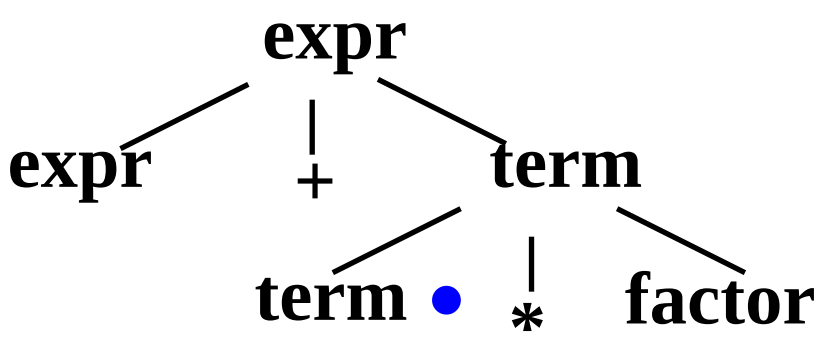
SLR分析表的构造

- **SLR (Simple LR)**
- **LR(0)项目（简称项目）**
  - 在右部的某个地方加点的产生式
  - 加点的目的是用来表示分析过程中的状态



SLR分析表的构造

- **SLR (Simple LR)**
- **LR(0)项目（简称项目）**
  - 在右部的某个地方加点的产生式
  - 加点的目的是用来表示分析过程中的状态



SLR分析表的构造

- **SLR (Simple LR)**
- **LR(0)项目（简称项目）**
  - 在右部的某个地方加点的产生式
  - 加点的目的是用来表示分析过程中的状态
- **例　$A \rightarrow XYZ$对应有四个项目**

  $A \rightarrow \cdot XYZ$

  $A \rightarrow X \cdot YZ$

  $A \rightarrow XY \cdot Z$

  $A \rightarrow XYZ \cdot$
- **例　$A \rightarrow \varepsilon$只有一个项目和它对应**

  $A \rightarrow \cdot$

项代表了一个可能的前缀

点的左边代表历史信息，点的右边代表展望信息。

SLR分析表的构造

- 从文法构造识别活前缀的DFA
- 从上述DFA构造分析表

构造识别活前缀的DFA

1. 拓（增）广文法（augmented grammar)

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid \mathbf{id}$$

构造识别活前缀的DFA

1. 拓（增）广文法（augmented grammar）

$E' \to E$

$E \to E + T \mid T$

$T \to T * F \mid F$

$F \to (E) \mid \mathrm{id}$

当且仅当分析器使用$E' \to E$归约时，宣告分析成功

构造识别活前缀的DFA

2. 构造LR(0)项目集规范族

项集族是若干可能前缀的集合，对应DFA的状态

I_0:

$E' \rightarrow \cdot E$

构造识别活前缀的DFA

2. 构造LR(0)项目集规范族

I_0:

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

求项目集的闭包closure(I)
闭包函数closure(I)
1、I的每个项目均加入closure(I)
2、如果A→α·Bβ在 closure(I)中，且B→γ是产生式，那么如果项目B→ ·γ还不在closure(I)中的话，那么把它加入。

构造识别活前缀的DFA

2. 构造LR(0)项目集规范族

I_0:

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

求项目集的闭包closure(I)
闭包函数closure(I)
1、I的每个项目均加入closure(I)
2、如果A→α·Bβ在 closure(I)中，且B→γ是产生式，那么如果项目B→ ·γ还不在closure(I)中的话，那么把它加入。

构造识别活前缀的DFA

2. 构造LR(0)项目集规范族

I_0:

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

$F \rightarrow \cdot (E)$

$F \rightarrow \cdot id$

求项目集的闭包closure(I)
闭包函数closure(I)
1、I的每个项目均加入closure(I)
2、如果A→α·Bβ在 closure(I)中，且B→γ是产生式，那么如果项目B→ ·γ还不在closure(I)中的话，那么把它加入。

构造识别活前缀的DFA

2. 构造LR(0)项目集规范族

I_0:

$E' \to \cdot E$

$E \to \cdot E + T$

$E \to \cdot T$

$T \to \cdot T * F$

$T \to \cdot F$

$F \to \cdot (E)$

$F \to \cdot \mathrm{id}$

核心项目:初始项目($E' \to \cdot E$)或者
点不在最左边的项

非核心项目：不是初始项，且点在最左边

可以通过对核心项目求闭包来获得
为节省存储空间，可省去

构造识别活前缀的DFA

2. 构造LR(0)项目集规范族

I_0:

$E' \to \cdot E$

$E \to \cdot E + T$

$E \to \cdot T$

$T \to \cdot T * F$

$T \to \cdot F$

$F \to \cdot (E)$

$F \to \cdot \mathrm{id}$

E

I_1:

$E' \to E \cdot$

$E \to E \cdot + T$

$I_1 := goto(I_0, E)$

求项目集I和文法符号X的I′ = goto(I, X)
(当输入为X时离开I状态后的转换)
1、I′ = ∅
2、对于I中的每一项A→α·Xβ,
I′ = I′ ∪ closure(A→αX·β)

构造识别活前缀的DFA

2. 构造LR(0)项目集规范族

I_0:

$E' \to \cdot E$

$E \to \cdot E + T$

$E \to \cdot T$

$T \to \cdot T * F$

$T \to \cdot F$

$F \to \cdot (E)$

$F \to \cdot \mathrm{id}$

E →

I_1:

$E' \to E\cdot$

$E \to E\cdot + T$

T →

I_2:

$E \to T\cdot$

$T \to T\cdot * F$

构造识别活前缀的DFA

2. 构造LR(0)项目集规范族

I_0:

$E' \to \cdot E$

$E \to \cdot E + T$

$E \to \cdot T$

$T \to \cdot T * F$

$T \to \cdot F$

$F \to \cdot (E)$

$F \to \cdot \mathrm{id}$

E → I_1:

$E' \to E\cdot$

$E \to E\cdot + T$

T → I_2:

$E \to T\cdot$

$T \to T\cdot * F$

F → I_3:

$T \to F\cdot$

构造识别活前缀的DFA

2. 构造LR(0)项目集规范族

I_0:

$E' \to \cdot E$

$E \to \cdot E + T$

$E \to \cdot T$

$T \to \cdot T * F$

$T \to \cdot F$

$F \to \cdot (E)$

$F \to \cdot \mathbf{id}$

$\xrightarrow{(}$

I_4:

$F \to (\cdot E)$

构造识别活前缀的DFA

2. 构造LR(0)项目集规范族

I_0:

$E' \to \cdot E$

$E \to \cdot E + T$

$E \to \cdot T$

$T \to \cdot T * F$

$T \to \cdot F$

$F \to \cdot (E)$

$F \to \cdot \mathrm{id}$

$\xrightarrow{(}$

I_4:

$F \to (\cdot E)$

$E \to \cdot E + T$

$E \to \cdot T$

$T \to \cdot T * F$

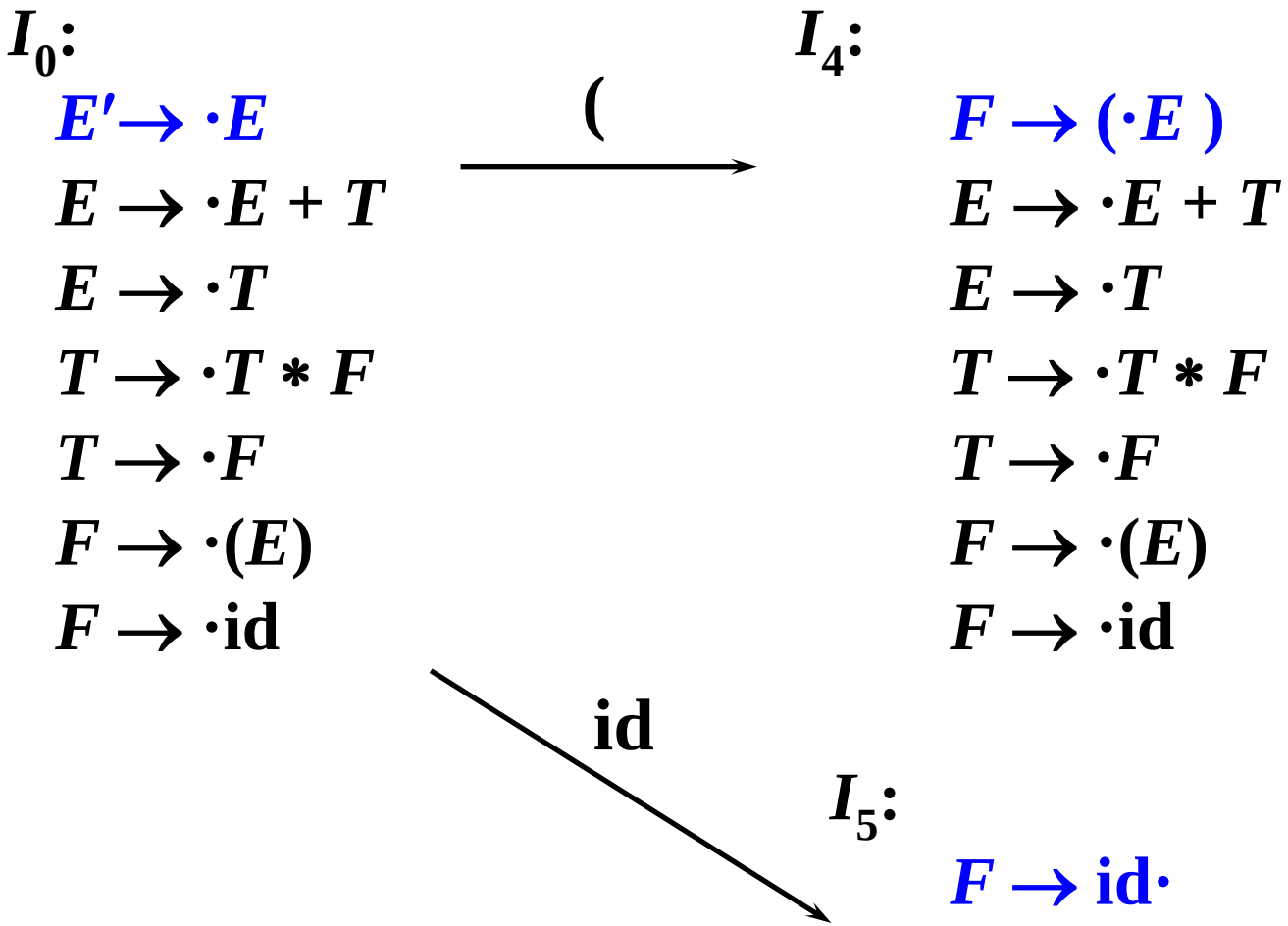
$T \to \cdot F$

$F \to \cdot (E)$

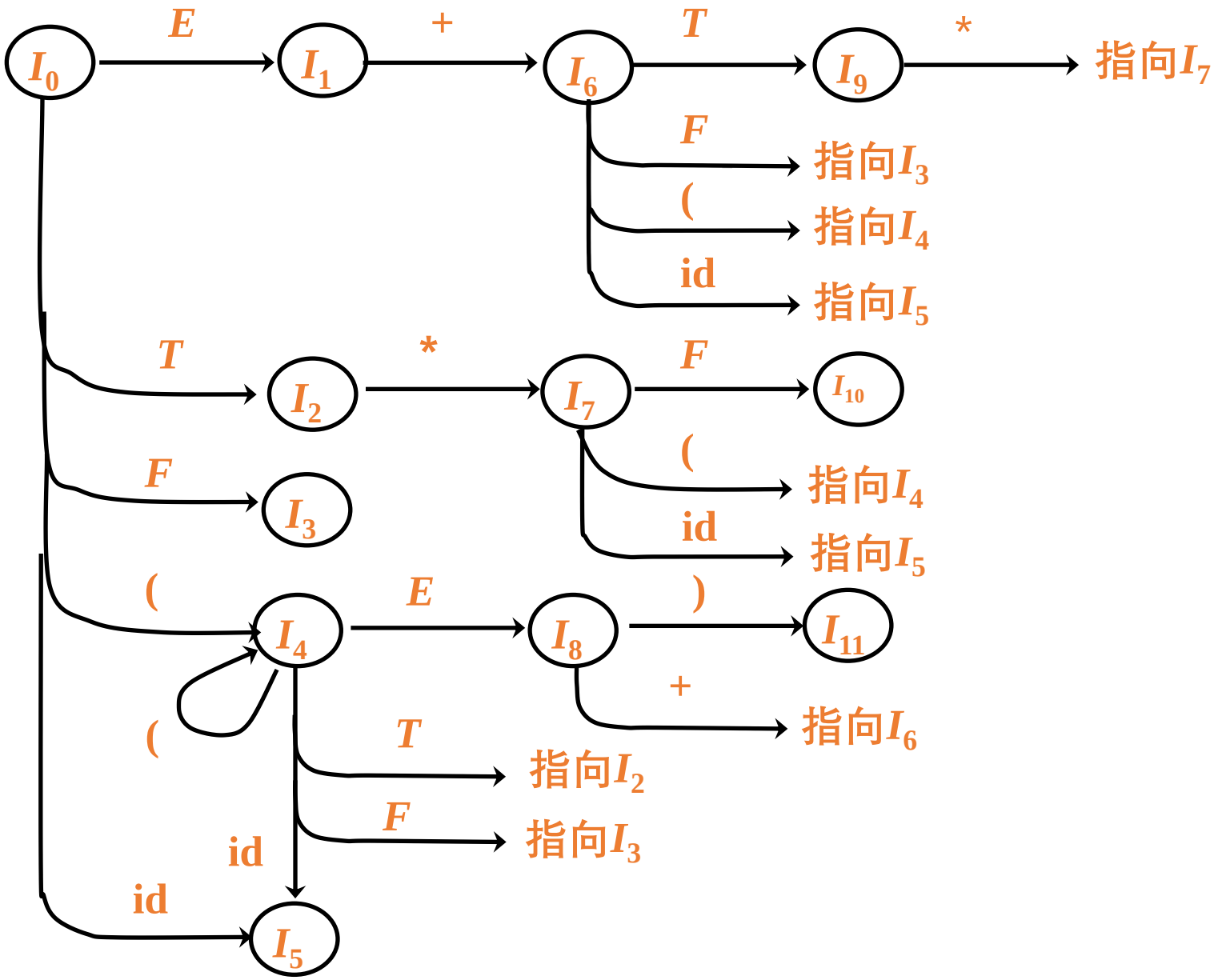
$F \to \cdot \mathrm{id}$

构造识别活前缀的DFA

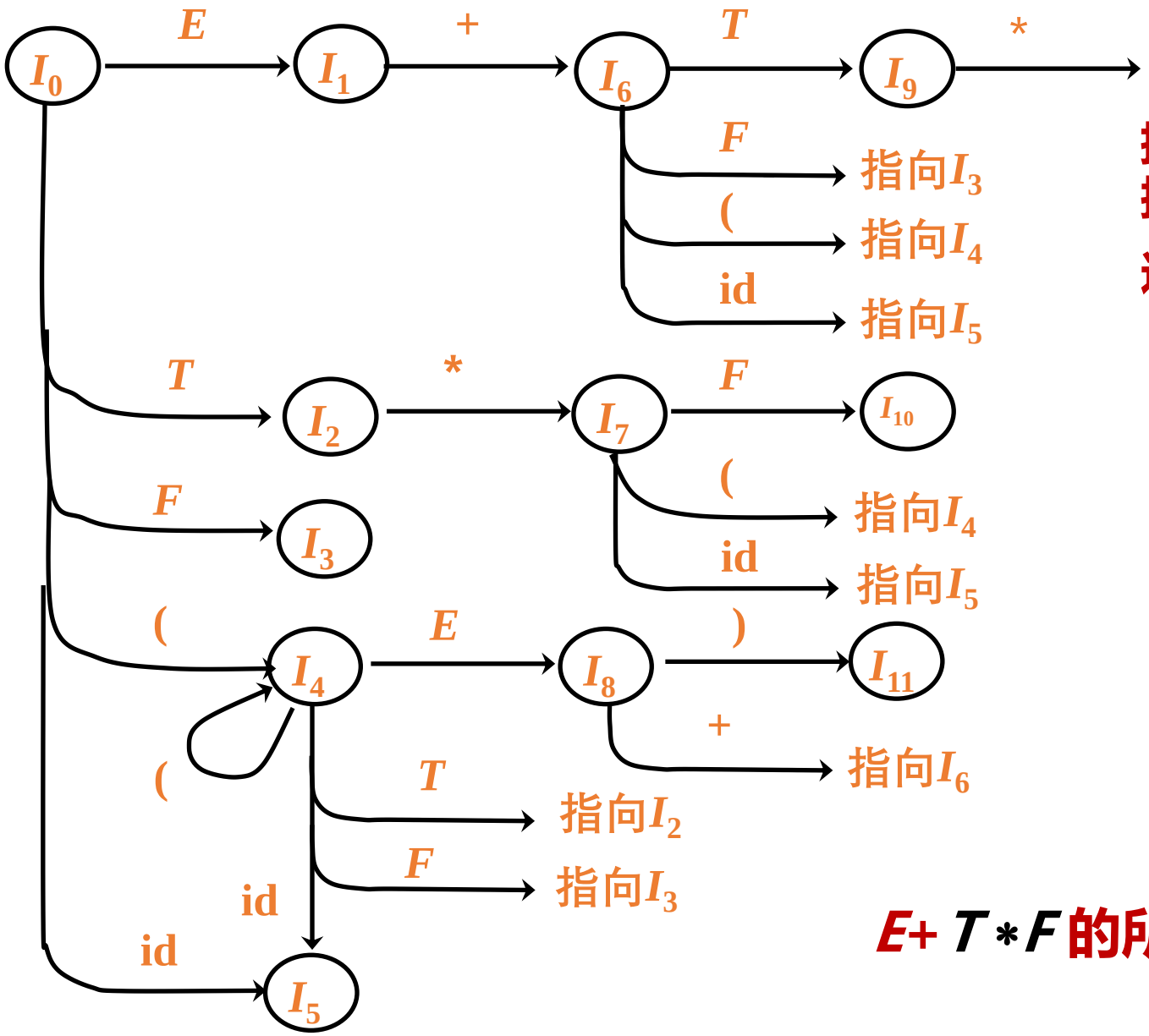
2. 构造LR(0)项目集规范族



构造识别活前缀的DFA



构造识别活前缀的DFA



把所有状态都作为接受状态

这是一个DFA

$$E' \Rightarrow E$$
$$\Rightarrow E+T$$
$$\Rightarrow E+T*F$$
$$\Rightarrow E+T*\text{id}$$
$$\Rightarrow E+T*F*\text{id}$$

$E+T*F$的所有前缀都可接受

本节提纲

- 简单的LR方法（简称SLR）
  - 活前缀
  - 识别活前缀的DFA/NFA
  - SLR算法

SLR分析表的构造

- 从文法构造识别活前缀的DFA
- 从上述DFA构造分析表

从DFA构造SLR分析表

- 状态 i 从 I_i 构造，它的 $action$ 函数如下确定：
  - 如果 $[A \to \alpha \cdot a\beta]$ 在 I_i 中，并且 $goto(I_i, a) = I_j$，那么置 $action[i, a]$ 为 sj
  - 如果 $[A \to \alpha \cdot]$ 在 I_i 中，那么对 $FOLLOW(A)$ 中的所有 a，置 $action[i, a]$ 为 rj，j 是产生式 $A \to \alpha$ 的编号
  - 如果 $[S' \to S \cdot]$ 在 I_i 中，那么置 $action[i, \$]$ 为接受 acc
  - 上面的 a 是终结符

- **如果出现动作冲突，那么该文法就不是SLR(1)文法**

从DFA构造SLR分析表

- 状态 i 从 I_i 构造，它的*action*函数如下确定：
  - 此处省略，参见上页

- 使用下面规则构造状态i的*goto*函数：
  - 对所有的非终结符A，如果$goto(I_i, A) = I_j$，那么$goto[i, A] = j$

从DFA构造SLR分析表

- 状态 i 从 I_i 构造，它的*action*函数如下确定：
  - 此处省略，参见上页
- 使用下面规则构造状态i的*goto*函数：
  - 此处省略，参见上页
- 分析器的初始状态是包含$[S' \rightarrow \cdot S]$的项目集对应的状态

不能由上面两步定义的条目都置为error

SLR分析器

例 (1) $E \rightarrow E + T$ (2) $E \rightarrow T$

(3) $T \rightarrow T * F$ (4) $T \rightarrow F$

(5) $F \rightarrow (E)$ (6) $F \rightarrow$ id

si 移进当前输入符号和状态*i*

rj 按第*j*个产生式进行归约

acc 接受

状态	动 作 action						转 移 goto		
	id	+	*	(	)	$	*E*	*T*	*F*
0	*s*5			*s*4			1	2	3
1		*s*6				*acc*			
2		*r*2	*s*7		*r*2	*r*2			
3		*r*4	*r*4		*r*4	*r*4			
4	*s*5			*s*4			8	2	3
5		*r*6	*r*6		*r*6	*r*6			
6	*s*5			*s*4				9	3

SLR(1)文法

- 一个上下文无关文法G，通过上述算法构造出SLR语法分析表，且表项中没有移进/归约或者归约/归约冲突，那么G就是SLR(1)文法。
- 1代表了当看到某个产生式右部时，只需要再向前看1个符号就可决定是否用该式进行归约。
- 通常可以省略1，写作SLR文法

SLR分析器——解释

- 例　I_2:

  (2)$E \rightarrow T\cdot$

  (3)$T \rightarrow T\cdot * F$

  - 归约：因为FOLLOW(E) = {$, +,)},

    所以 $action[2, \$]=action[2, +]=action[2,)]=r2$

  - 移进：因为圆点在中间，且点后面是终结符，

    所以，$action[2, *] = s7$

活前缀的概念-revisit

- LR(0)自动机刻画了可能出现在文法符号栈中的所有串；
- 栈中的内容一定是某个最右句型的前缀；
- 但是不是所有前缀都会出现在栈中。

$$E \Rightarrow^*_{rm} F * \mathbf{id} \Rightarrow_{rm} (E) * \mathbf{id}$$

活前缀的概念-revisit

- LR(0)自动机刻画了可能出现在文法符号栈中的所有串；
- 栈中的内容一定是某个最右句型的前缀；
- 但是不是所有前缀都会出现在栈中。

$$E \Rightarrow^*_{rm} F * \mathbf{id} \Rightarrow_{rm} (E) * \mathbf{id}$$

- 栈中只能出现(, (*E*, (*E*)，而不会出现(*E*)*
  - 因为看到*时，(*E*)是句柄，会被归约成为*F*

例 (1) $E \to E + T$ (2) $E \to T$
(3) $T \to T * F$ (4) $T \to F$
(5) $F \to (E)$ (6) $F \to \mathbf{id}$

一起努力
打造国产基础软硬件体系！

徐 伟

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2025年09月28日