



第八章 自下而上的语法分析

赵银亮 西安交通大学 2025

赵银亮

- ▶ 自上而下语法分析是对输入串的一种确定性的最左推导过程
 - 等价于一种确定性的最左扩展语法树的过程（先根遍历语法树）
 - 目标是构建LL(1)分析框架
- ▶ 自下而上语法分析是输入串的一种确定性的归约过程（称为规范归约，是最右推导的逆过程）
 - 等价于对语法树的句柄剪枝过程
- ▶ 自下而上的分析适用更广泛的文法，优点是不需要像LL分析中那么大量的文法修剪即可进行LR分析。也能容忍一些歧义性存在，如运算符优先级这种常见情形。
- ▶ 本章目标是构建SLR(1)分析框架（规范归约模拟器）

5.1 自下而上分析的基本问题

赵银亮

► 自下而上分析：

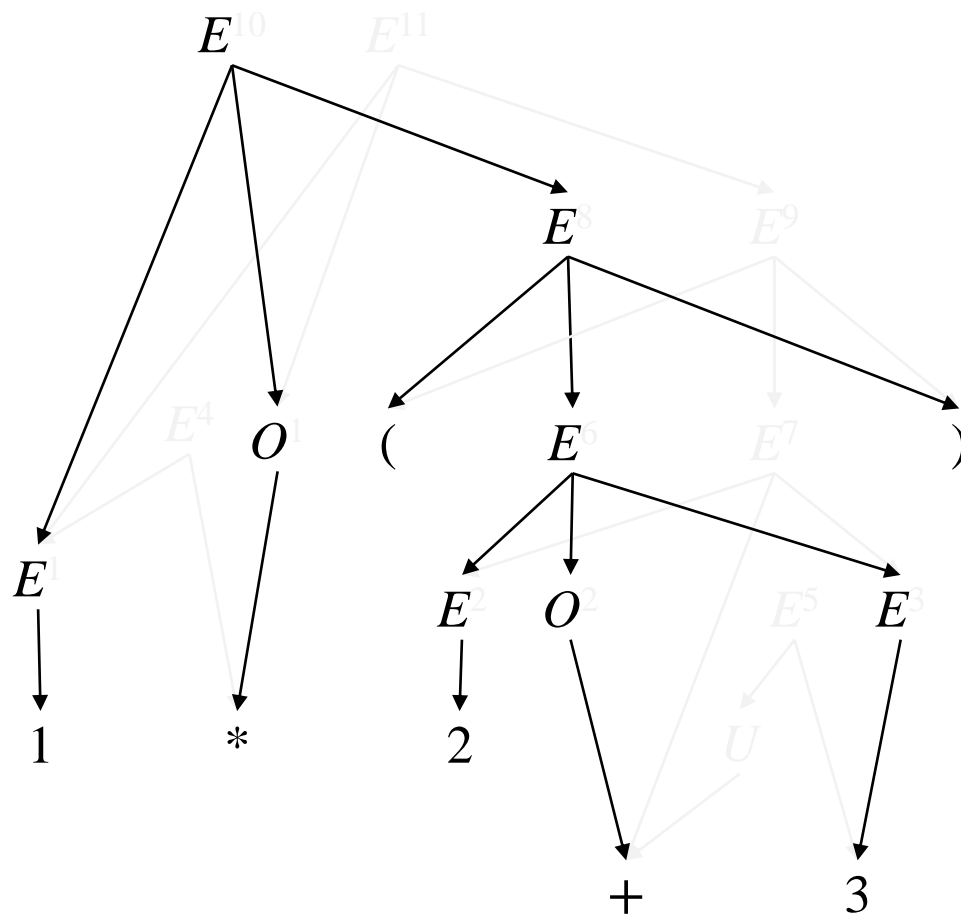
- 从输入串开始连续归约，每个归约步都是将当前符号串中某个候选式的一次出现替换成对应变元。如此连续归约，直到归约成为文法初始符号，表示分析成功。
- 分析过程中每一步直接归约都与语法树构建一一对应。

► 问题是归约过程存在不确定性。

赵银亮

示例：归约的不确定性

王冬冬



$E \rightarrow EOE$

$E \rightarrow E + E$

$E \rightarrow (E)$

$E \rightarrow E^*$

$E \rightarrow i$

$O \rightarrow +$

$O \rightarrow *$

$U \rightarrow +$

$E \rightarrow Ui$

$1^*(2+3) \Leftarrow E^*(2+3) \Leftarrow EO(2+3) \Leftarrow EO(E+3) \Leftarrow EO(EO3) \Leftarrow EO(\underline{EOE}) \Leftarrow EO(\underline{E}) \Leftarrow \underline{EOE} \Leftarrow E$

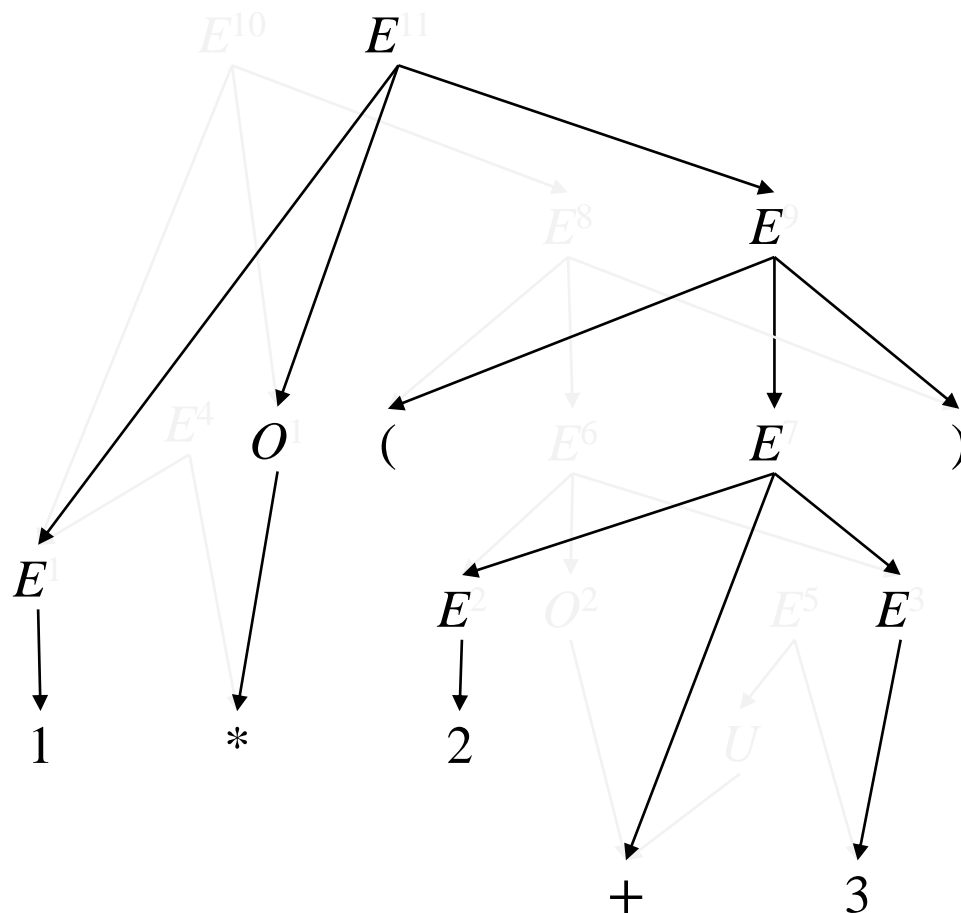
...

$1^*(2+3) \Leftarrow 1^*(2+E) \Leftarrow 1^*(\underline{EOE}) \Leftarrow 1^*(\underline{E}) \Leftarrow 1^*_E \Leftarrow \underline{EOE} \Leftarrow E$

王冬冬

示例：归约的不确定性（绪）

2018



$E \rightarrow EOE$

$E \rightarrow E+E$

$E \rightarrow (E)$

$E \rightarrow E^*$

$E \rightarrow i$

$O \rightarrow +$

$O \rightarrow *$

$U \rightarrow +$

$E \rightarrow Ui$

$1^*(2+3) \Leftarrow E^*(2+3) \Leftarrow EO(2+3) \Leftarrow EO(E+3) \Leftarrow EO(\underline{E+3}) \Leftarrow EO(\underline{E+E}) \Leftarrow EO(\underline{E}) \Leftarrow \underline{EOE} \Leftarrow E$

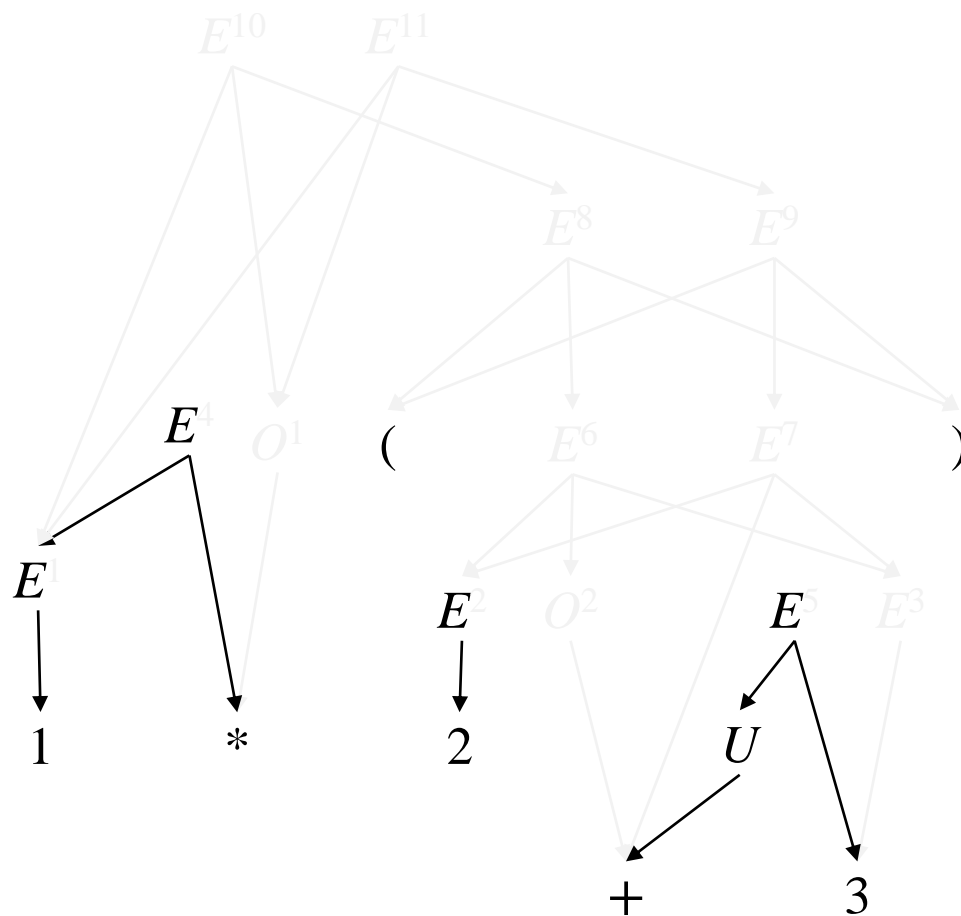
...

$1^*(2+3) \Leftarrow 1^*(\underline{2+E}) \Leftarrow 1^*(\underline{E+E}) \Leftarrow 1^*(\underline{E}) \Leftarrow 1^*_E \Leftarrow \underline{1OE} \Leftarrow \underline{EOE} \Leftarrow E$

2018

示例：归约的不确定性（绪）

王冬冬



$E \rightarrow EOE$

$E \rightarrow E + E$

$E \rightarrow (E)$

$E \rightarrow E^*$

$E \rightarrow i$

$O \rightarrow +$

$O \rightarrow *$

$U \rightarrow +$

$E \rightarrow Ui$

$\underline{1}^*(2+3) \Leftarrow \underline{E}^*(2+3) \Leftarrow E(\underline{2}+3) \Leftarrow E(E\pm 3) \Leftarrow E(E\underline{U}3) \Leftarrow E(EE) \Leftarrow$

...

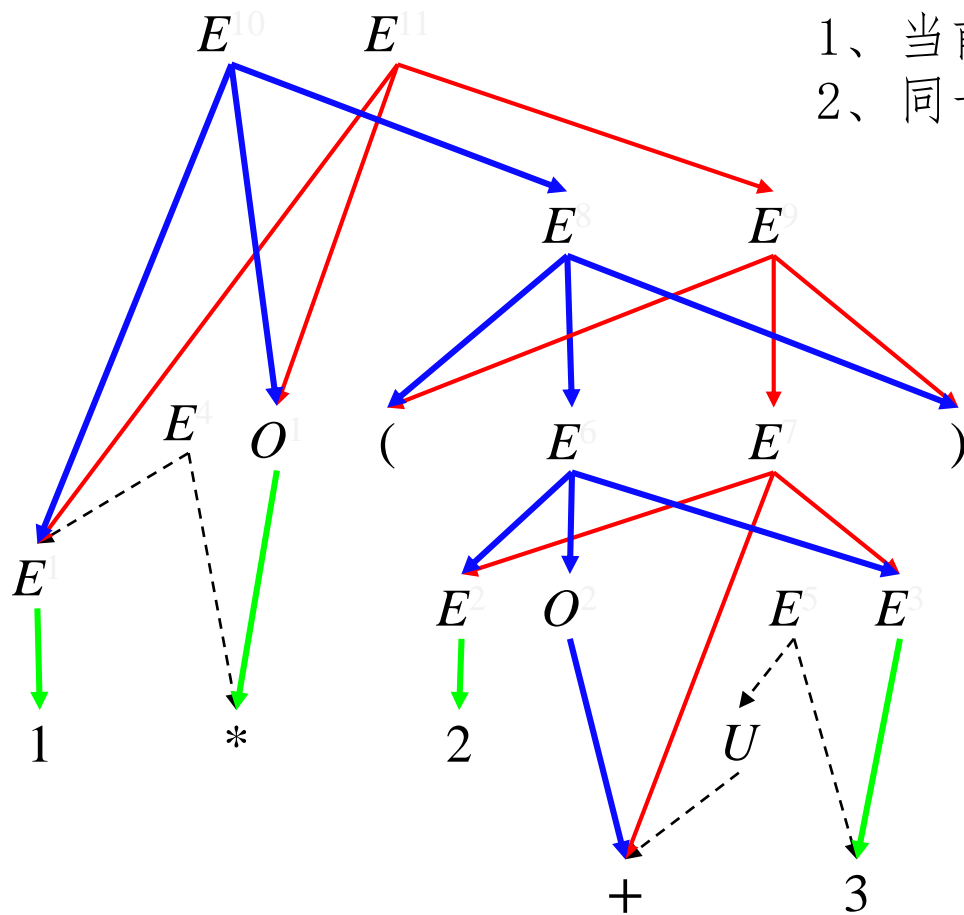
$1^*(2\pm 3) \Leftarrow 1^*(2\underline{U}3) \Leftarrow 1^*(\underline{2}E) \Leftarrow \underline{1}^*(EE) \Leftarrow \underline{E}^*(EE) \Leftarrow E(EE) \Leftarrow$

王冬冬

归纳：归约面临的问题

王元

- 1、当前归约步中有多个可归约串
- 2、同一个可归约串是多个变元的候选式



$$E \rightarrow EOE$$

$$E \rightarrow E+E$$

$$E \rightarrow (E)$$

$$E \rightarrow E^*$$

$$E \rightarrow i$$

$$O \rightarrow +$$

$$O \rightarrow *$$

$$U \rightarrow +$$

$$E \rightarrow Ui$$

王元

归约过程的不确定性

李永亮

- ▶ **定义8.1** 归约 $w \leftarrow^* \alpha \leftarrow^* \dots$ 是非确定性的，如果存在如下两种情形使得 $m * n > 1$ ：
- ▶ （情形1）归约产生的串中可能含有多个可归约串。即 α 有 n 个子串，它们都是候选式；（ α 中包含的每个候选式都被称为是一个可归约串）；
- ▶ （情形2）归约产生的串中的一个可归约串可能是多个（ m 个）变元的候选式。即 α 含有共享候选式。
- ▶ 对于 **最左归约** 而言，情形1仅限于考虑 α 有 n 个 **最左可归约串**。

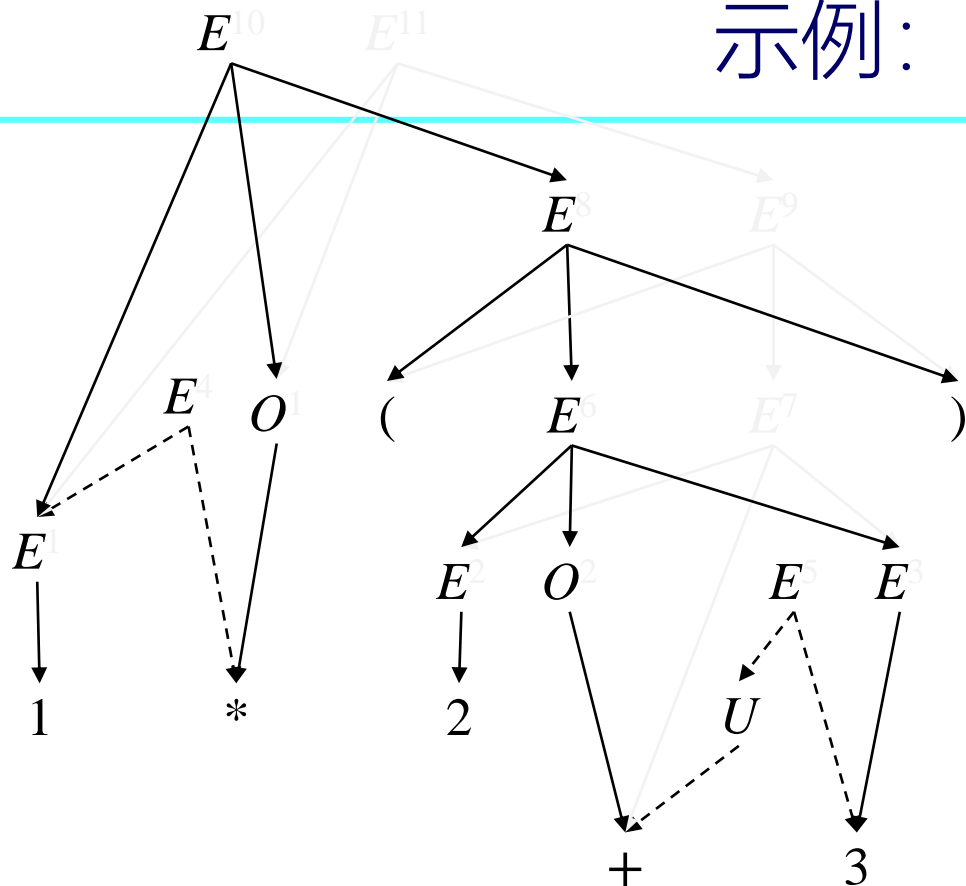
最左归约

李永亮

- ▶ $w \leftarrow_{\text{lm}}^* \alpha \leftarrow_{\text{lm}}^* \dots$
- ▶ α_1 **最左归约** 为 α_2 是指将 α_1 中的一个 **最左可归约串** 替换为它的一个变元而成为 α_2 ，写为 $\alpha_1 \leftarrow_{\text{lm}} \alpha_2$ 。（事实上是直接最左归约）
- ▶ 如果 γ_1 和 γ_2 都是 α 的最左可归约串且 $\gamma_1 \neq \gamma_2$ ，那么 $\alpha = \rho\beta$ 且 $\gamma_1, \gamma_2 \in \tau(\rho)$ 。
- ▶ 如果 γ 和 γ' 都是 α 的可归约串并且 γ 是最左可归约串，那么 $\alpha = \rho\beta$ 且 $\gamma \in \tau(\rho), \gamma' \notin \tau(\rho)$ 。
- ▶ 如果 α 只有一个可归约串 γ ，那么它是最左可归约串。

李永亮

示例：最左归约的不确定性



$E \rightarrow EOE$

$E \rightarrow E+E$

$E \rightarrow (E)$

$E \rightarrow E^*$

$E \rightarrow i$

$O \rightarrow +$

$O \rightarrow *$

$U \rightarrow +$

$E \rightarrow Ui$

E^* 和 $*$ 都是
 $E^*(2+3)$ 的
最左可归约串

$+$ 是 $EO(E+3)$ 的
最左可归约串，
是 O 和 U 的
共享候选式

$EO(E+3) \leftarrow EO(E+E)$
不是最左归约

$1^*(2+3) \leftarrow \underline{E}^*(2+3) \leftarrow EO(\underline{2}+3) \leftarrow EO(E+3) \leftarrow EO(E\underline{O}3) \leftarrow EO(\underline{EOE}) \leftarrow EO(\underline{E}) \leftarrow \underline{EOE} \leftarrow E$

$1^*(2+3) \leftarrow E^*(2+3) \leftarrow EO(\underline{2}+3) \leftarrow EO(E+3) \leftarrow EO(E\underline{U}3) \leftarrow EO(\underline{EE}) \neq$

$1^*(2+3) \leftarrow \underline{E}^*(2+3) \leftarrow E(\underline{2}+3) \leftarrow E(E+3) \leftarrow E(E\underline{O}3) \leftarrow E(\underline{EOE}) \leftarrow E(\underline{E}) \leftarrow EE \neq$

$1^*(2+3) \leftarrow \underline{E}^*(2+3) \leftarrow E(\underline{2}+3) \leftarrow E(E+3) \leftarrow E(E\underline{O}3) \leftarrow E(\underline{EOE}) \leftarrow E(\underline{E}) \leftarrow EE \neq$

$1^*(2+3) \leftarrow \underline{E}^*(2+3) \leftarrow E(\underline{2}+3) \leftarrow E(E+3) \leftarrow E(E\underline{U}3) \leftarrow E(\underline{EUE}) \neq$

$1^*(2+3) \leftarrow E^*(2+3) \leftarrow EO(\underline{2}+3) \leftarrow \underline{EO(E+3)} \leftarrow \underline{EO(E+E)} \leftarrow EO(E) \leftarrow EOE \leftarrow E$ 不完备

最左归约的不确定性

王冬冬

► 定义8.3 最左归约

$w \leftarrow_{\text{lm}}^* \rho x \leftarrow_{\text{lm}}^* \dots$, s.t. $\rho = \delta\gamma\beta \wedge \gamma \in \text{ran}(\mathcal{P}) \rightarrow \beta = \varepsilon, x \in \tau(w)$,
是非确定性的过程，如果下列两个情形中 $n*m > 1$:

- (情形1) ρ 有 n 个后缀都是可归约串，形式地，
 $|\tau(\rho) \cap \text{ran}(\mathcal{P})| = n$
- (情形2) 有一个由 m 个变元共享的候选式是 ρ 的可归约串，
形式地， $\exists \gamma \in (\tau(\rho) \cap \text{ran}(\mathcal{P})) \cdot |\{A : (A, \gamma) \in \mathcal{P}\}| = m$

$\rho = \delta\gamma\beta \wedge \gamma \in \text{ran}(\mathcal{P}) \rightarrow \beta = \varepsilon$ 表示 ρ 中每个可归约串都只能是它的后缀。而且这些可归约串都是最左的，这是由 ρ 处在最左归约过程中这一事实来保证。

这是基于栈的过程

基于栈的归约、移进-归约分析框架

王冬冬

- ▶ 利用栈来完成语法分析任务 $\varepsilon \cdot w \leftarrow_{lm}^* \rho \cdot ay \leftarrow_{lm}^* S \cdot \varepsilon$
 - 若 ρ 有最左可归约串 γ 则有 $\rho \cdot ay = \alpha \gamma \cdot ay \leftarrow_{lm} \alpha A \cdot ay$
 - 否则, $\rho \cdot ay \leftarrow_{lm}^* \rho a \cdot y$
 - 若把栈设为逻辑空的话, $Z_0 \cdot w \# \leftarrow_{lm}^* Z_0 \rho \cdot ay \# \leftarrow_{lm}^* Z_0 S \cdot \#$
- ▶ 从而形成**移进-归约** (shift-reduce) 分析框架:
 - 初始化栈为 Z_0 , 剩余串为 $w \#$, 形成初始归约串 $\varepsilon \cdot w$ 。
 - 过程中对于归约串 $\rho \cdot ay$, 若有 $\gamma \in \tau(\rho)$ 且 $(A, \gamma) \in \mathcal{P}$, 则 $\rho = \alpha \gamma$ 被更新为 αA 并形成一步直接**归约** $\alpha \gamma \cdot ay \leftarrow_{lm} \alpha A \cdot ay$
 - 过程中对于 $\rho \cdot ay$, 形成0步归约 $\rho \cdot ay \leftarrow_{lm}^* \rho a \cdot y$ (**移进**)
 - 过程进行到 $Z_0 S \cdot \#$ 时断言接受之, 否则有语法错误。

$1^*(2+3) \leftarrow E^*(2+3) \leftarrow EO(2+3) \leftarrow EO(E+3) \leftarrow EO(E+E) \leftarrow EO(E) \leftarrow EOE \leftarrow E$

$EO(2+3) \leftarrow EO(E \pm \cdot 3) \leftarrow EO(E + \underline{3} \cdot) \leftarrow EO(\underline{E+E} \cdot) \leftarrow EO(E)$ 无不完备⁸

示例：移进-归约分析过程

2018

步骤	符号栈	剩余输入串	分析动作
0	Z_0	$i*(i+i)\#$	shift
1	Z_0i	$*(i+i)\#$	reduce: $E \rightarrow i$; 还是shift ?
2	Z_0E	$*(i+i)\#$	shift
3	Z_0E^*	$(i+i)\#$	reduce: $O \rightarrow *$; $E \rightarrow E^*$; 还是shift ?
4	Z_0EO	$(i+i)\#$	shift
5	$Z_0EO($	$i+i)\#$	shift
6	$Z_0EO(i$	$+i)\#$	reduce: $E \rightarrow i$; 还是shift ?
7	$Z_0EO(E$	$+i)\#$	shift
8	$Z_0EO(E+$	$i)\#$	reduce: $O \rightarrow +$; $U \rightarrow +$; 还是shift ?
9	$Z_0EO(E+i$	$)\#$	reduce: $E \rightarrow i$; 还是shift ?
10	$Z_0EO(E+E$	$)\#$	reduce: $E \rightarrow E+E$; 还是shift ?
11	$Z_0EO(E$	$)\#$	shift
12	$Z_0EO(E)$	$\#$	reduce: $E \rightarrow (E)$; 还是shift ?
13	Z_0EOE	$\#$	reduce: $E \rightarrow EOE$; 还是shift ?
14	Z_0E	$\#$	acc

$E \rightarrow EOE$

$E \rightarrow E+E$

$E \rightarrow (E)$

$E \rightarrow E^*$

$E \rightarrow i$

$O \rightarrow +$

$O \rightarrow *$

$U \rightarrow +$

$E \rightarrow Ui$

李永强

规范归约

王

- ▶ 最右推导的逆过程是确定化的，被称为规范归约。
- ▶ 最左归约虽然与规范归约不相同，但支撑基于栈的分析框架
- ▶ 移进-归约框架是一种基于栈的分析框架，允许无条件移进。
- ▶ 展望：在移进-归约框架上可模拟规范归约。

$1^*(2+3) \leftarrow \underline{E}^*(2+3) \leftarrow EO(\underline{2}+3) \leftarrow EO(E\underline{+}3) \leftarrow EO(E\underline{O}3) \leftarrow EO(\underline{EOE}) \leftarrow EO(\underline{E}) \leftarrow \underline{EOE} \leftarrow E$

$1^*(2+3) \leftarrow \underline{E}^*(2+3) \leftarrow EO(\underline{2}+3) \leftarrow EO(E\underline{+}3) \leftarrow EO(E\underline{U}3) \leftarrow EO(\underline{EE}) \neq$

$1^*(2+3) \leftarrow \underline{E}^*(2+3) \leftarrow E(\underline{2}+3) \leftarrow E(E\underline{+}3) \leftarrow E(E\underline{O}3) \leftarrow E(\underline{EOE}) \leftarrow E(\underline{E}) \leftarrow \underline{EE} \neq$

$1^*(2+3) \leftarrow \underline{E}^*(2+3) \leftarrow E(\underline{2}+3) \leftarrow E(E\underline{+}3) \leftarrow E(E\underline{O}3) \leftarrow E(\underline{EOE}) \leftarrow E(\underline{E}) \leftarrow \underline{EE} \neq$

$1^*(2+3) \leftarrow \underline{E}^*(2+3) \leftarrow E(\underline{2}+3) \leftarrow E(E\underline{+}3) \leftarrow E(E\underline{U}3) \leftarrow E(\underline{EE}) \neq$

$1^*(2+3) \leftarrow \underline{E}^*(2+3) \leftarrow EO(\underline{2}+3) \leftarrow \underline{EO(E+3)} \leftarrow \underline{EO(E+E)} \leftarrow EO(E) \leftarrow EOE \leftarrow E$

- ▶ “归约-归约”冲突：栈顶有多个可归约串，选哪个归约？
- ▶ “移进-归约”冲突：栈顶有可归约串，归约呢还是移进？
- ▶ 栈顶没有可归约串，只须移进即可。

王

对示例文法的讨论

► 对于有歧义性的文法进行处理:

- EOE 与 $E+E$ 有一定重复, 导致文法有歧义性, 去除 $E+E$, 但是不如将 O 省去更优
- 单目运算符 $*$ 的结合性与其他运算不一致, 去除 E^* , 但或许有别的用途, 重新设计
- 让单目运算符 $+$ 存在的话, 就让 i 为无符号整数, 当然还应该有单目运算符 $-$, 这样的话采用 UE 取代 Ui , 并使得单目 $+$ 的优先级高于双目 $+$ 。

- 剩下与运算符的结合性、优先级有关的歧义性未消除

$$E \rightarrow E+E$$

$$E \rightarrow E^*E$$

$$E \rightarrow (E)$$

$$E \rightarrow i$$

$$U \rightarrow +$$

$$E \rightarrow UE$$

$$E \rightarrow EOE$$

$$E \rightarrow E+E$$

$$E \rightarrow (E)$$

$$E \rightarrow E^*$$

$$E \rightarrow i$$

$$O \rightarrow +$$

$$O \rightarrow *$$

$$U \rightarrow +$$

$$E \rightarrow Ui$$

$$\underline{1}^*(2+3) \leftarrow E^*(2+3) \leftarrow EO(\underline{2}+3) \leftarrow EO(E\pm 3) \leftarrow EO(E\underline{O}3) \leftarrow EO(\underline{EOE}) \leftarrow EO(\underline{E}) \leftarrow \underline{EOE} \leftarrow E$$

$$1^*(2+3) \leftarrow E^*(2+3) \leftarrow EO(2+3) \leftarrow \underline{EO(E+3)} \leftarrow \underline{EO(E+E)} \leftarrow EO(E) \leftarrow EOE \leftarrow E$$

Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - Expr$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

Stack

Input String

num	*	(num	+	num)
-----	---	---	-----	---	-----	---

Shift-Reduce Parser Example

$Expr \rightarrow Expr\ Op\ Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - Expr$

$Expr \rightarrow \text{num}$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

num	*	(num	+	num)
-----	---	---	-----	---	-----	---

Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - Expr$

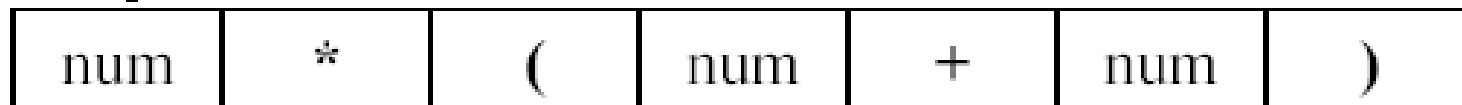
$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

SHIFT



num	*	(num	+	num)
-----	---	---	-----	---	-----	---

Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - Expr$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

num

SHIFT

*	(num	+	num)
---	---	-----	---	-----	---

Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - Expr$

$Expr \rightarrow \text{num}$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

num

REDUCE

*	(num	+	num)
---	---	-----	---	-----	---

Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - Expr$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

Expr

REDUCE

num

*

(

num

+

num

)

Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - Expr$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

SHIFT

$Expr$

num

*

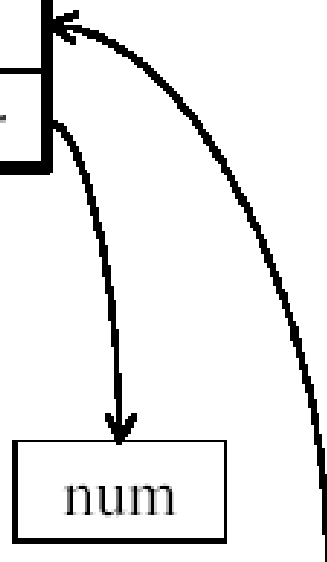
(

num

+

num

)



Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - Expr$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

SHIFT

num

(num	+	num)
---	-----	---	-----	---

Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

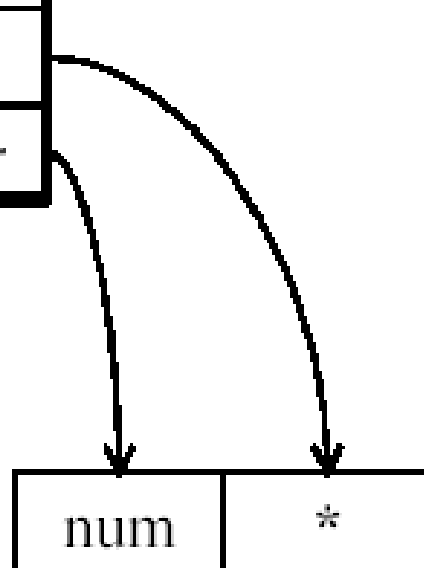
$Expr \rightarrow - Expr$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$



(num	+	num)
---	-----	---	-----	---

Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - Expr$

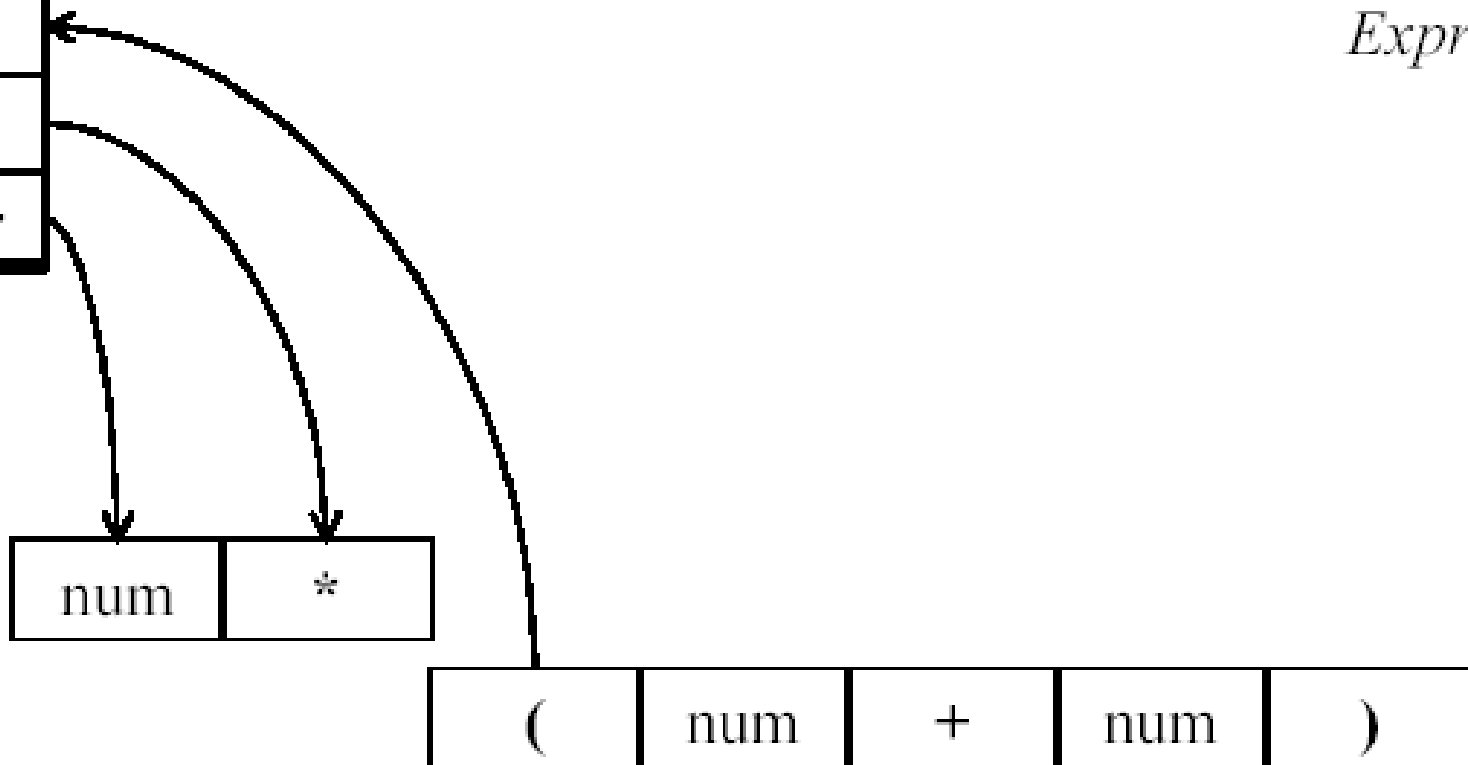
$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

SHIFT



Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - Expr$

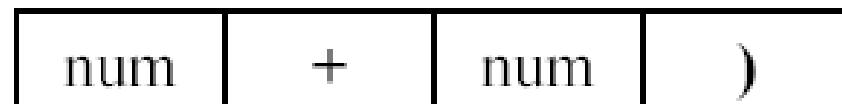
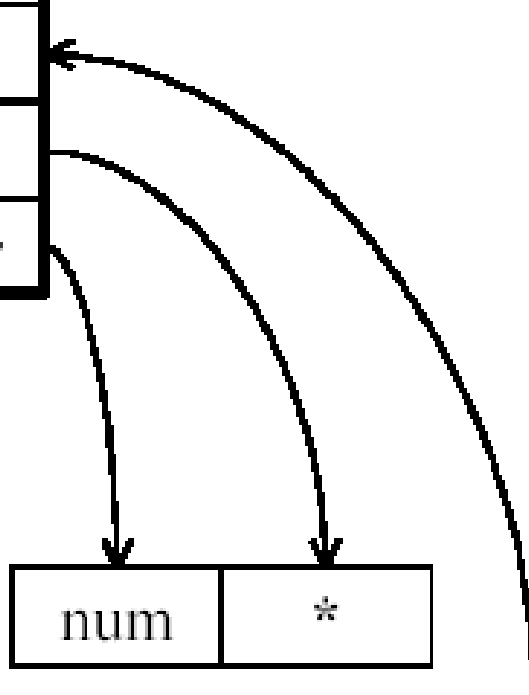
$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

SHIFT



Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - Expr$

$Expr \rightarrow num$

$Op \rightarrow +$

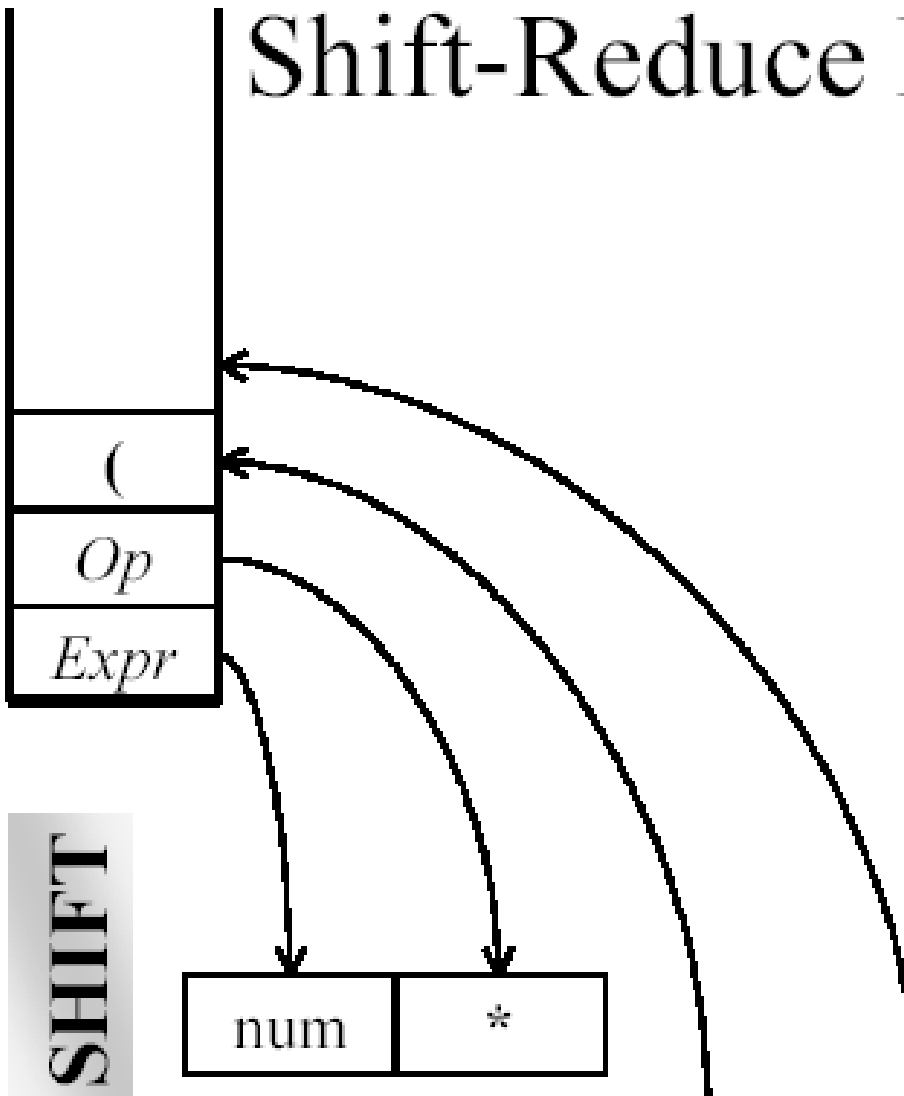
$Op \rightarrow -$

$Op \rightarrow *$

SHIFT

num	*
-----	---

num	+	num)
-----	---	-----	---



Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

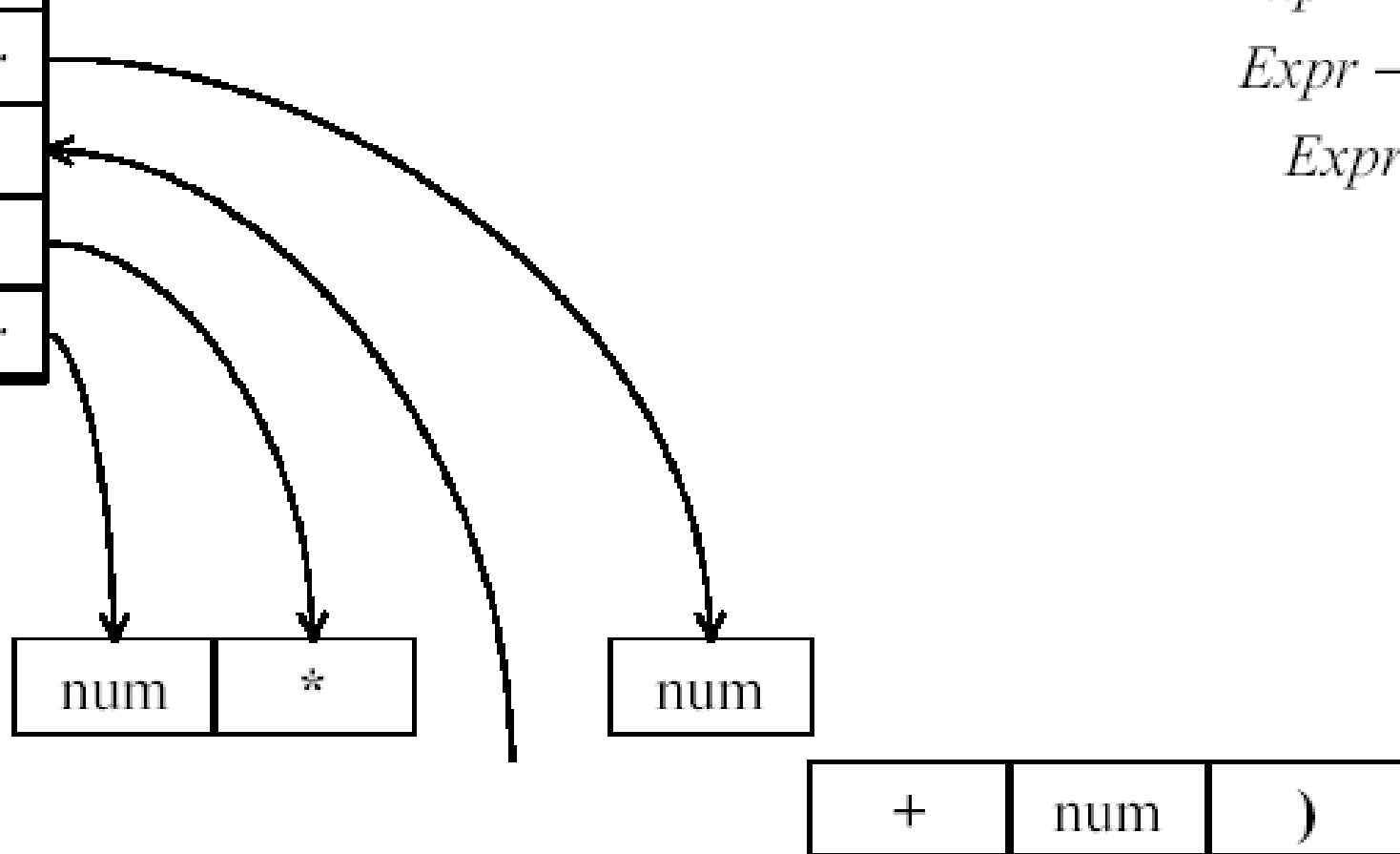
$Expr \rightarrow - Expr$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$



Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - Expr$

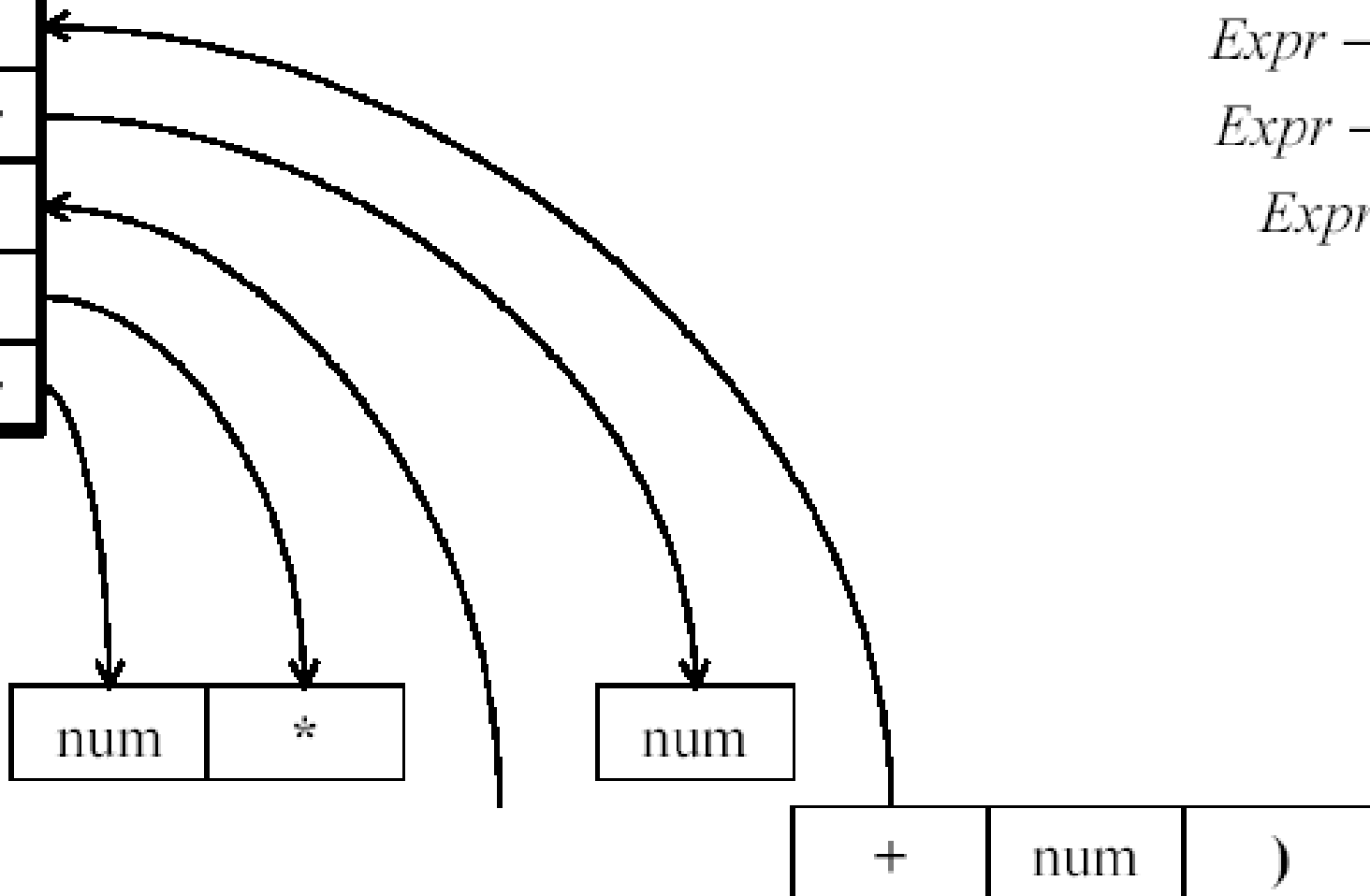
$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

SHIFT



Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - Expr$

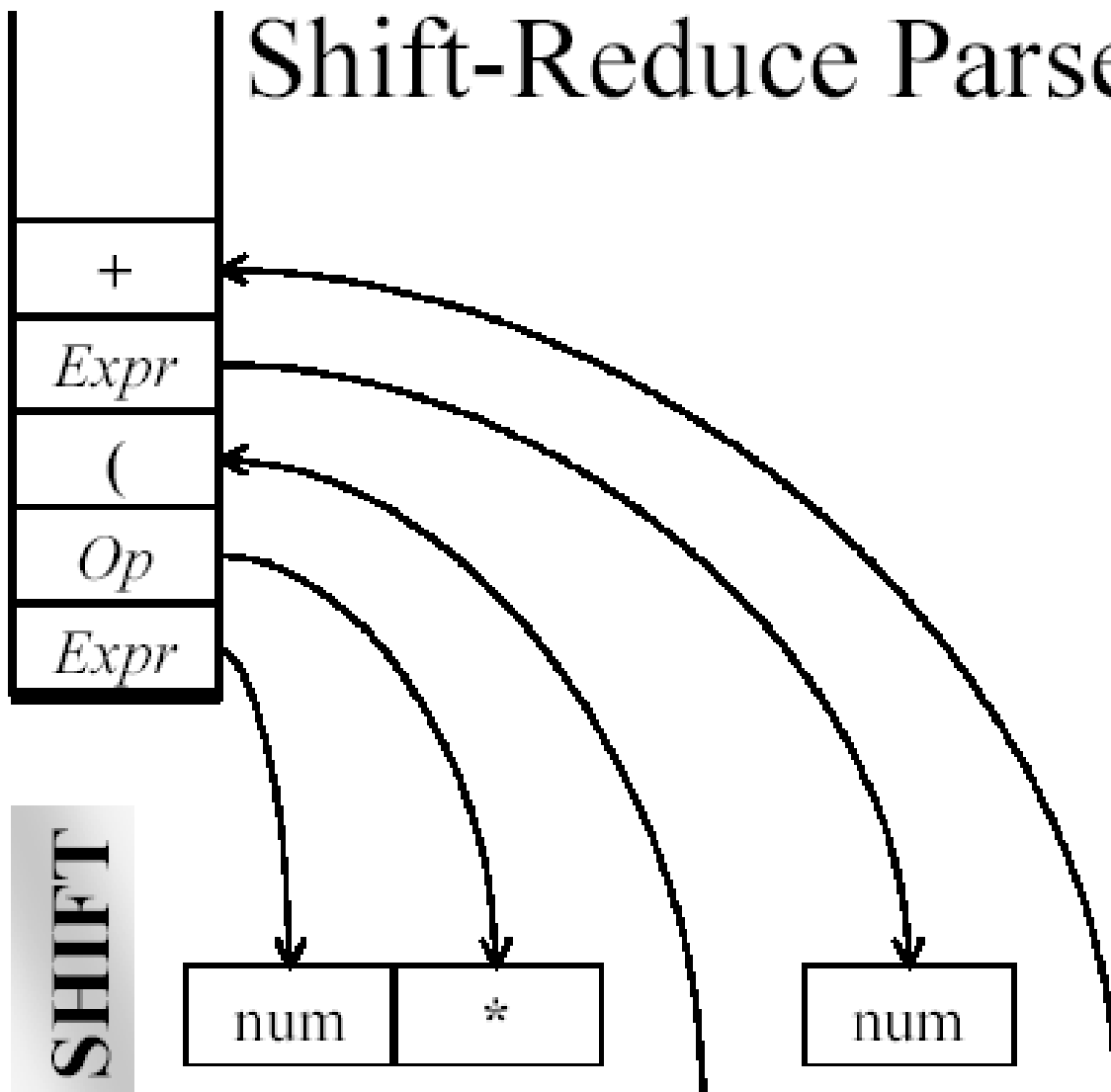
$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

SHIFT



num)
-----	---

Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

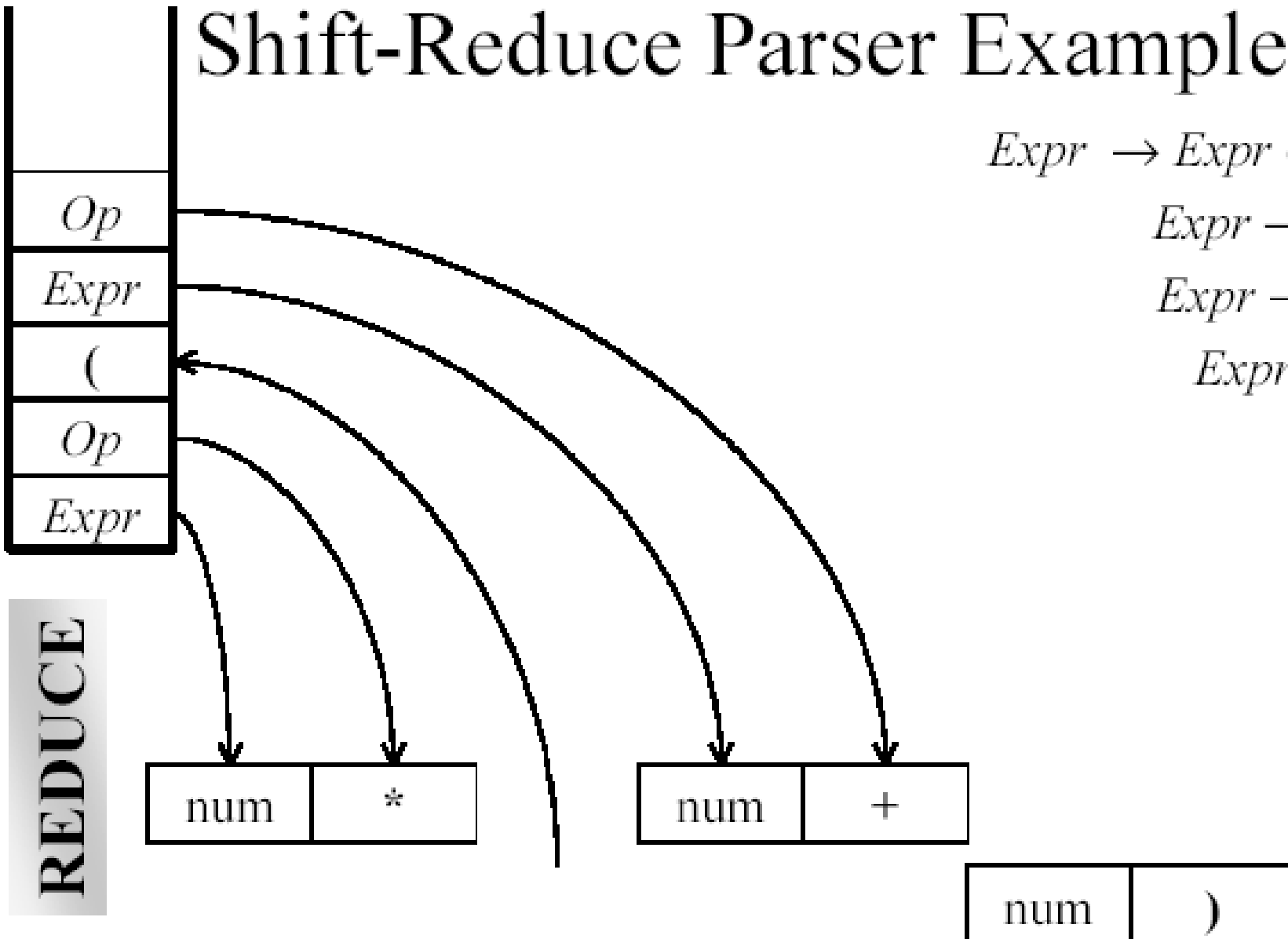
$Expr \rightarrow - Expr$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$



Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - Expr$

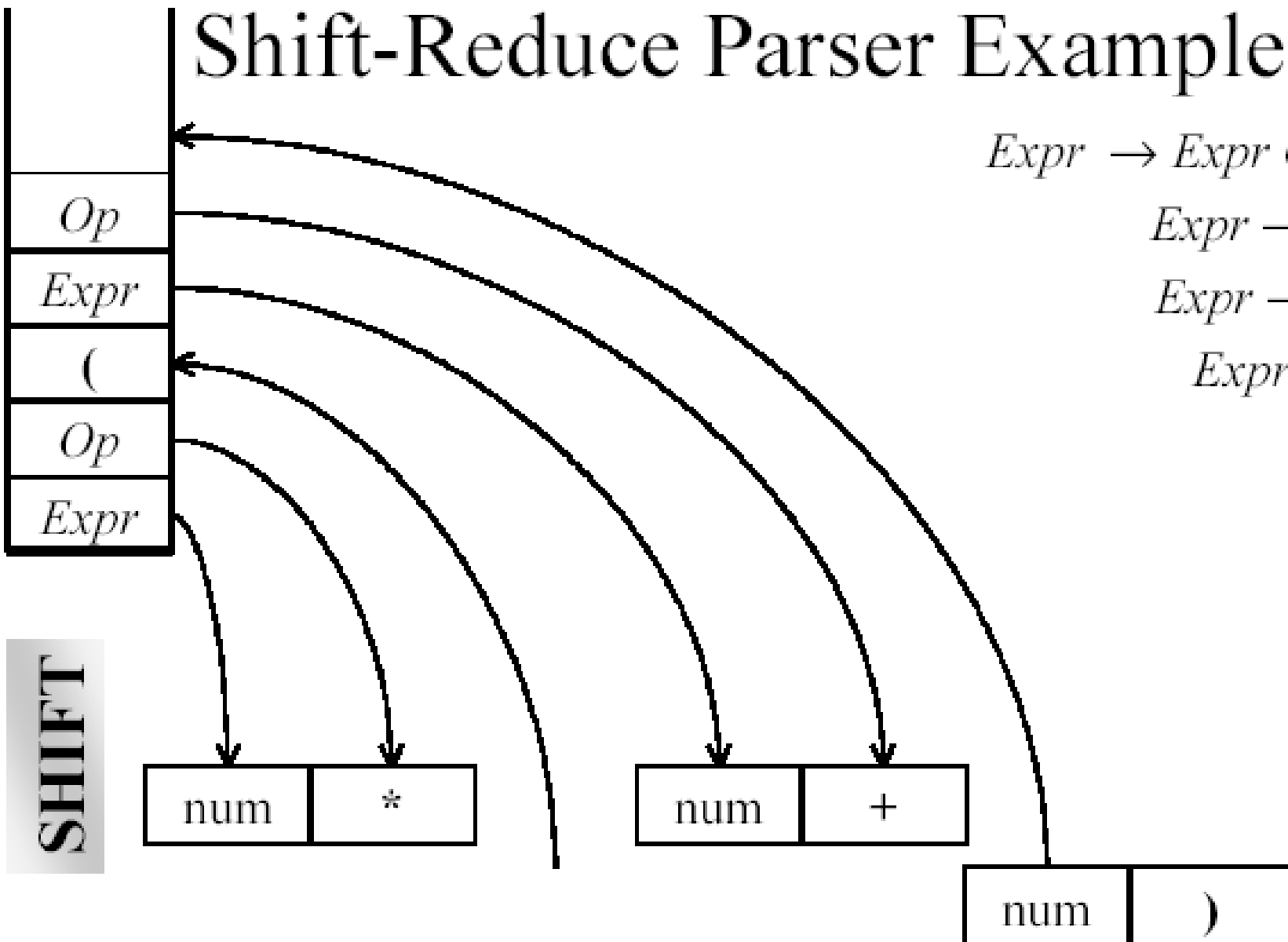
$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

SHIFT



Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - Expr$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

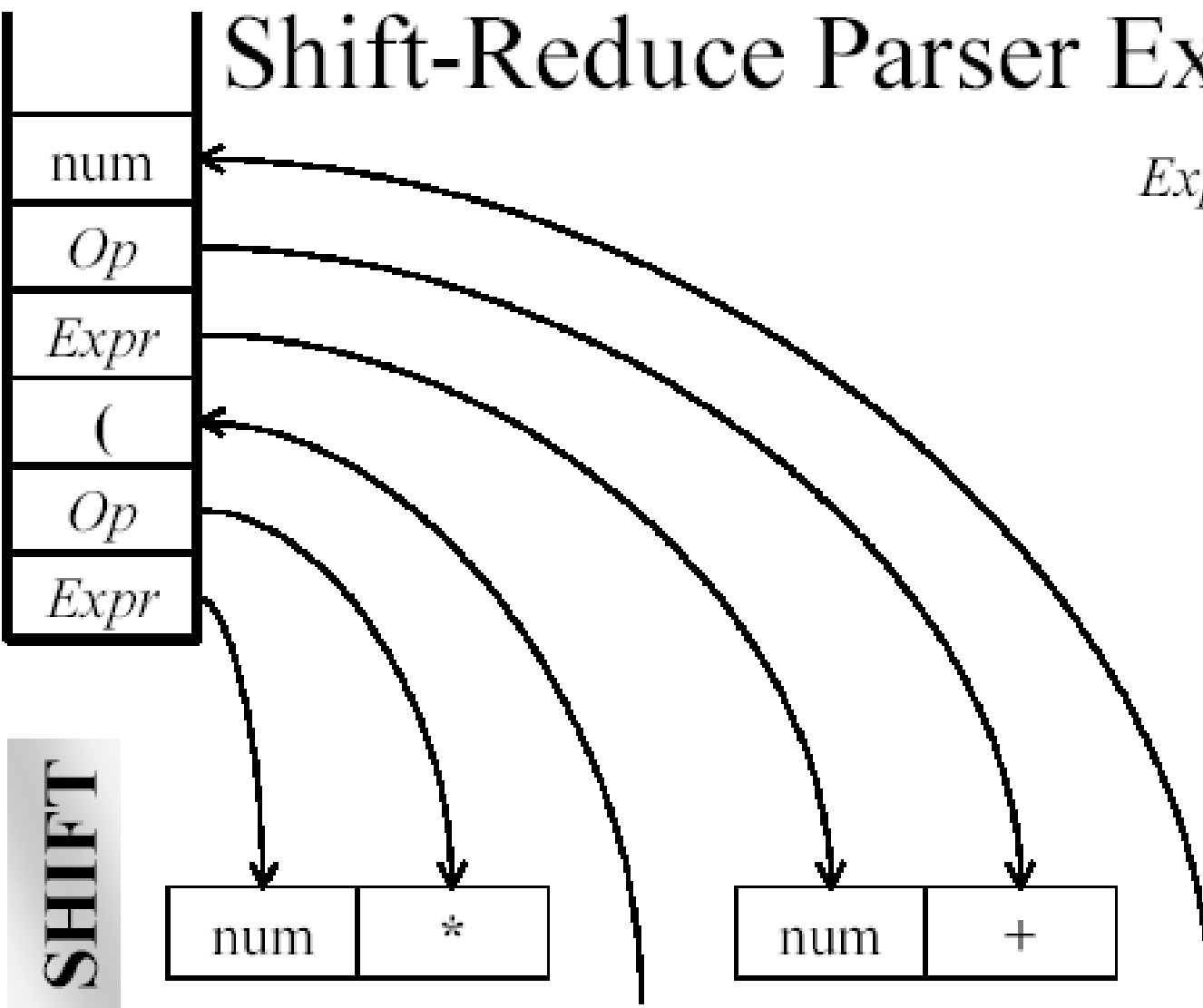
$Op \rightarrow *$

SHIFT

num	*
-----	---

num	+
-----	---

)



Shift-Reduce Parser Example

Expr

Op

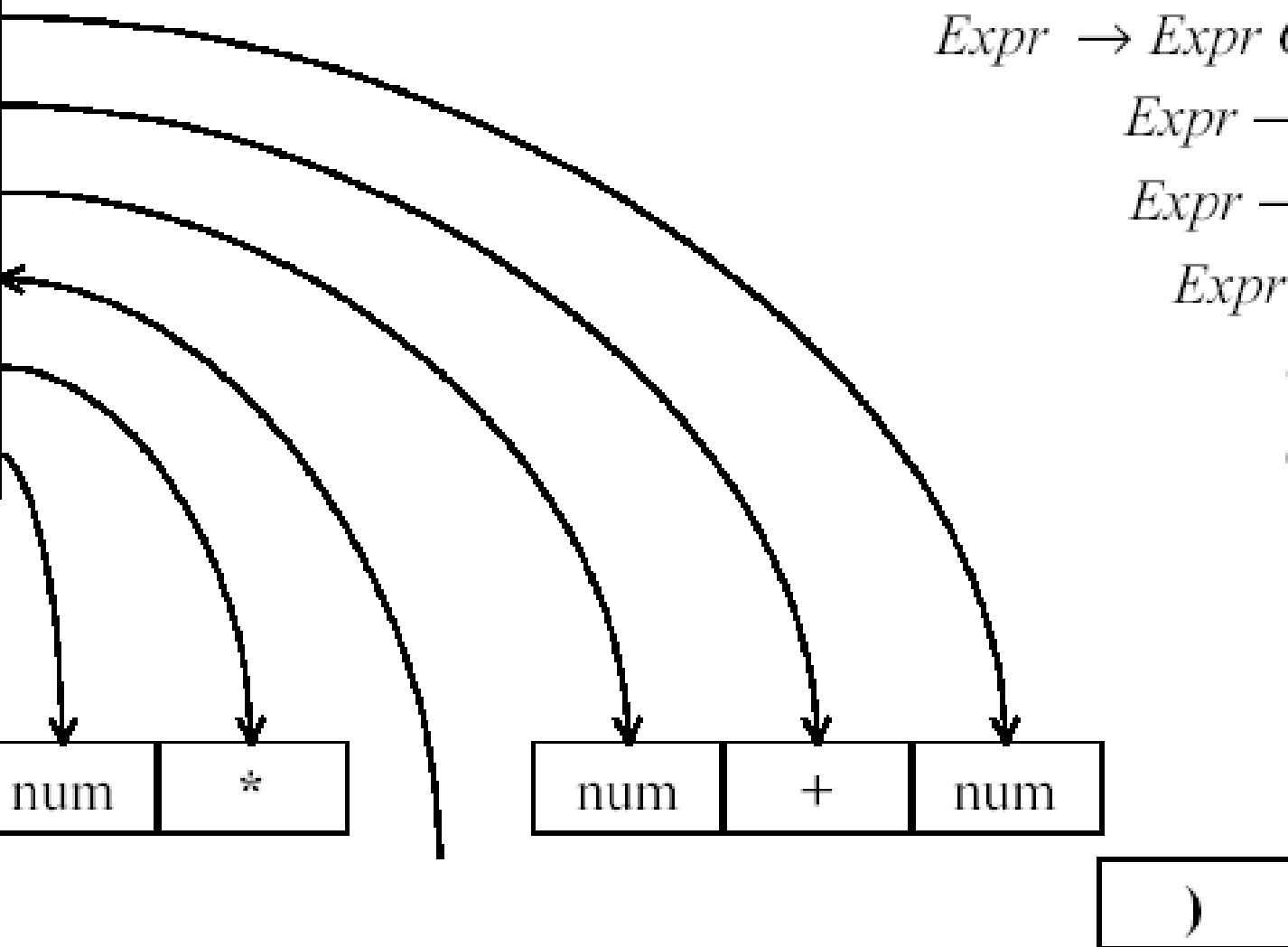
Expr

(

Op

Expr

REDUCE



$Expr \rightarrow Expr \ Op \ Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - \ Expr$

$Expr \rightarrow \text{num}$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - Expr$

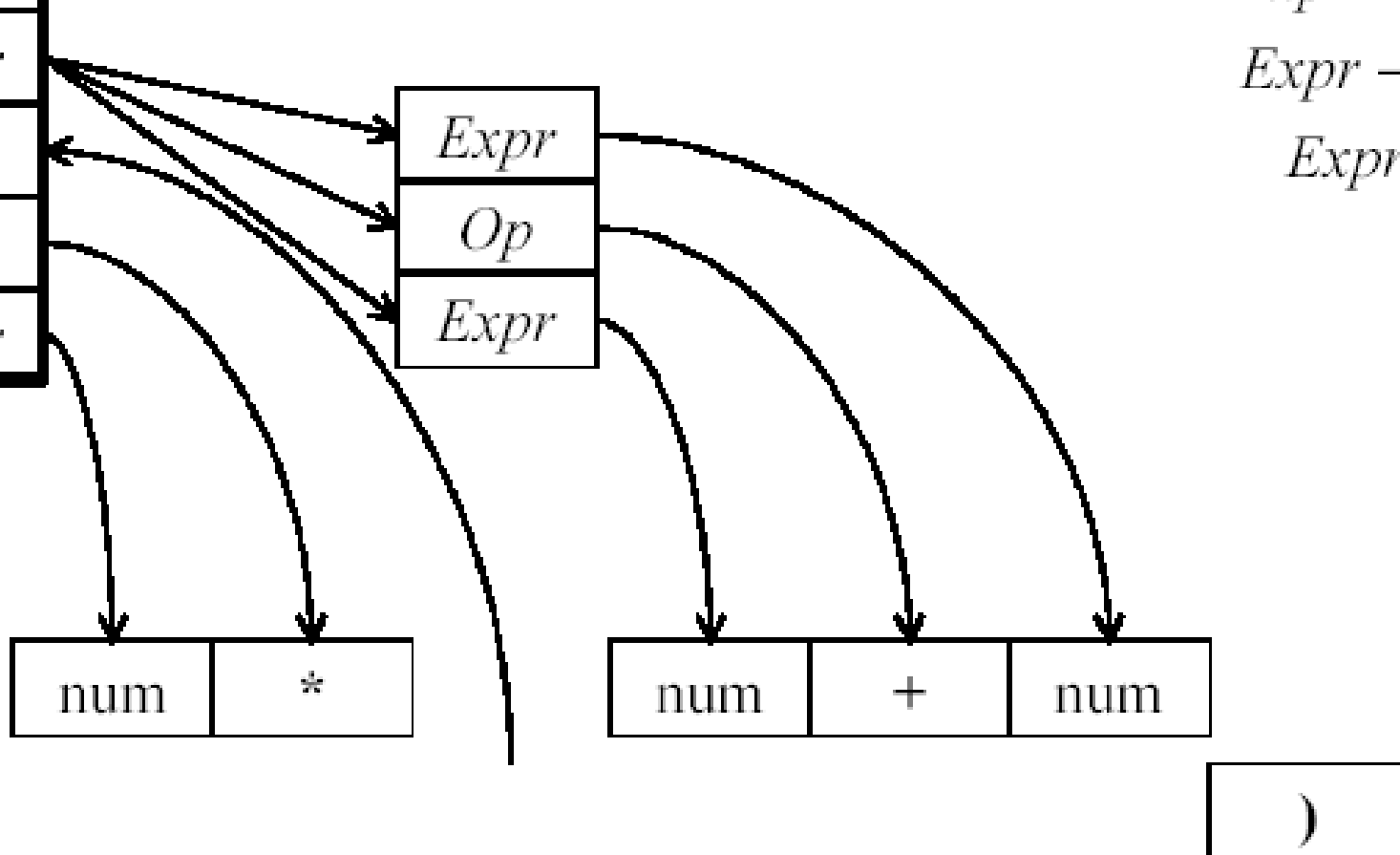
$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

REDUCE



Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - Expr$

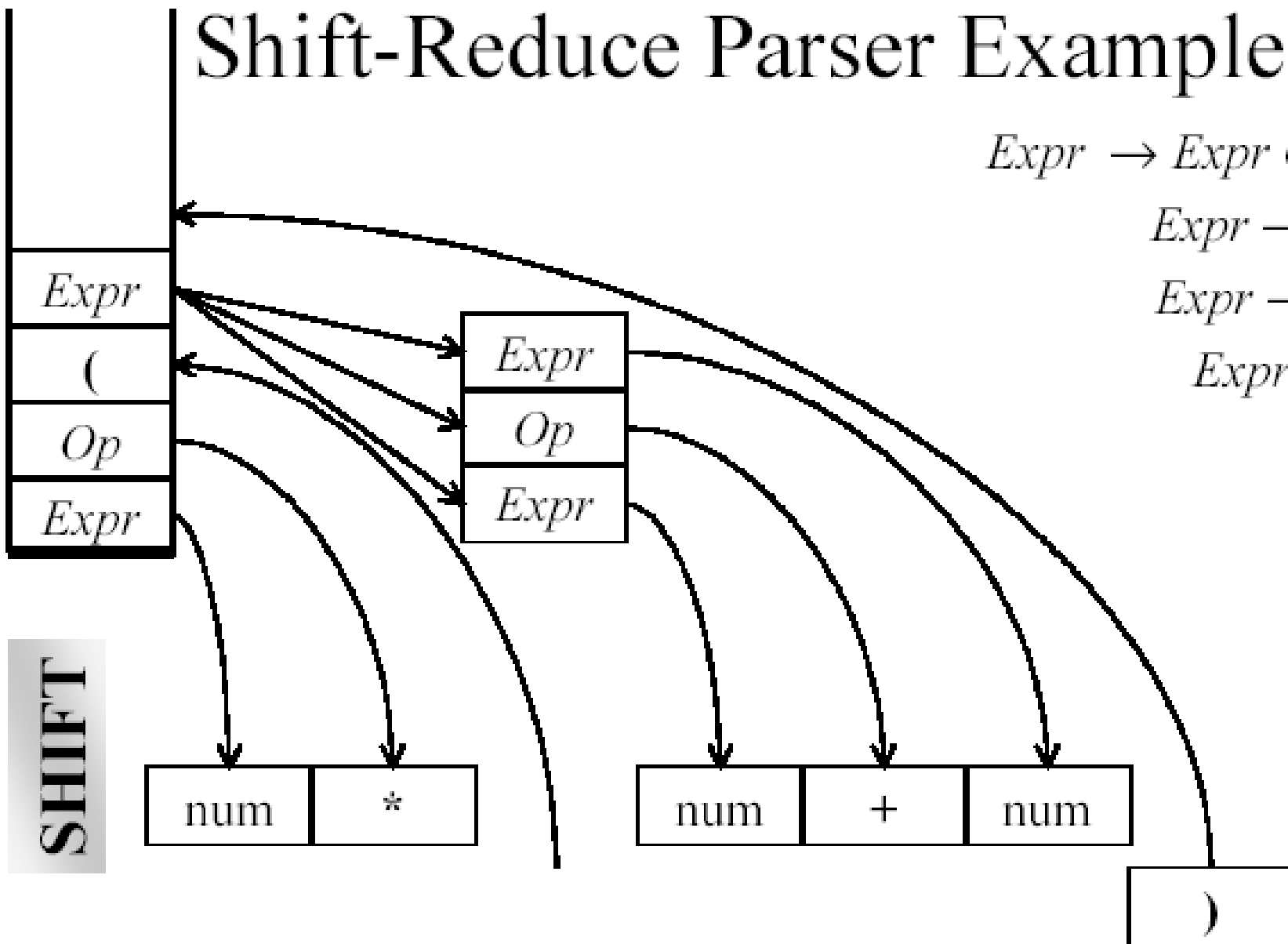
$Expr \rightarrow num$

$Op \rightarrow +$

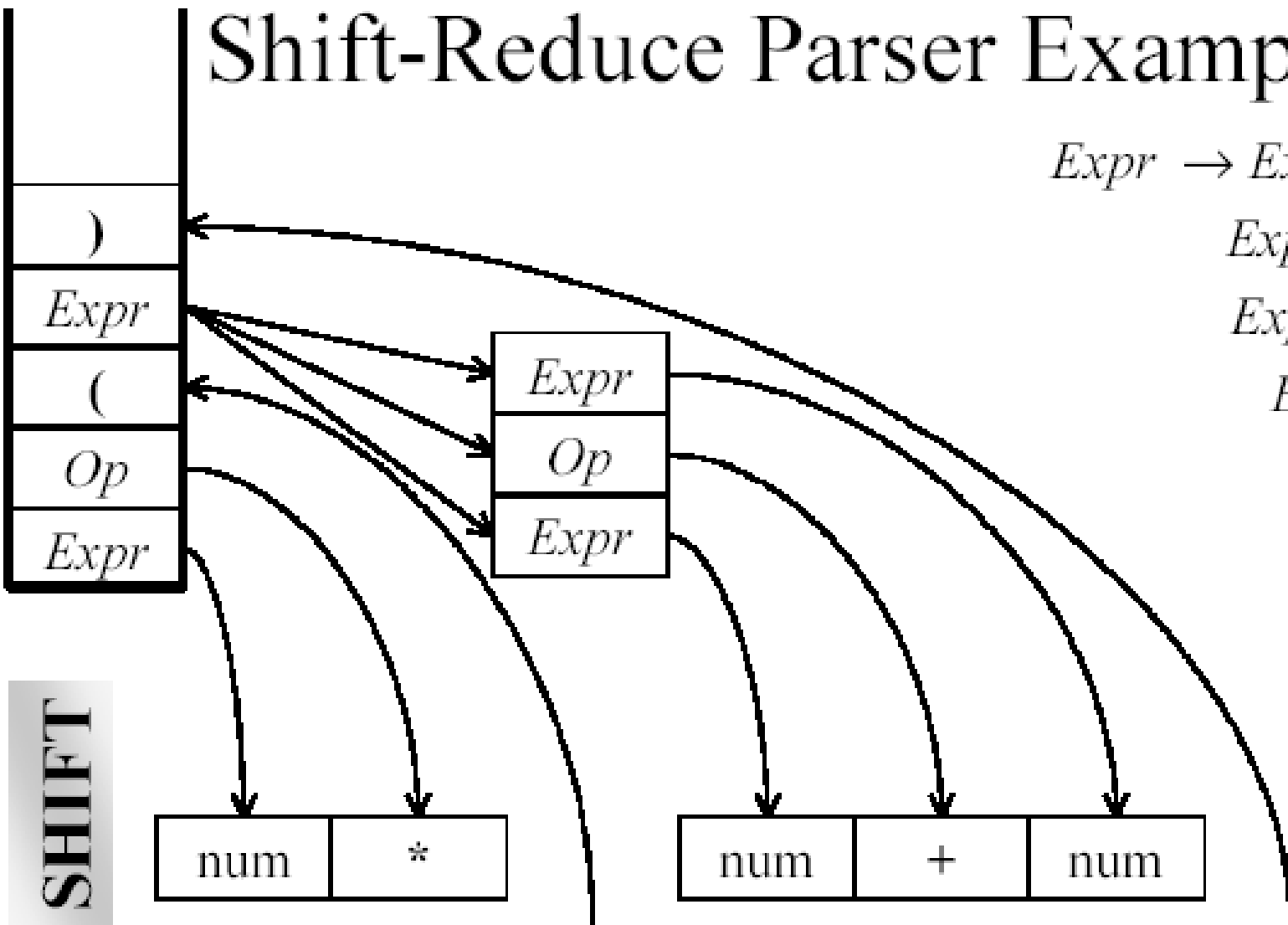
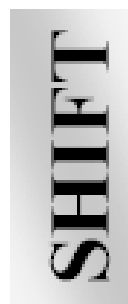
$Op \rightarrow -$

$Op \rightarrow *$

SHIFT



Shift-Reduce Parser Example

$$Expr \rightarrow Expr \ Op \ Expr$$
$$Expr \rightarrow (Expr)$$
$$Expr \rightarrow - Expr$$
$$Expr \rightarrow \text{num}$$
$$Op \rightarrow +$$
$$Op \rightarrow -$$
$$Op \rightarrow *$$


Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

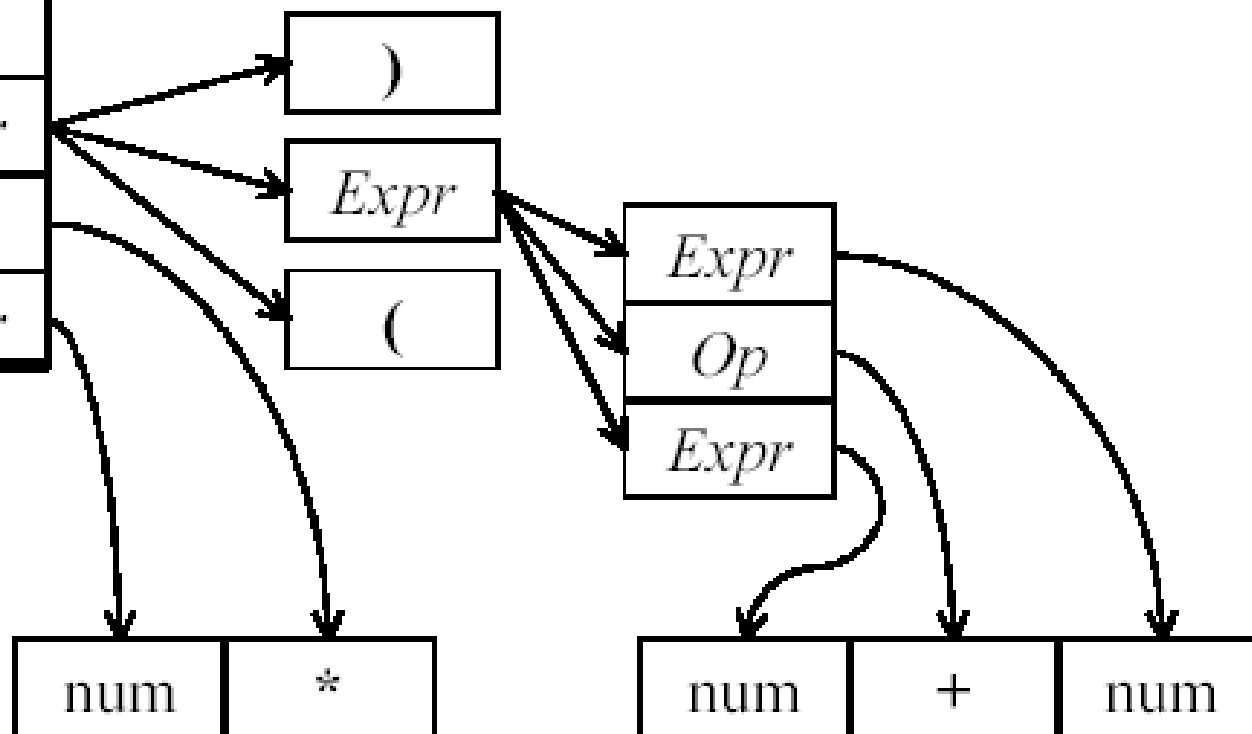
$Expr \rightarrow - Expr$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$



Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - Expr$

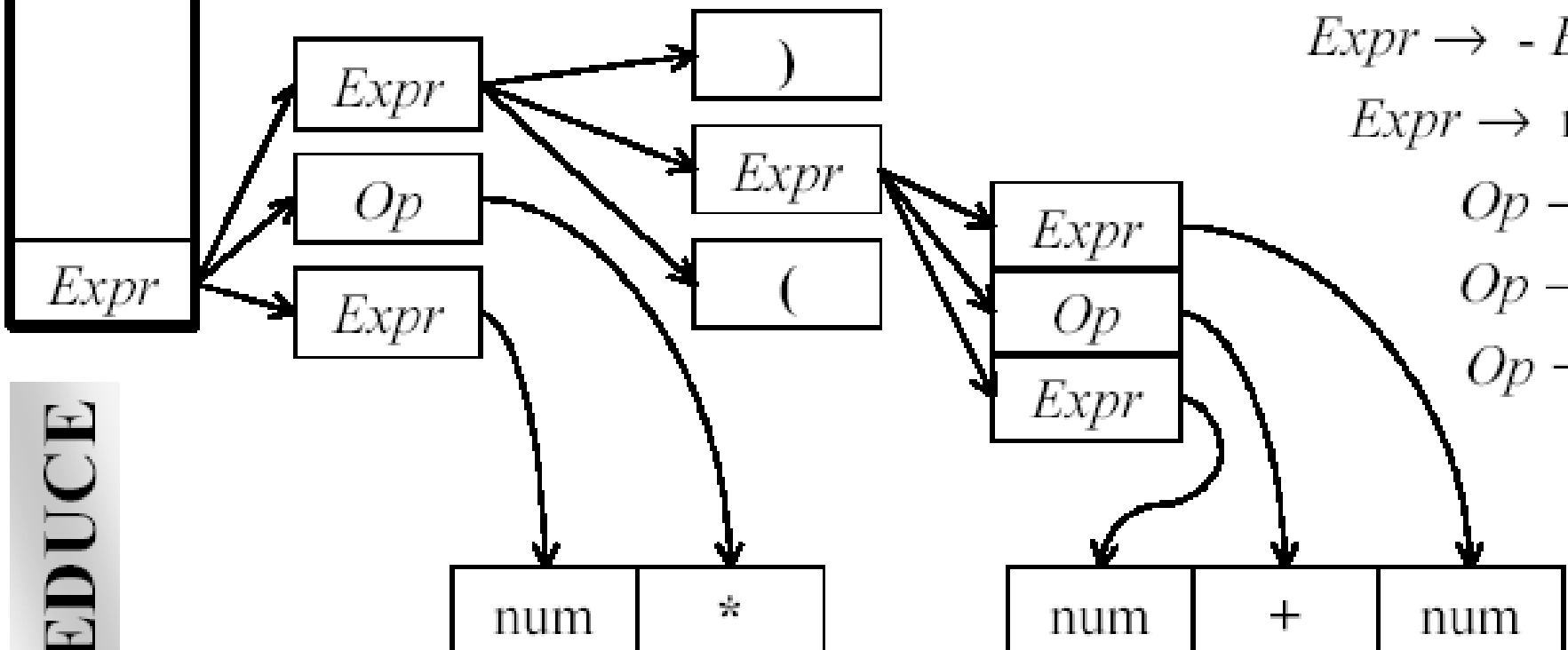
$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

REDUCE



Shift-Reduce Parser Example

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow - Expr$

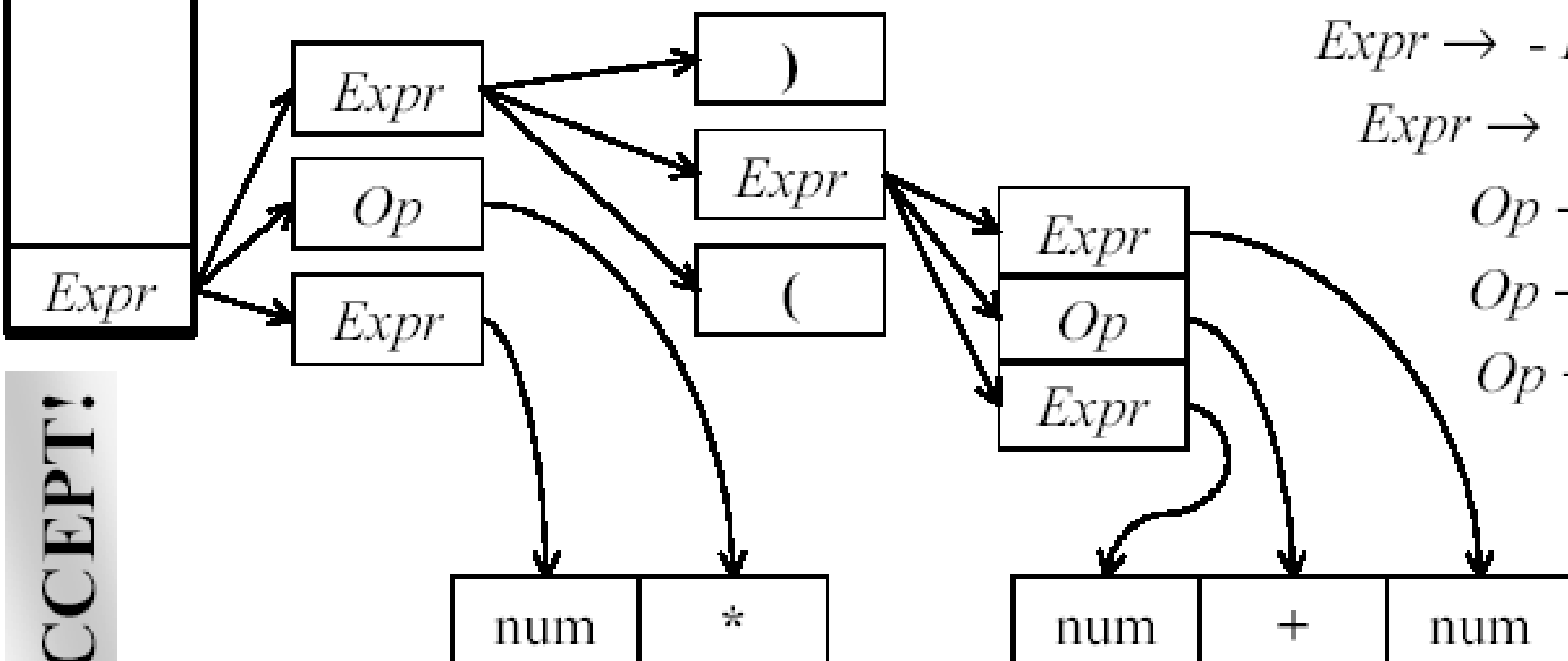
$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

ACCEPT!



观察

石川

- ▶ 一个不断移进-归约的过程，从输入串开始，到初始符号结束或者出错结束。
- ▶ 如果能成功的话，在过程的每一步都得到一个句型。
- ▶ 在过程的每一步，总是做出如下的一个选择：
 - 输入串中的当前输入符号被移进栈，即压入到栈顶，当前输入符号更改为下一个。
 - 如果栈顶部分的符号串刚好跟某个变元A的某个候选式一样，则这部分全部出栈，然后将A压入栈。这个过程即是归约。将来可以给归约附加一些行为或动作如构造语法树、进行语义分析等。
- ▶ 注意到移进-归约分析过程对应于最左归约。

Parsing Stack	Input String	Action
Z_0	num*(num+num)#	shift
Z_0 num	*(num+num)#	shift
Z_0 Expr	*(num+num)#	reduce $Expr \rightarrow \text{num}$
Z_0 Expr*	(num+num)#	shift
Z_0 Expr Op	(num+num)#	reduce $Op \rightarrow *$
Z_0 Expr Op(num+num)#	shift
Z_0 Expr Op(num	+num)#	shift
Z_0 Expr Op(Expr	+num)#	reduce $Expr \rightarrow \text{num}$
Z_0 Expr Op(Expr+	num)#	shift
Z_0 Expr Op(Expr Op	num)#	reduce $Op \rightarrow +$
Z_0 Expr Op(Expr Op num)#	shift
Z_0 Expr Op(Expr Op Expr)#	reduce $Expr \rightarrow \text{num}$
Z_0 Expr Op(Expr)#	red. $Expr \rightarrow Expr Op Expr$
Z_0 Expr Op(Expr)	#	shift
Z_0 Expr Op Expr	#	reduce $Expr \rightarrow (Expr)$
Z_0 Expr	#	red. $Expr \rightarrow Expr Op Expr$

栈的变化

- ▶ 初始时栈顶为 Z_0 ;
- ▶ 移进当前输入符号至栈顶（压入该符号）;
- ▶ 当栈顶出现“可归约串”则弹出并将该串的归约变元压入;
- ▶ 分析成功时栈顶为初始符号，下面为#;
- ▶ 分析不成功时以报错结束。

▶ 可能的问题:

- 怎么知道可归约串出现了？如何确定它的归约变元？
- 如果栈顶有可归约串，是归约还是不归约？
- 如果栈顶没有可归约串，就移进还是断言为失败？
- 栈上若有可归约串，它们一定都是在栈顶吗？

分析过程中的冲突

石川

- ▶ 知道“可归约串”出现、确定出以它为候选式的变元、移进还是失败，会面临多选的情形。
- ▶ 对于确定性的过程而言，多选步骤即就是冲突。
- ▶ 宏观来看，确定性的自下而上分析过程面临两种冲突：
 - 栈顶部分可能与多个候选式匹配；
 - 归约-归约冲突
 - 也可能即使有匹配的候选式也不能选，而继续移进直到随后找到其它候选式匹配。
 - 移进-归约冲突

李永亮

Conflicts

$Expr \rightarrow Expr\ Op\ Expr$

$Expr \rightarrow Expr\ -\ Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr\ -$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$



Conflicts

$Expr \rightarrow Expr\ Op\ Expr$

$Expr \rightarrow Expr\ -\ Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr\ -$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

num

SHIFT

-

num

Conflicts

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr -$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

Expr

REDUCE

num

-

num

Conflicts

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

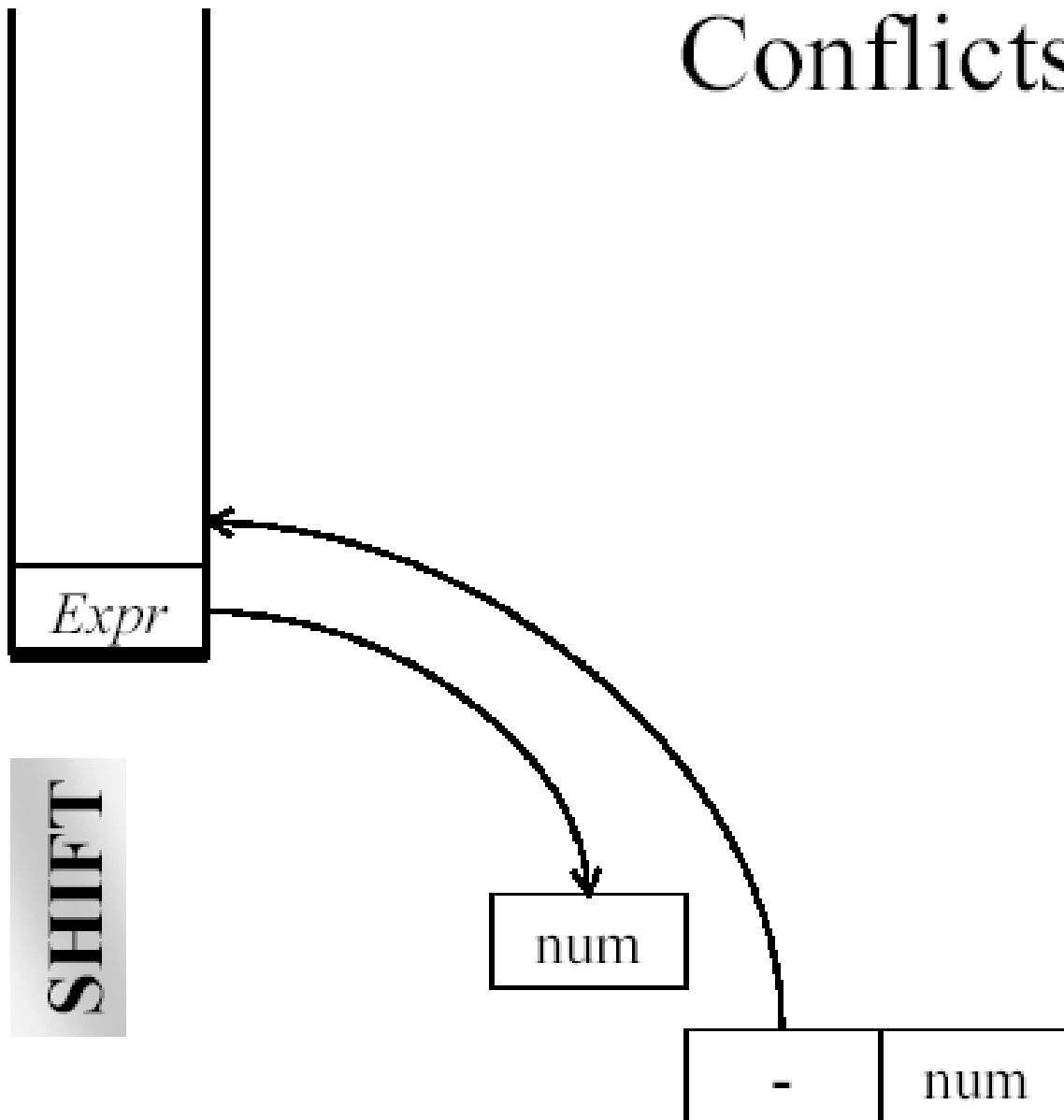
$Expr \rightarrow Expr -$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$



Conflicts

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr -$

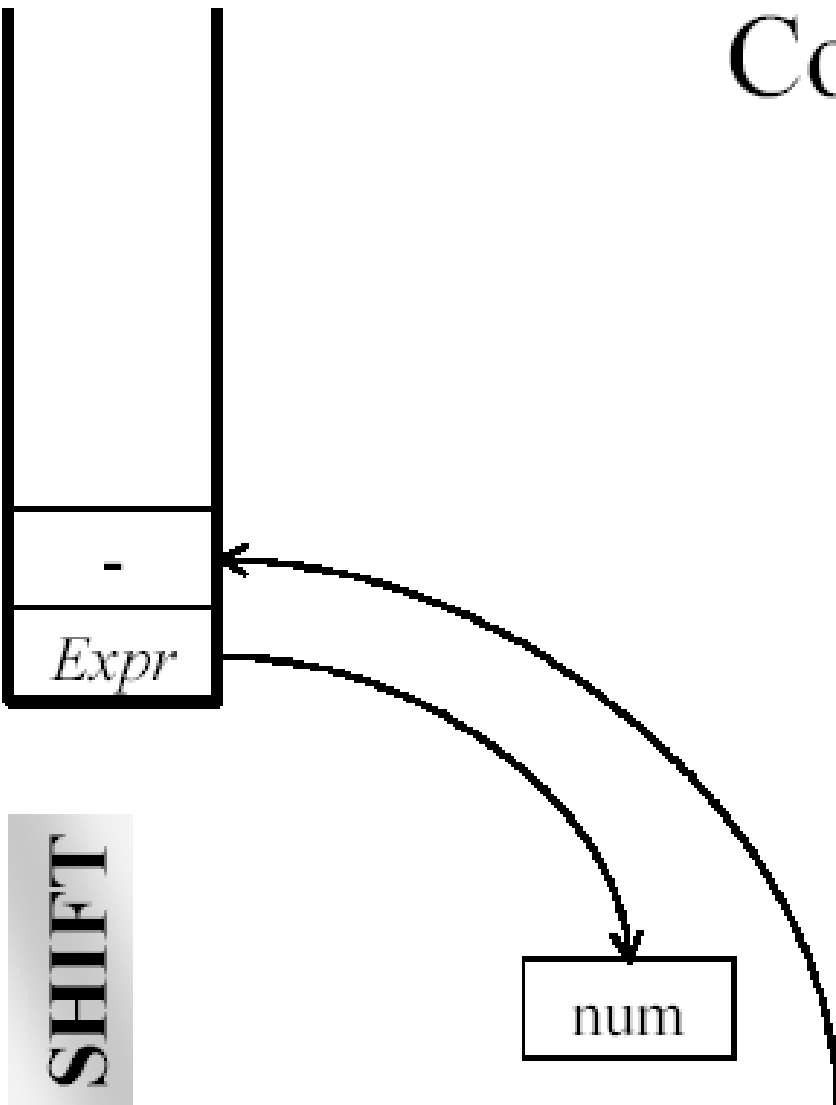
$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

SHIFT



num

Shift/Reduce/Reduce Conflict

$Expr \rightarrow Expr\ Op\ Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr -$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

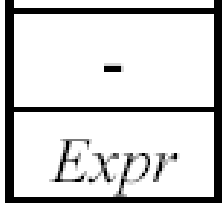
$Op \rightarrow *$

Options:

Reduce

Reduce

Shift



num

num

Shift/Reduce/Reduce Conflict

What Happens
if Choose

Reduce

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr -$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

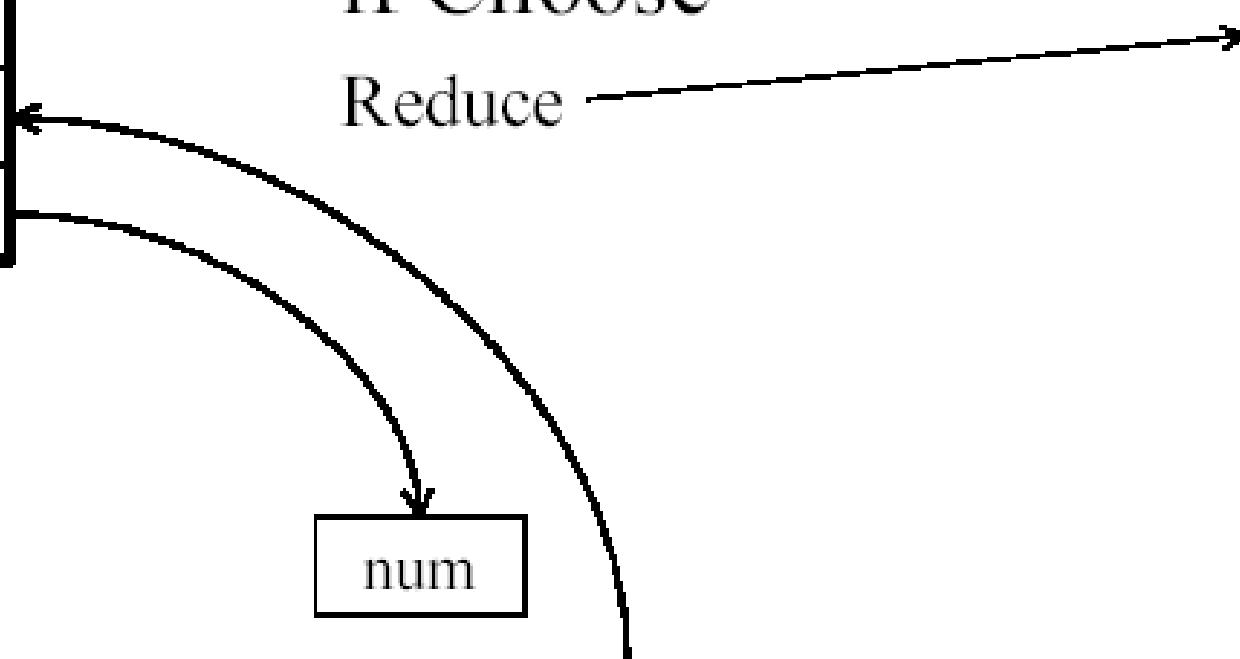
$Op \rightarrow *$

-
Expr

REDUCE

num

num



Shift/Reduce/Reduce Conflict

What Happens
if Choose

Reduce

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr -$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

SHIFT

$Expr$

$Expr$

num

-

num

Shift/Reduce/Reduce Conflict

What Happens
if Choose

Reduce

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr -$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

num

Expr

Expr

num

-

SHIFT

Shift/Reduce/Reduce Conflict

What Happens
if Choose

Reduce

$Expr \rightarrow Expr \text{ Op } Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr -$

$Expr \rightarrow \text{num}$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

$Expr$

$Expr$

$Expr$

num

-

num

REDUCE

Shift/Reduce/Reduce Conflict

What Happens
if Choose

Reduce

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr -$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

$Expr$

$Expr$

$Expr$

num

-

num

FAILS!

Shift/Reduce/Reduce Conflict

Both of These
Actions Work

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

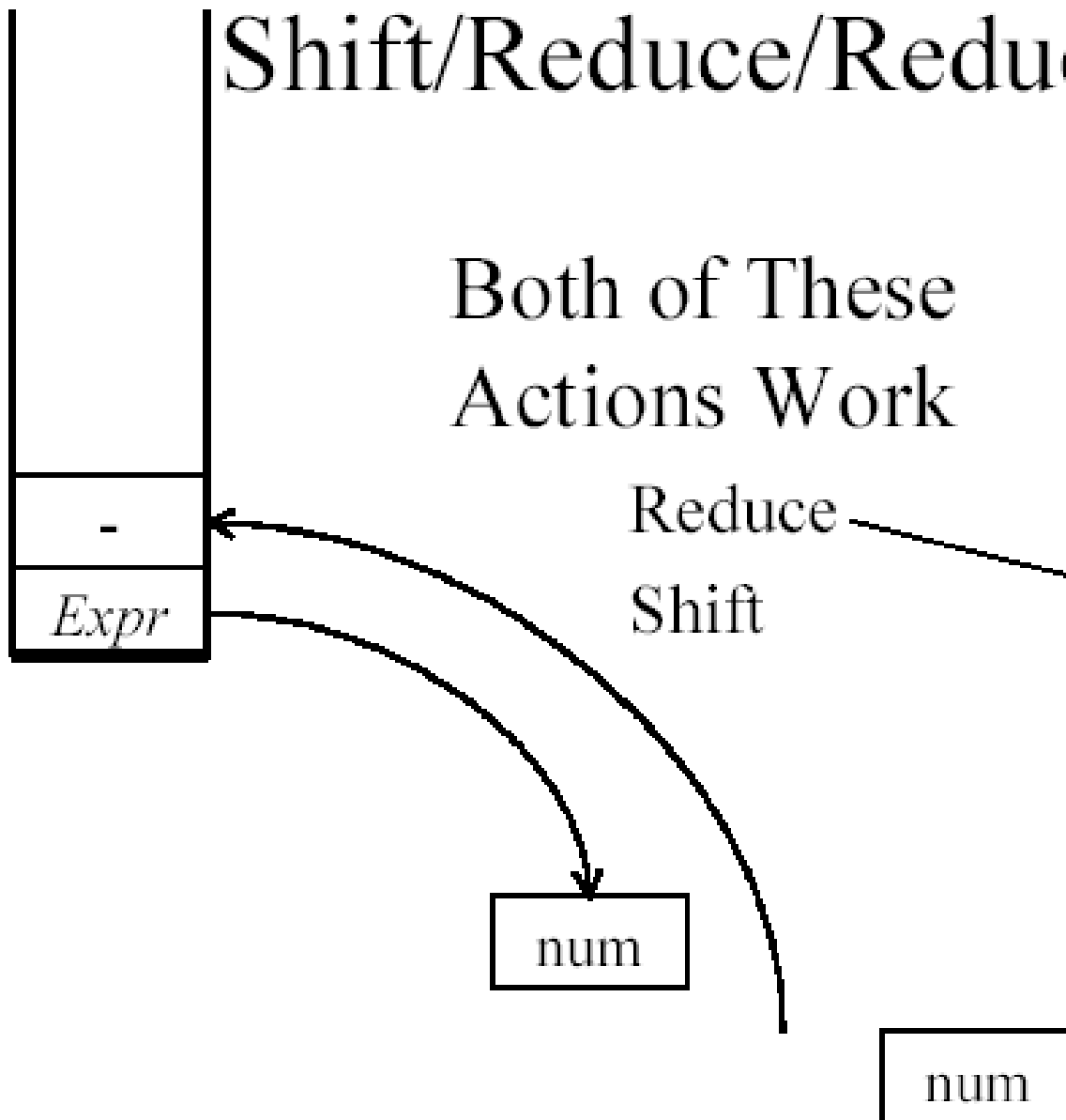
$Expr \rightarrow Expr -$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$



Shift/Reduce/Reduce Conflict

What Happens
if Choose

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr -$

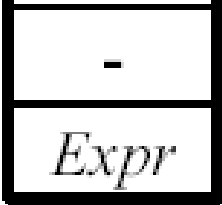
$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

Reduce



num

num

Shift/Reduce/Reduce Conflict

What Happens
if Choose

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr -$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

Reduce

Op
$Expr$

REDUCE

num	-
-----	---

num

Shift/Reduce/Reduce Conflict

What Happens
if Choose

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr -$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

Reduce

num

Op

Expr

num

-

SHIFT

Shift/Reduce/Reduce Conflict

What Happens
if Choose

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr -$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

Expr

Op

Expr

Reduce

REDUCE

num

-

num

Shift/Reduce/Reduce Conflict

What Happens
if Choose

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

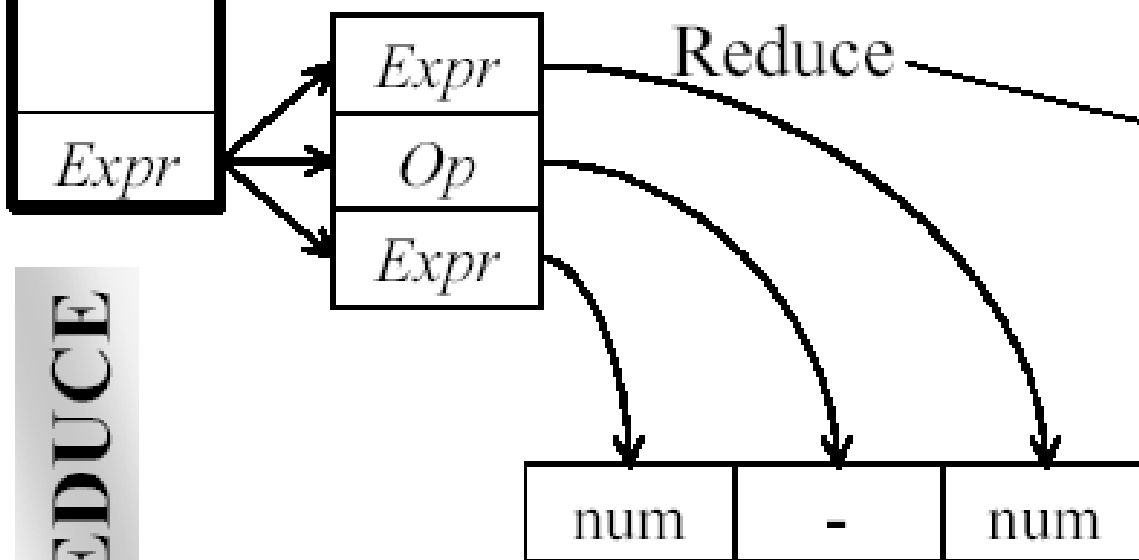
$Expr \rightarrow Expr -$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$



REDUCE

Shift/Reduce/Reduce Conflict

What Happens
if Choose

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr -$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

Expr

Expr

Op

Expr

Reduce

num

-

num

ACCEPT

Conflicts

What Happens
if Choose

Shift

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr -$

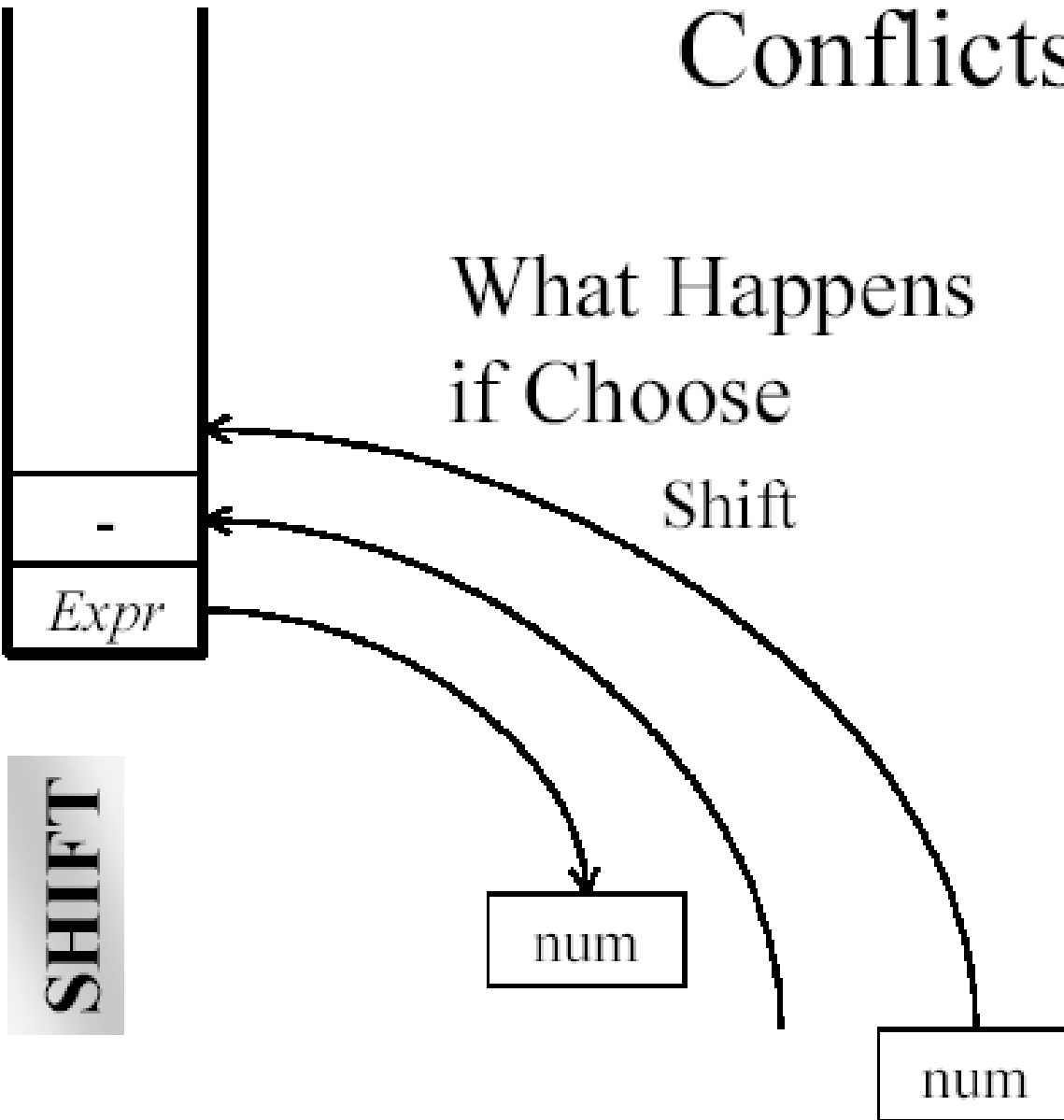
$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

SHIFT



Conflicts

What Happens
if Choose

Shift

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr -$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

num

-

Expr

num

SHIFT

Conflicts

What Happens
if Choose

Shift

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr -$

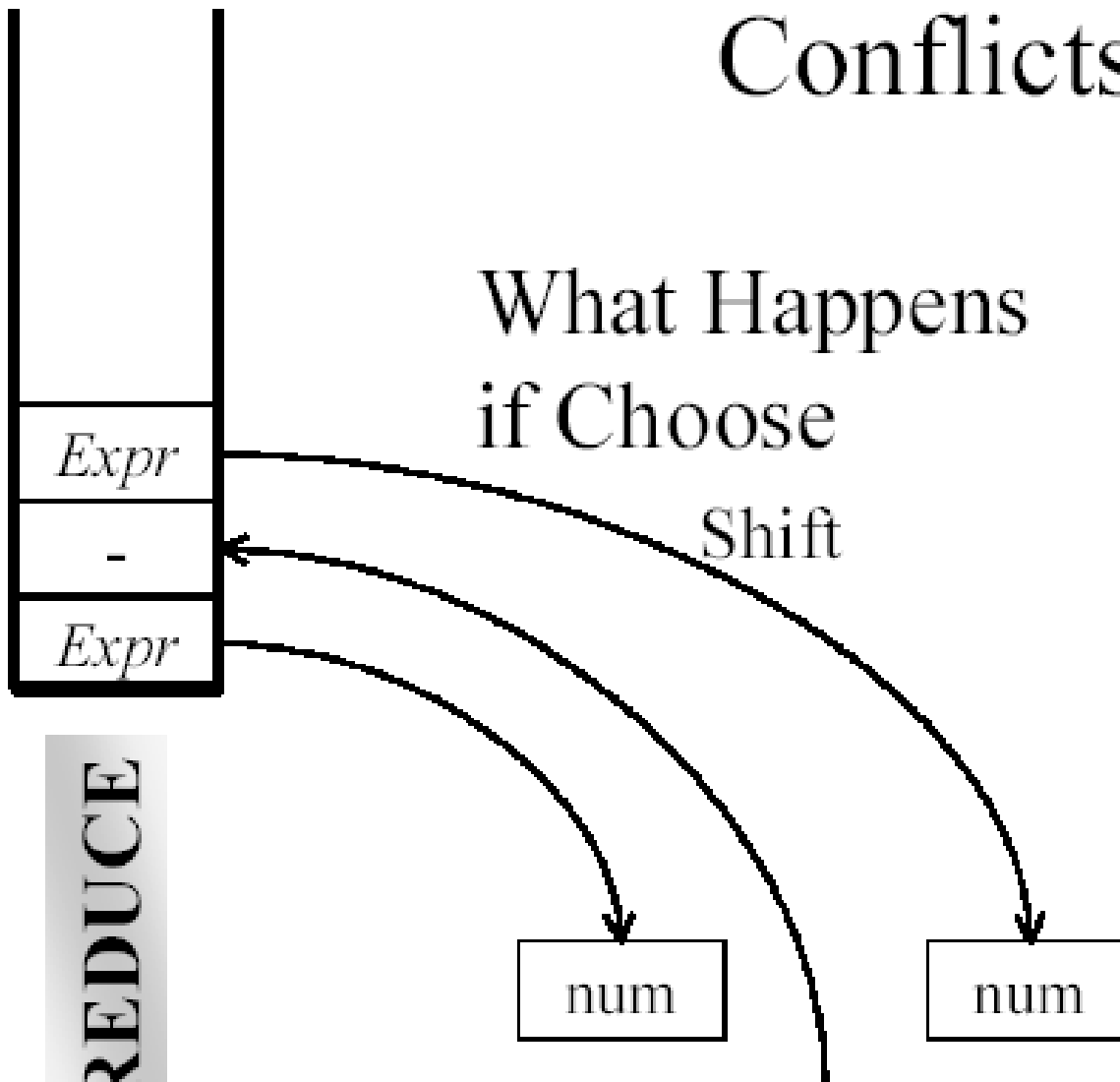
$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

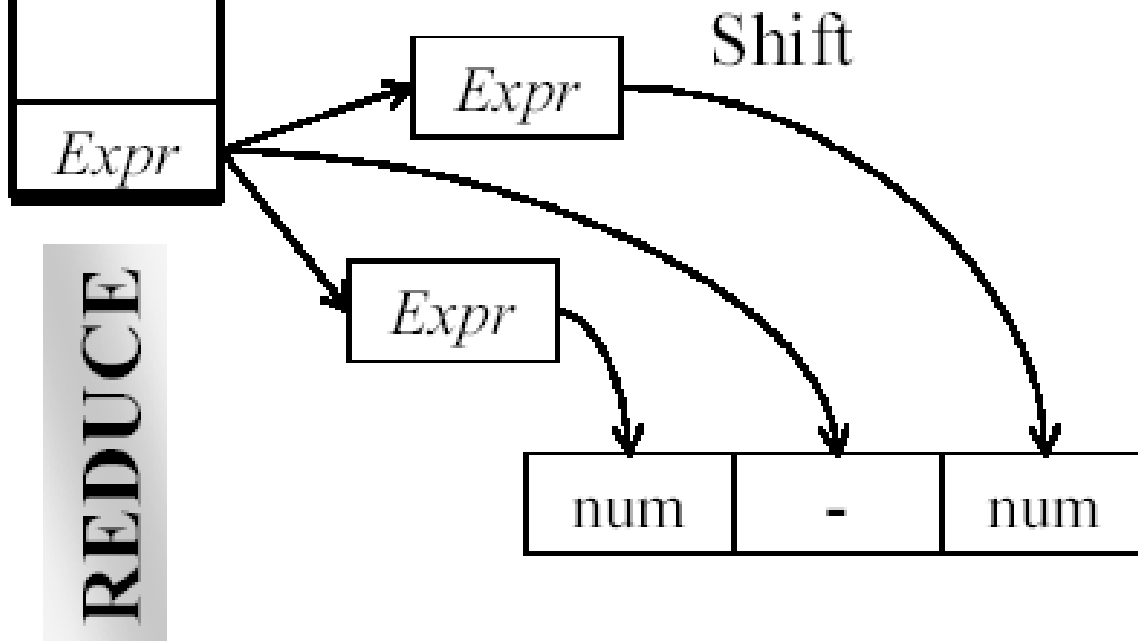
$Op \rightarrow *$

REDUCE



Conflicts

What Happens
if Choose



$Expr \rightarrow Expr \text{ Op } Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr -$

$Expr \rightarrow \text{num}$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

Conflicts

What Happens
if Choose

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

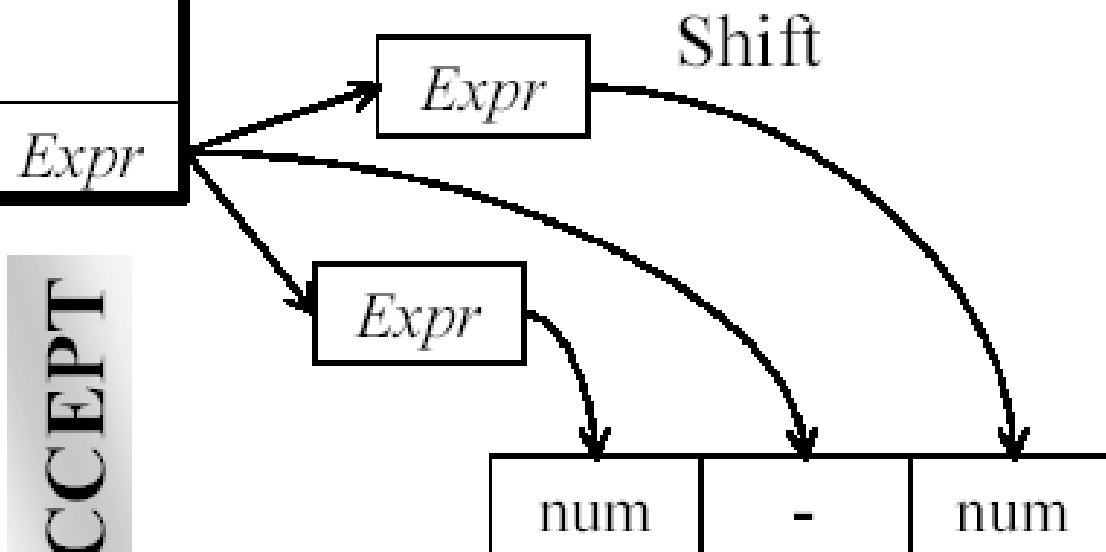
$Expr \rightarrow Expr -$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$



ACCEPT

Shift/Reduce/Reduce Conflict

This Shift/Reduce
Conflict Reflects
Ambiguity in
Grammar

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

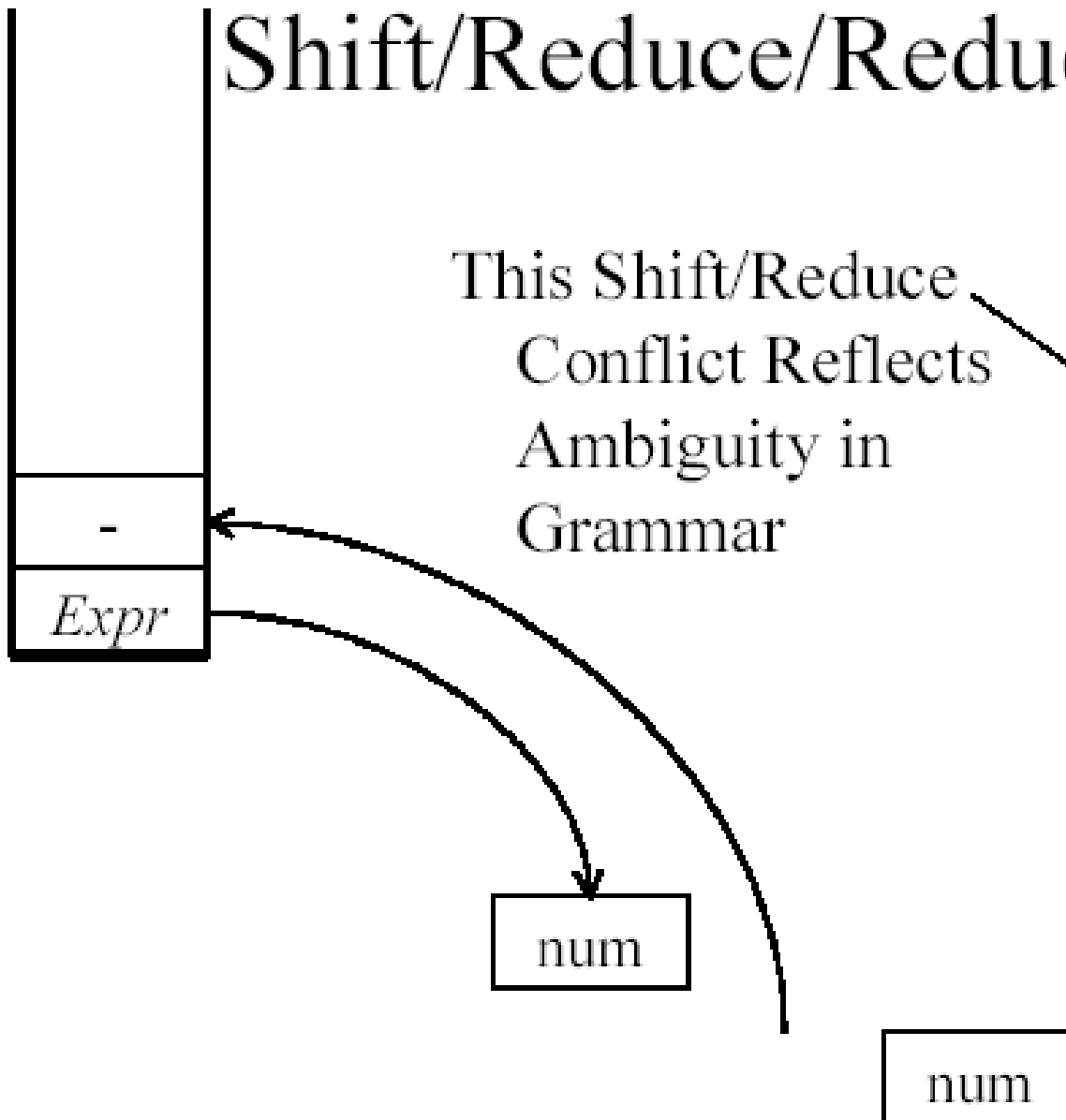
$Expr \rightarrow Expr -$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$



Shift/Reduce/Reduce Conflict

This Shift/Reduce
Conflict Reflects
Ambiguity in
Grammar

$Expr \rightarrow Expr Op Expr$

~~$Expr \Rightarrow Expr - Expr$~~

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr -$

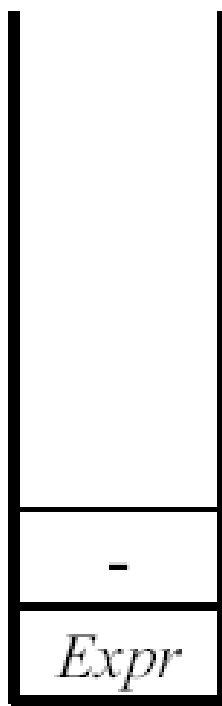
$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

Eliminate by Hacking
Grammar



num

num

Shift/Reduce/Reduce Conflict

This Shift/Reduce
Conflict Can Be
Eliminated By
Lookahead of One
Symbol

$Expr \rightarrow Expr Op Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr -$

$Expr \rightarrow num$

$Op \rightarrow +$

$Op \rightarrow -$

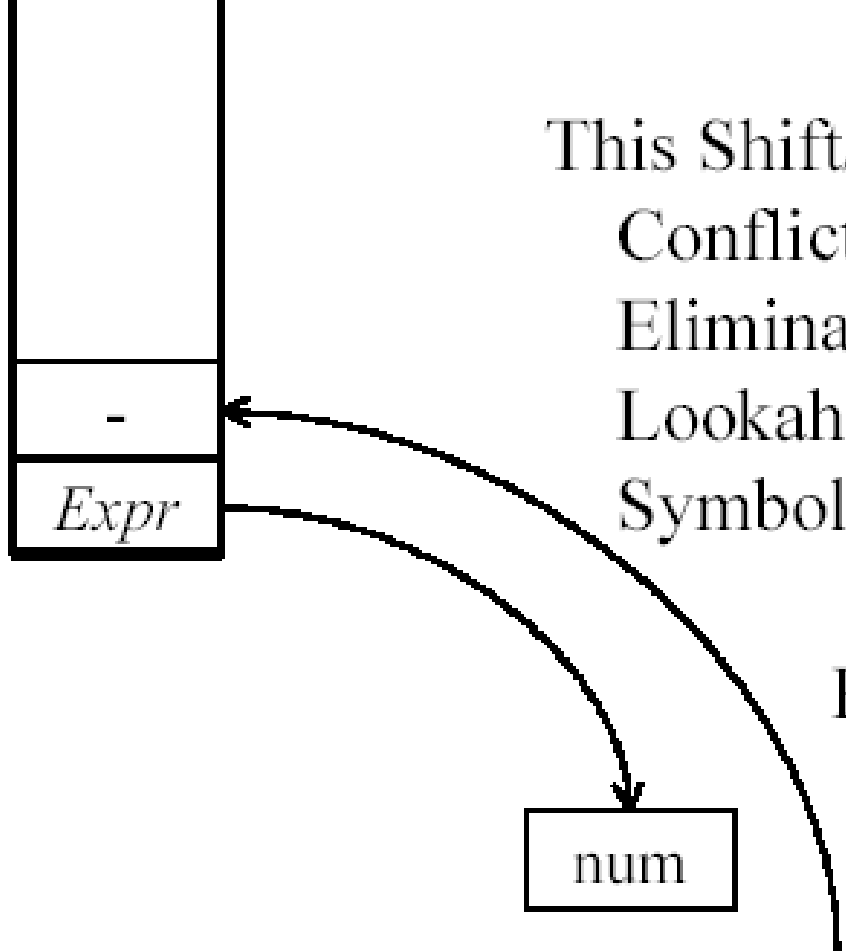
$Op \rightarrow *$

Parser Generator
Should Handle It

-
Expr

num

num



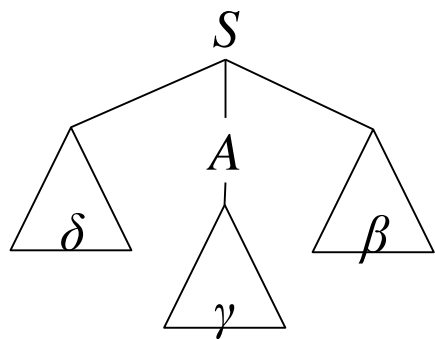
- ▶ “归约-归约”冲突：当前栈顶部分与多个产生式右部匹配：
 - 一个可归约串匹配多个变元的候选式，或者，
 - 多个可归约串匹配多个候选式。
- ▶ “移进-归约”冲突：当可归约串出现时可以归约也可以继续移进。
- ▶ 造成冲突的原因是什么呢？
 - 只能在文法上找原因。
 - 文法有歧义性才造成此类冲突？
 - 与当前输入符号有关？
 - 与当前输入符号及其随后的 $k-1$ 个符号有关？
 - 通过消除文法歧义性、向前查看 k 个符号，几乎避免了冲突的发生？

8.2.3 规范归约

- ▶ 规范推导 = 最右推导、
- ▶ 规范句型 = 右句型，若 $S \Rightarrow_{\text{rm}}^* \beta$ 那么 β 是规范句型
- ▶ 规范归约 = 规范推导的逆，用符号 \Rightarrow 表示。
- ▶ 直接最右推导 $\delta Ax \Rightarrow_{\text{rm}} \delta \gamma x, (A, \gamma) \in \mathcal{P}$ 的逆是 $\delta \gamma x \Rightarrow \delta Ax, (A, \gamma) \in \mathcal{P}$

短语与直接短语

- **短语**: 对于句型 $\delta A \beta$ 若有 $A \Rightarrow^+ \gamma$ 则称 γ 是句型 $\delta \gamma \beta$ 的短语, 是相对于变元 A 的短语。
- **直接短语**: 对于句型 $\delta A \beta$ 若有 $A \Rightarrow \gamma$ 则称 γ 是句型 $\delta \gamma \beta$ 的直接短语, 是相对于 $A \rightarrow \gamma$ 的直接短语。



句型的语法树中每个子树的产物都是一个短语

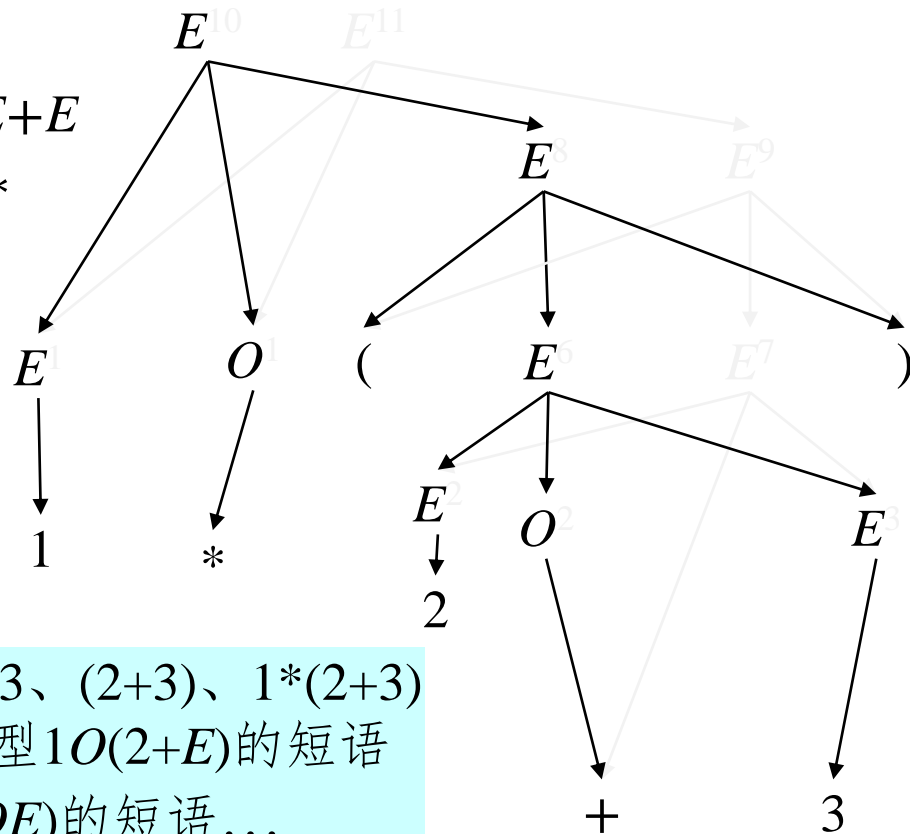
$E \rightarrow EOE \quad E \rightarrow E + E$

$E \rightarrow (E) \quad E \rightarrow E^*$

$E \rightarrow i \quad O \rightarrow +$

$O \rightarrow * \quad U \rightarrow +$

$E \rightarrow Ui$



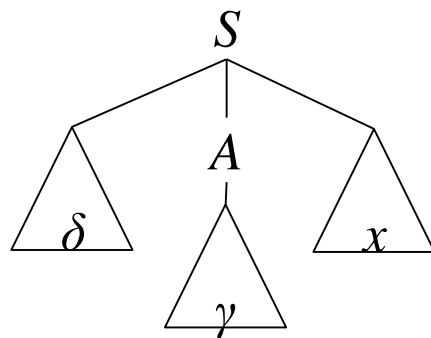
句子 $1*(2+3)$ 的短语: 1、*、2、+、3、 $2+3$ 、 $(2+3)$ 、 $1*(2+3)$
1、2、+、 $2+E$ 、 $(2+E)$ 、 $1O(2+E)$ 都是句型 $1O(2+E)$ 的短语
EOE、 (EOE) 、 $EO(EOE)$ 都是句型 $EO(EOE)$ 的短语...
 带下划线的都是直接短语, 也都是可归约串。

句柄

2016

- ▶ 对于CFG (V, T, P, S) , 若有规范句型 δAx 且其最右推导为 $\delta \gamma x$, 则称 γ 是 $\delta \gamma x$ 的句柄。

最右推导对应于扩展叶子里最右位置的那个变元

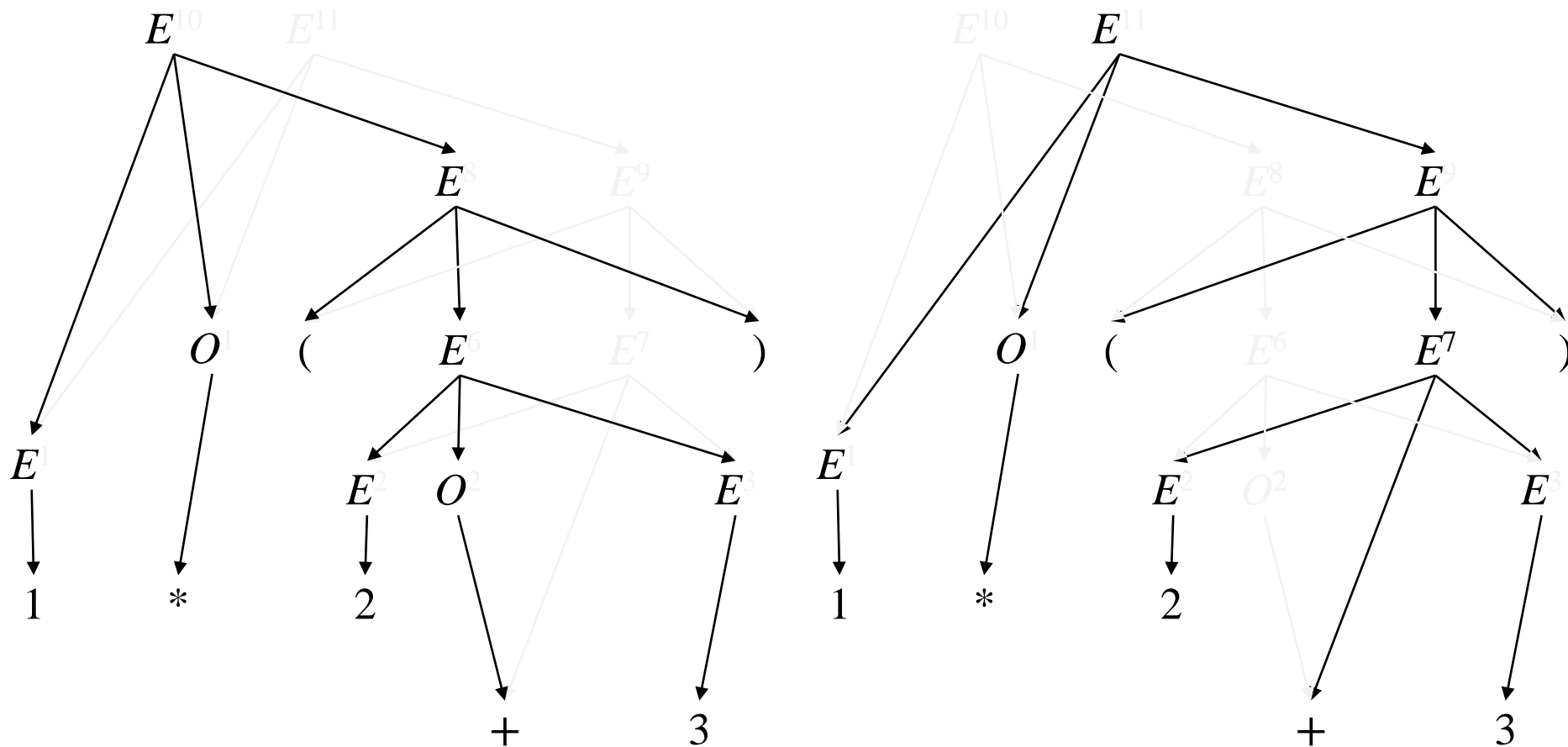


最右推导的逆对应于构建句柄的双亲节点

- ▶ 一个句柄是一个二层子树的产物;
- ▶ 若当前树中有多个二层子树, 句柄就是最左边的那一个。
- ▶ 所以: 一个句型中的最左直接短语就是该句型的句柄。
 - 对无歧义文法而言每个句型都有唯一一个句柄, 这是因为句柄与语法树是一一对应的。回顾有歧义性的文法的某个句子可能有多多个语法树。

2016

一个句型的句柄是其语法树的最左直接短语



- 把一个语法树的最左直接短语剪枝掉（句柄剪枝）会怎样？
- 得到另一个语法树，继续对其进行句柄剪枝，...，直到只剩下初始符号。——这就形成了这种规范归约过程。

规范归约（句柄剪枝）示例

2018

$$P \rightarrow \check{D} \check{S}$$

$$\check{D} \rightarrow \varepsilon \mid D ; \check{D}$$

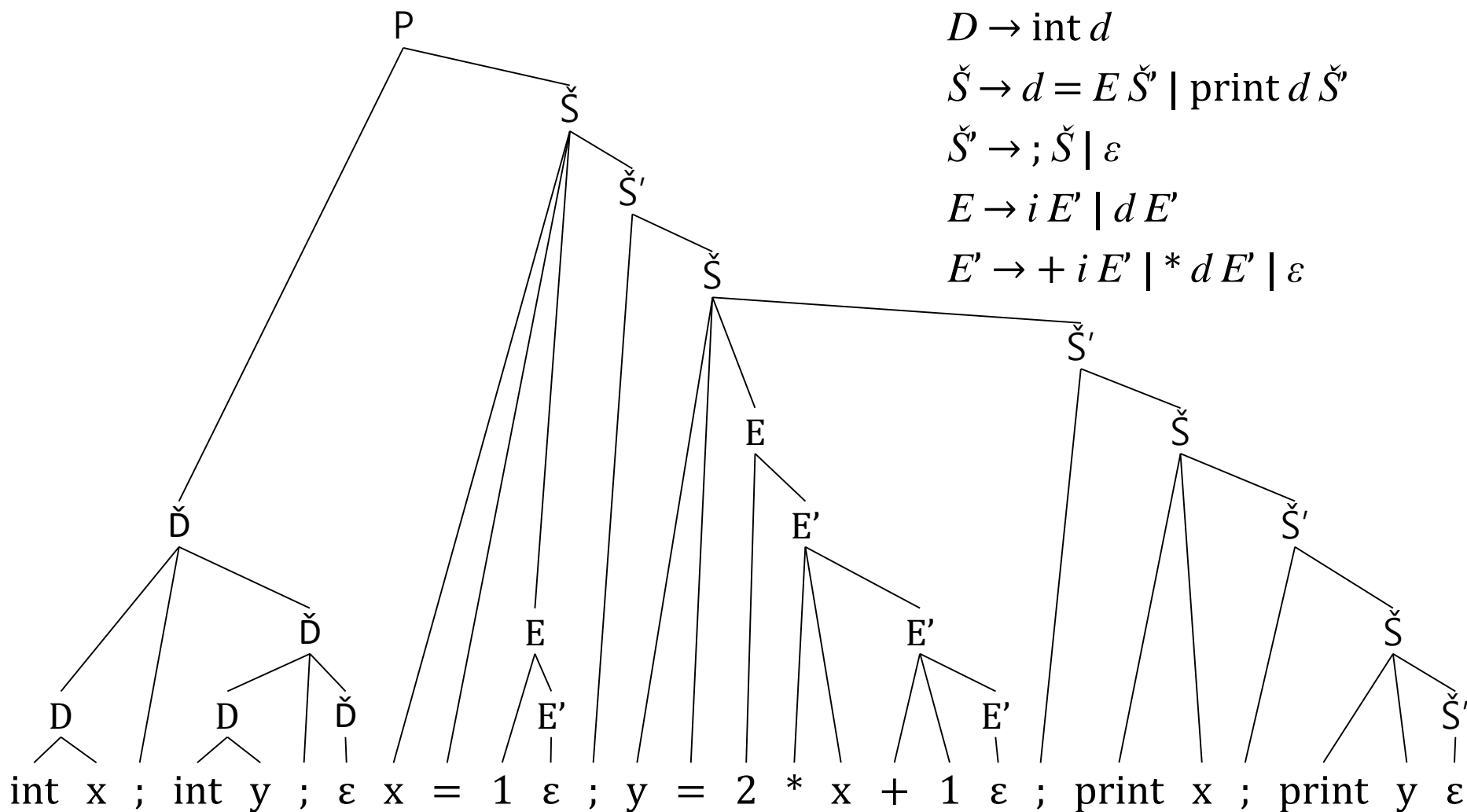
$$D \rightarrow \text{int } d$$

$$\check{S} \rightarrow d = E \check{S}' \mid \text{print } d \check{S}'$$

$$\check{S}' \rightarrow ; \check{S} \mid \varepsilon$$

$$E \rightarrow i E' \mid d E'$$

$$E' \rightarrow + i E' \mid * d E' \mid \varepsilon$$



李永强

规范归约的序列表示

李永亮

- ▶ CFG (V, T, P, S) 的几个术语:
- ▶ 规范归约是最右推导的逆过程
- ▶ 最右推导又被称为规范推导
- ▶ 规范句型: 从句子的规范推导得出的句型, 也叫右句型。
- ▶ 对于 $\alpha_0 = w$, $\alpha_n = S$, 称 $\alpha_0, \dots, \alpha_n$ 是一个规范归约序列, $n \geq 1$, 如果满足下面任意一条:
 - $\alpha_{i+1} \Rightarrow_{rm} \alpha_i$, $i=0, \dots, n-1$
 - α_i 的句柄 γ_i 被替换为 A_{i+1} 得到 α_{i+1} , $(A_{i+1}, \gamma_i) \in P$, $i=0, \dots, n-1$
- ▶ 用规范归约算符 \Rightarrow 表示为 $\alpha_0 \Rightarrow^* \alpha_1 \Rightarrow^* \dots \Rightarrow^* \alpha_n$
- ▶ \Rightarrow 既指归约一步 (句型有变), 也指归约 0 步但同时剩余串的 1 前缀被压入栈顶, 是移进一步 (句型没变)

李永亮

对规范归约过程的观察

王冬冬

$Z_0 w \# \Rightarrow^* \dots, Z_0 \alpha \gamma \cdot y \# \Rightarrow Z_0 \alpha A \cdot y \#, Z_0 \beta \cdot az \# \Rightarrow Z_0 \beta a \cdot z \#, \dots \Rightarrow^* Z_0 S \cdot \#$

其中：句柄 γ 为 A 的候选式； β 的所有后缀都不是句柄。

•int x; int y; x=1; y=2*x+1; print x; print y \Rightarrow^*

int x•; int y; x=1;...

$\check{D}x=E; y=2^*xE'\bullet; \dots$

D;int y•;x=1;...

$\check{D}x=E;y=2E'\bullet; \dots$

D;D;•x=1;...

$\check{D}x=E;y=E\bullet; \text{print } x; \text{print } y$

D;D; $\check{D}\bullet x=1; \dots$

$\check{D}x=E;y=E; \text{print } x; \text{print } y\bullet$

D; $\check{D}\bullet x=1; \dots$

$\check{D}x=E;y=E; \text{print } x; \text{print } y\check{S}'\bullet$

$\check{D}\bullet x=1; \dots$

$\check{D}x=E;y=E; \text{print } x; \check{S}\bullet$

$\check{D}x=1\bullet; y=2^*x+1; \dots$

$\check{D}x=E;y=E; \text{print } x\check{S}'\bullet$

$\check{D}x=1E'\bullet; y=2^*x+1; \dots$

$\check{D}x=E;y=E; \check{S}\bullet$

$\check{D}x=E\bullet; y=2^*x+1; \dots$

$\check{D}x=E;y=E\check{S}'\bullet$

$\check{D}x=E; y=2^*x+1\bullet; \dots$

$\check{D}x=E; \check{S}\bullet$

$\check{D}x=E; y=2^*x+1E'\bullet; \dots$

$\check{D}x=E\check{S}'\bullet \Rightarrow \check{D}\check{S}\bullet \Rightarrow P\bullet$

$P \rightarrow \check{D}\check{S}$

$\check{D} \rightarrow \epsilon$

$\check{D} \rightarrow D; \check{D}$

$D \rightarrow \text{int } d$

$\check{S} \rightarrow d=E\check{S}'$

$\check{S} \rightarrow \text{print } d\check{S}'$

$\check{S}' \rightarrow \epsilon$

$\check{S}' \rightarrow ; \check{S}$

$E \rightarrow iE'$

$E \rightarrow dE'$

$E' \rightarrow + iE'$

$E' \rightarrow^* dE'$

$E' \rightarrow \epsilon$

王冬冬

用“移进-归约”框架模拟规范归约

赵银亮

- ▶ 为了叙述方便将“移进-归约”框架记为**SRF**
- ▶ **活前缀**：规范句型的前缀，它不包括句柄之后的符号。
 - **最大活前缀**：规范句型的一个前缀，它的一个后缀就是句柄。
 - 最大活前缀的任何前缀都是活前缀。
- ▶ 模拟规范归约的**SRF**栈有如下的变化规律：
 - 栈内容永远都是活前缀
 - 当栈内容为最大活前缀时，立即进行归约，即实现一步 \Rightarrow
 - 当栈内容不是最大活前缀时，进行移进操作，实现一步 \Rightarrow

模拟规范归约的SRF流程（归约优先）

2018

- ▶ 先考虑归约优先（最左归约）这种理想情形，然后处理不理想情形（消解冲突），对应于从LR(0)到SLR(1)。
- ▶ 设当前栈内容 ρ ，剩余串 x ，待归约串 ρx 。
 1. 初始化步： $\rho = \varepsilon$ ； $x = w$ ， $ay = x$ ， a 为当前输入符号。
 2. 若 $\tau(\rho) \cap \text{ran}(\mathcal{P}) = \emptyset$ 则将 a 移进栈（push a ），并置 $x = y$ 。
 3. 若 $\rho = \delta\gamma$ 且 $(A, \gamma) \in \mathcal{P}$ 则将句柄 γ 归约为 A ，并置 $\rho = \delta A$ 。
 4. 如果 $\rho = \delta A$ 还是最大活前缀，则如第3步那样继续直到不再有此情形为止。
 5. 重复上述第2~4步直到：
 - $\rho = S$ 且 $x = \varepsilon$ ，则分析成功（接受输入串）。
 - 如果 $\rho = S \wedge x = \varepsilon$ 不能成立，则分析失败（拒绝输入串）。

李永强

示例：移进-归约分析过程

王

步骤	符号栈	剩余串	分析动作
0	Z_0	$(())\#$	shift
1	$Z_0($	$)\#$	shift
2	$Z_0(($	$)\#$	shift
3	$Z_0()$	$)\#$	reduce: $X \rightarrow ()$; 还是shift?
4	$Z_0(X$	$)\#$	shift
5	$Z_0(X)$	$\#$	reduce: $X \rightarrow (X)$; 还是shift?
6	Z_0X	$\#$	acc
7	Z_0S'	$\#$	

$S' \rightarrow X$
 $X \rightarrow (X)$
 $X \rightarrow ()$

文法 $G=(V, T, \mathcal{P}, S)$ 的 **增广文法** $G'=(V \cup \{S'\}, T, \mathcal{P} \cup \{(S', S)\}, S')$

增广文法的初始符号的候选式唯一；

增广文法对移进归约框架保证了归约为原文法的初始符号即成功。

王

确定化的移进-归约分析过程

赵银亮

- ▶ 先考虑不存在冲突的分析过程
 - 不需要查看输入符号就是确定化的**LR(0)**
 - 需要向前查看一个输入符号就是确定化的**SLR(1)**、**LR(1)**
 - 需要向前查看 k 个输入符号才是确定化的**LR(k)**
 - 其它, **LALR(k)**
- ▶ 有冲突时需要进行冲突消解
 - 分析引起冲突的原因, 给出冲突消解规则
 - 运算符结合性和优先级、悬空**ELSE**等冲突都可通过冲突消解规则在**SLR(1)**上得到消解。
- ▶ 本章的方法可以满足程序设计语言。

8.3构建规范归约模拟器

赵永亮

- ▶ 在讨论了栈的变化规律基础上，基于PDA模型，考虑用NFA控制栈的动作。称为可模拟规范归约的模拟器，记为itemPDA，可进一步确定化为itemDPDA。
- ▶ 构建itemDFA：
 - 文法项目与文法项目集
 - 增广文法
 - 活前缀的有效项目
 - 识别活前缀的NFA，即itemNFA，得到由其与SRF组成的itemPDA
 - 得到LR(0)规范簇（识别活前缀的DFA，即itemDFA）
 - 得到由itemDFA和SRF组成的itemDPDA

为了叙述方便，将移进归约框架简写为SRF

赵永亮

- ▶ **定义8.7** 文法项目集和文法项目。文法 G 的**项目集**
 $\text{itemset}(G) = \{(A, \alpha, \beta) \mid (A, \alpha\beta) \in \mathcal{P}\}$, 其中 (A, α, β) 是一个文法项目, 直观地表示为 $A \rightarrow \alpha \cdot \beta$, 是由产生式规则 $A \rightarrow \alpha\beta$ 产生的项目。可以看出, 由产生式 $A \rightarrow \gamma$ 为文法项目集贡献 $|\gamma|+1$ 个项目
- ▶ 例CFG G : $S' \rightarrow X$ $X \rightarrow (X)$ $X \rightarrow ()$
 $\text{itemset}(G)$: $S' \rightarrow \cdot X$ $S' \rightarrow X \cdot$ $X \rightarrow \cdot (X)$ $X \rightarrow (\cdot X)$ $X \rightarrow (X \cdot)$ $X \rightarrow (X) \cdot$
 $X \rightarrow \cdot ()$ $X \rightarrow (\cdot)$ $X \rightarrow () \cdot$
- ▶ 模拟器建模思路: 将一个句子的分析过程 (SRF的步骤序列) 中的每个步骤采用一个一个状态来描述。
- ▶ 每一个文法项目都用于构成一个状态。
 - SRF初始化步所对应的状态是 $[S' \rightarrow \cdot X]$
 - 中间某步骤可能对应状态 $[X \rightarrow (\cdot X)]$ 或其他
 - SRF成功步对应状态 $[S' \rightarrow X \cdot]$

建模规范归约模拟器

王冬冬

- ▶ 将SRF+CFG看作一个系统，作为规范归约模拟器，可进一步写程序实现为Parser（语法分析器）。
- ▶ 基于PDA对SRF+CFG系统进行建模
 - SRF栈的移进动作表示为状态间的转移标记 $a, \varepsilon/a$
 - SRF栈的归约动作表示为状态间的转移标记 $\varepsilon, \gamma/A, (A, \gamma) \in \mathcal{P}$
- ▶ 对状态建模的思路：既兼顾分析过程的历史又展望其未来：
 - 初始状态 $[S' \rightarrow \cdot S]$ 意味着历史为 ε ，未来是“消耗掉 w 并归约出 S ”，所对应的句型为 $\varepsilon \cdot w$ 。
 - 接受状态 $[S' \rightarrow S \cdot]$ 意味着历史为“已经消耗掉了 w 并归约出 S ”而未来是 ε （没有更多的事情），所对应的句型为 $S \cdot \varepsilon$ 。
 - 一般状态 $[A \rightarrow \gamma^{\text{past}} \cdot \gamma^{\text{future}}]$ 意味着历史为 $\gamma^{\text{past}} \in \tau(\rho^{\text{current}})$ ，未来为 $\gamma \in \tau(\rho^{\text{future}})$ ，所对应的句型为 $\rho^{\text{current}} \cdot a\gamma$ ，其中 $\gamma = \gamma^{\text{past}} \cdot \gamma^{\text{future}}$ ， $\rho^{\text{current}} \gamma^{\text{future}} = \rho^{\text{future}}$ 。若 $\gamma^{\text{past}} = \varepsilon$ 意味着未来是“归约出 A ”。
 $\gamma^{\text{future}} = \varepsilon$ 意味着已经归约出 A 了。 ρ 指SRF的栈内容。

王冬冬

建模规范归约模拟器（续）

2016

- 将移动 \vdash 分解为“状态转移”和“SRF栈动作”两部分同步进行，前者在itemNFA上进行，后者在SRF上进行。
- 状态 $[A \rightarrow \alpha \cdot c \beta]$ 转移到 $[A \rightarrow \alpha c \cdot \beta]$ 与把 c 移进栈二者同步进行
 - 从 $[A \rightarrow \alpha \cdot N \beta]$ 转移到 $[A \rightarrow \alpha N \cdot \beta]$ ，前提是消耗剩余串归约出 N 。将此设为归约目标，并分解为（许愿）能从当前状态开始并移动到栈顶为 γ 的ID，其中 $(N, \gamma) \in \mathcal{P}$ 。
 - “从状态为 $[A \rightarrow \alpha \cdot N \beta]$ 的ID移动到栈顶为 γ 的ID”必然要开启归约为 N 的过程，即从状态为 $[N \rightarrow \cdot \gamma]$ 的ID移动到状态为 $[N \rightarrow \gamma \cdot]$ 的ID（这是许下的愿望）。
 - 在 $[N \rightarrow \gamma \cdot]$ 状态时愿望达成，然后（还愿）先将当前状态回溯至许愿处，然后将栈顶 γ 替换为 N ，这就实现了归约为 N 的目标，注意伴随状态转移到 $[A \rightarrow \alpha N \cdot \beta]$ 。

示例：构建规范归约模拟器

李永亮

$\varepsilon \cdot ()$

状态 $[S' \rightarrow \cdot X]$ ，目标①归约出 X ，许愿①能移动到栈顶为 (X) 的ID或②能移动到栈顶为 $()$ 的ID。

$\Rightarrow (\cdot)$

状态 $[X \rightarrow (\cdot X)]$ ，目标②归约出 X ，许愿能移动到栈顶为③ (X) 或④ $()$ 的ID，或者；
状态为 $[X \rightarrow (\cdot)]$ ，积跬步 \Rightarrow ， $\rho^{\text{current}} = ($ ， $\rho^{\text{future}} = ()$

$\Rightarrow ((\cdot))$

状态 $[X \rightarrow (\cdot)]$ ，积跬步 \Rightarrow ， $\rho^{\text{current}} = (($ ， $\rho^{\text{future}} = (()$

$\Rightarrow () \cdot$

状态 $[X \rightarrow () \cdot]$ ，愿望④已达成，还愿④。

$\Rightarrow (X \cdot)$

状态 $[X \rightarrow (X \cdot)]$ ，积跬步 \Rightarrow^* ，达成目标②。 $\rho^{\text{current}} = (X$ ， $\rho^{\text{future}} = (X)$ 。注：正走在实现愿望①的路上

$\Rightarrow (X) \cdot \varepsilon$

状态 $[X \rightarrow (X) \cdot]$ ，愿望①已达成，还愿①。

$\Rightarrow X \cdot \varepsilon$

状态 $[S' \rightarrow X \cdot]$ ，积跬步 \Rightarrow^* ，达成目标①， $\rho^{\text{current}} = X$ ， $\rho^{\text{future}} = X$

李永亮

示例：得出itemNFA

李永亮

$\varepsilon \cdot (())$

$[S' \rightarrow \cdot X];$ ① X ; ① (X) ; ② $()$

$\Rightarrow (\cdot ())$

$[X \rightarrow (\cdot X)];$ ② X ; ③ (X) ; ④ $()$ 。
 $[X \rightarrow (\cdot)]$, 积跬步 $\rho^{\text{current}} = ($, $\rho^{\text{future}} = ()$

$\Rightarrow ((\cdot))$

$[X \rightarrow (\cdot)]$, 积跬步 $\rho^{\text{current}} = (($, $\rho^{\text{future}} = (()$

$\Rightarrow (()) \cdot$

$[X \rightarrow () \cdot]$, 愿望④已达成, 还愿④。

$\Rightarrow (X \cdot)$

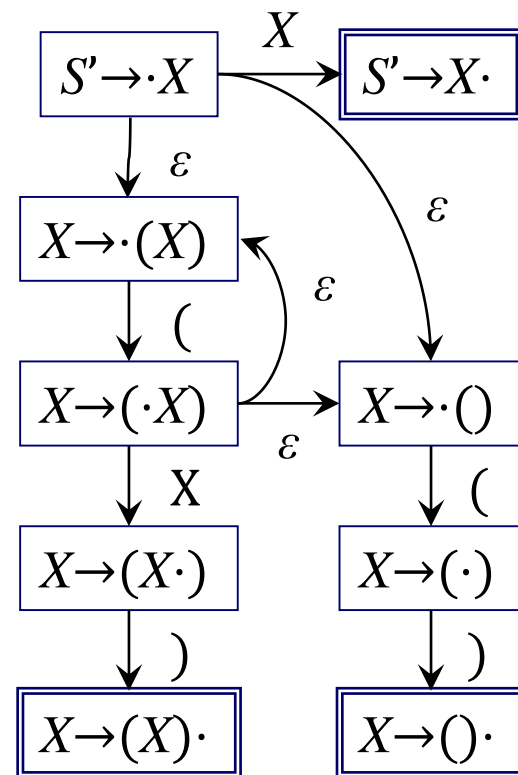
$[X \rightarrow (X \cdot)]$, 积跬步达成目标②。
 $\rho^{\text{current}} = (X$, $\rho^{\text{future}} = (X)$

$\Rightarrow (X) \cdot \varepsilon$

$[X \rightarrow (X) \cdot];$ 愿望①已达成, 还愿①。

$\Rightarrow X \cdot \varepsilon$

$[S' \rightarrow X \cdot]$, 积跬步达成总目标①,
 $\rho^{\text{current}} = X$, $\rho^{\text{future}} = X$



李永亮

示例：得出itmeDFA

2018

$\varepsilon \cdot ()$

状态0; ① X; ① (X); ② ()

$\Rightarrow (\cdot())$

2含 $X \rightarrow (\cdot X)$; ② X; ③ (X); ④ ()。
2含 $X \rightarrow (\cdot)$, $\rho^{\text{current}} = ($, $\rho^{\text{future}} = ()$

$\Rightarrow ((\cdot))$

2含 $X \rightarrow (\cdot X)$; X; ⑤ (X); ⑥ ()。
2含 $X \rightarrow (\cdot)$, $\rho^{\text{current}} = (($, $\rho^{\text{future}} = ((()$

$\Rightarrow (()) \cdot$

4[$X \rightarrow () \cdot$], 愿望④已达成还愿④。

$\Rightarrow (X \cdot)$

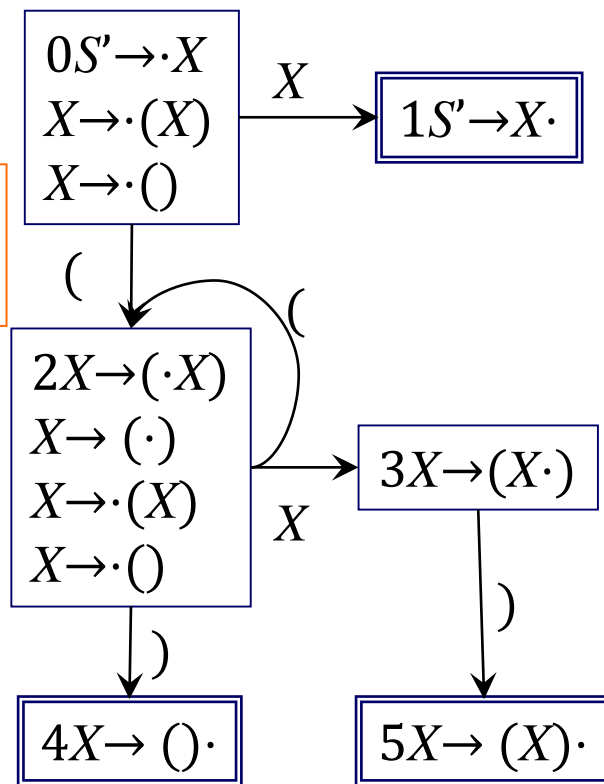
3[$X \rightarrow (X \cdot)$], 积跬步达成目标②。
 $\rho^{\text{current}} = (X$, $\rho^{\text{future}} = (X)$

$\Rightarrow (X) \cdot \varepsilon$

5[$X \rightarrow (X) \cdot$]; 愿望①已达成还愿①。

$\Rightarrow X \cdot \varepsilon$

1[$S' \rightarrow X \cdot$], 积跬步达成总目标①,
 $\rho^{\text{current}} = X$, $\rho^{\text{future}} = X$



4还愿④：从4退到2再到2栈做()/X, 接着 $v(2, X) = 3$

1还愿①：从5退到3再退到2再退到0, 栈做(X)/X, 接着 $v(0, X) = 1$

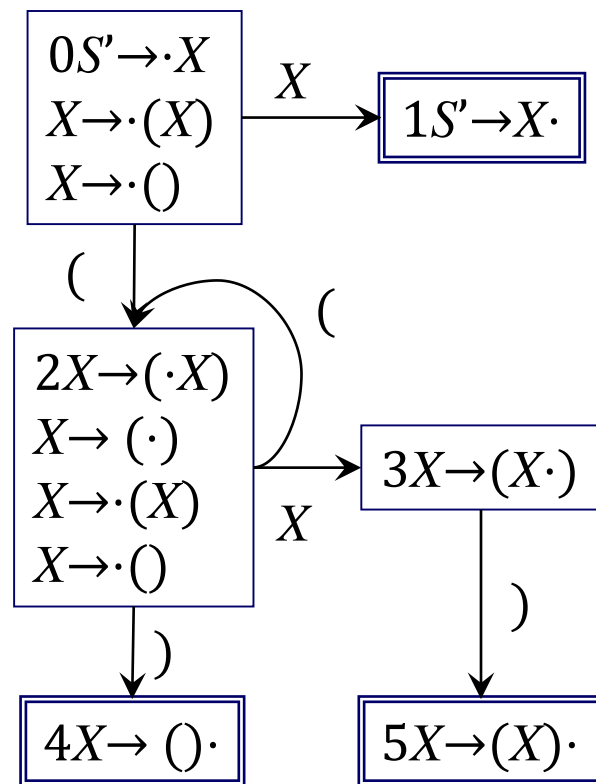
2018

一种确定性的分析过程

李永亮

$[X \rightarrow \gamma \bullet]$ 时，双栈同步弹出 $|\gamma|$ 个符号，符号栈压入 X ，状态找顶 s 替换为 $v(s, X)s$ 。

状态栈	符号栈	剩余串	动作
0	#	(())#	shift
02	#()#	shift
022	#(()#	shift
0224	#())#	reduce $X \rightarrow ()$
023	#(X)#	shift
0235	#(X)	#	reduce $X \rightarrow (X)$
01	#X	#	acc



李永亮

$(A, \gamma) \in \mathcal{P}$

对状态的进一步认识

8.16.2

- ▶ 从待约项目 $N \rightarrow \alpha \cdot A \beta$ 定下目标“消耗剩余串归约出A”，并许下愿望“能移动到栈顶为 γ 这样的ID”。这里，A是点右变元， γ 是其候选式
 - 例：待约项目 $S' \rightarrow \cdot X$ 和 $X \rightarrow (\cdot X)$ 的点右变元都是X
- ▶ 从初始项目（作为状态）开启实现愿望之路（积跬步）
 - 最大的愿望是栈顶为 S' 的唯一候选式
 - 在后面还愿时，该起点会被回溯到并直至许愿点。
 - 初始项目 $S' \rightarrow \cdot X$ 、 $X \rightarrow \cdot (X)$ 和 $X \rightarrow \cdot ()$ 作为状态就都是实现愿望之起点
- ▶ 从完全项目知道愿望已经达成，即一条愿望之路已完成，SRF栈顶为 γ ，并立即还愿，先回退到许愿时所处状态 $[N \rightarrow \alpha \cdot A \beta]$ ，同时将栈顶 γ 替换为A，然后（积跬步）状态转移到 $[N \rightarrow \alpha A \cdot \beta]$
 - 例：完全项目 $S' \rightarrow X \cdot$ 、 $X \rightarrow (X) \cdot$ 和 $X \rightarrow () \cdot$ 分别是各愿望之路的终点
- ▶ 从移进项目看到是走在愿望之路上（积跬步）
 - 移进项目 $X \rightarrow \cdot (X)$ 、 $X \rightarrow (X \cdot)$ 、 $X \rightarrow \cdot ()$ 和 $X \rightarrow (\cdot)$ 都是通过SRF移进而形成一步规范归约 \Rightarrow

李永亮

对文法项目归类

- ▶ G 的项目集 $\text{itemset}(G)$:
- ▶ $S' \rightarrow \bullet X$ 初始项目、待约项目
- ▶ $S' \rightarrow X \bullet$ 接受项目、归约项目、完全项目
- ▶ $X \rightarrow \bullet (X)$ 初始项目、移进项目
- ▶ $X \rightarrow (\bullet X)$ 待约项目
- ▶ $X \rightarrow (X \bullet)$ 移进项目
- ▶ $X \rightarrow (X) \bullet$ 归约项目、完全项目
- ▶ $X \rightarrow \bullet ()$ 初始项目、移进项目
- ▶ $X \rightarrow (\bullet)$ 移进项目
- ▶ $X \rightarrow () \bullet$ 归约项目、完全项目

CFG G : $S \rightarrow X$ $X \rightarrow (X)$ $X \rightarrow ()$

SRF栈内容的变化规律采用itemNFA的状态来表达

- ▶ 设置两类状态，移进状态和归约状态。
- ▶ 在移进一类状态下，SRF栈中不是最大活前缀。是否允许移进当前输入符号呢？问itemNFA（或itemDFA）：
 - 通过当前状态里的项目来指明当前输入符号为哪些终结符时移进，否则出错。
 - 不失一般性，在 $[A \rightarrow \alpha \bullet c \beta]$ 状态，如果 $\text{tok} = c$ 那么移进。
- ▶ 在归约一类状态下，SRF栈中为最大活前缀。用哪个产生式归约呢？问itemNFA（或itemDFA）：
 - 设句柄为 γ ，通过当前状态里的项目来指明用 $A \rightarrow \gamma$ 归约。
 - 不失一般性，在 $[A \rightarrow \gamma \bullet]$ 状态，表示栈顶句柄 γ 归约为 A 。

基于文法穷尽所有状态及状态转移关系

赵永亮

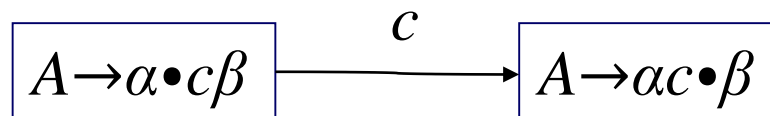
- ▶ 使用itemNFA实现穷尽所有情形。
- ▶ 已知CFG(V, T, \mathcal{P}, S)它的增广文法的项目集为 $\text{itemset}(G')$ 。该集合就是itemNFA的状态集合, $V \cup T$ 是字母表, $[S' \rightarrow \bullet S]$ 是初始状态, $\{[A \rightarrow \gamma \bullet] \mid (A, \gamma) \in \mathcal{P}\}$ 为接受状态集合。
- ▶ 转移函数:

赵永亮

itemNFA的转移函数

李永亮

- ▶ $v([A \rightarrow \alpha \bullet c \beta], c) = \{[A \rightarrow \alpha c \bullet \beta]\}$ 对应于规范归约
 $\dots \alpha \bullet c \dots \Rightarrow^* \dots \alpha c \bullet \dots$



- ▶ $v([A \rightarrow \alpha \bullet N \beta], \varepsilon) \supseteq \bigcup_{(N, \gamma) \in \mathcal{P}} \{[N \rightarrow \bullet \gamma]\}$

- ▶ $v([A \rightarrow \alpha \bullet N \beta], N) \supseteq \{[A \rightarrow \alpha N \bullet \beta]\}$

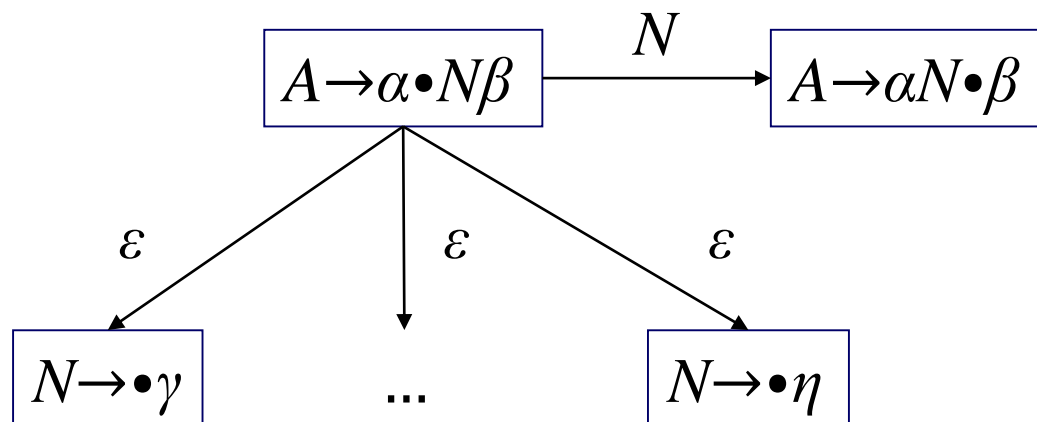
- ▶ 对应于规范归约

$\delta \alpha \bullet y_k z_k \Rightarrow^* \delta \alpha N \bullet z_k$ 若有以下中仅第 k 个计算是合法的

$\delta \alpha \bullet y_1 z_1 \Rightarrow^* \delta \alpha \gamma \bullet z_1$

...

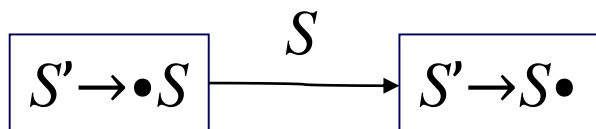
$\delta \alpha \bullet y_m z_m \Rightarrow^* \delta \alpha \eta \bullet z_m$



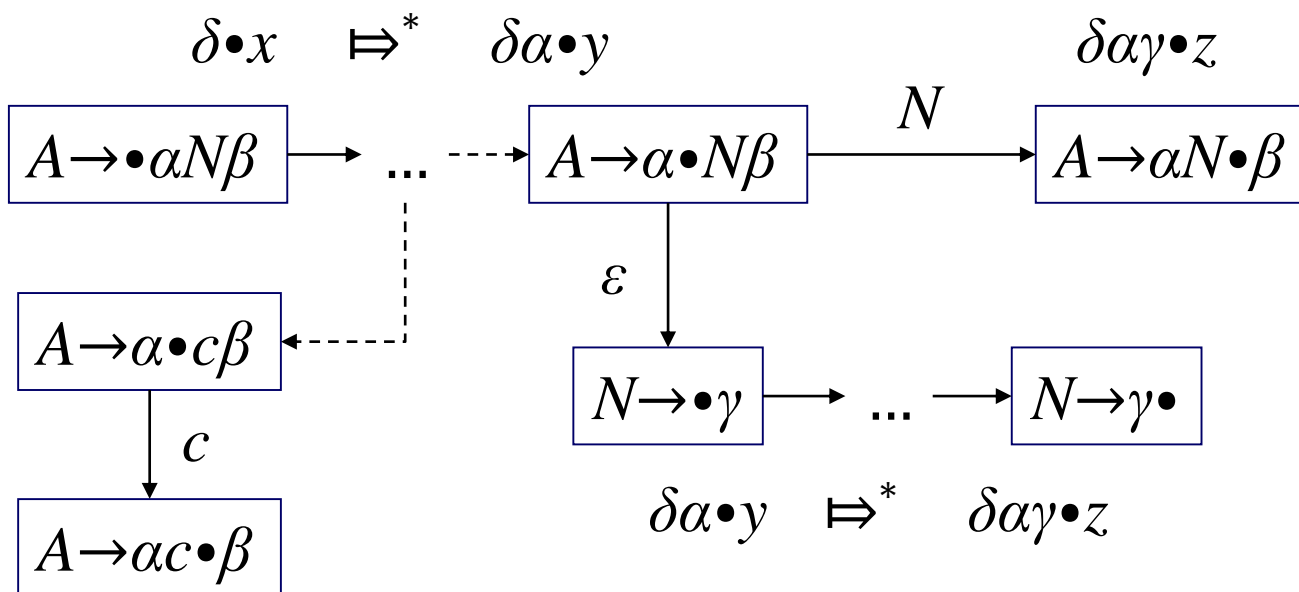
李永亮

itemNFA示例：移动与规范规约对应

王冬冬



$$\varepsilon \bullet W \Rightarrow^* S \bullet \varepsilon$$



$$\delta \alpha \bullet y \Rightarrow^* \delta \alpha \gamma \bullet z$$

$$\delta \bullet x \Rightarrow^* \delta \alpha \bullet c y \Rightarrow^* \delta \alpha c \bullet y$$

王冬冬

活前缀的可用项目与有效项目

2018

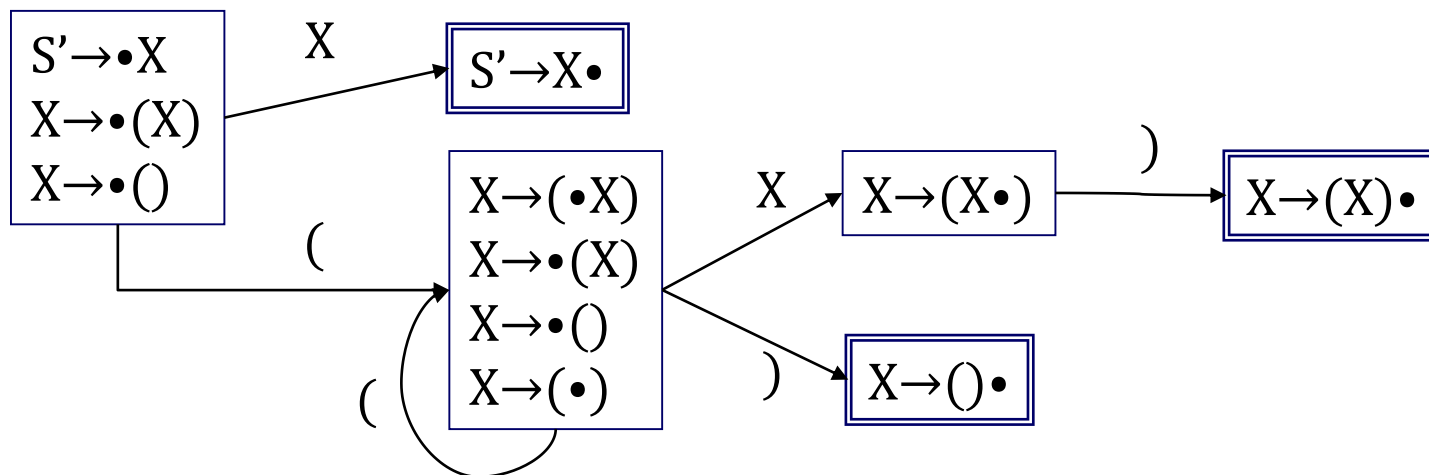
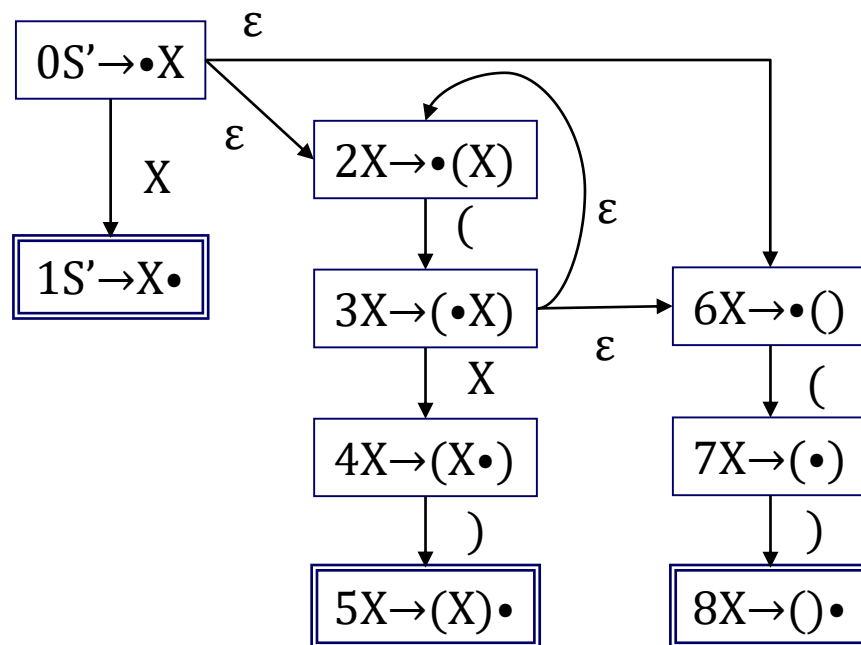
- ▶ 对于itemPDA, SRF栈内容 ρ 始终是规范句型 ρx 的活前缀。
- ▶ 项目 $A \rightarrow \gamma^{\text{past}} \cdot \gamma^{\text{future}}$ 被称为是活前缀 $\rho = \delta \gamma^{\text{past}}$ 的可用项目。
- ▶ 可用项目的意义在于表示这个活前缀 ρ 是在 $[A \rightarrow \gamma^{\text{past}} \cdot \gamma^{\text{future}}]$ 状态可能存在的, 这个状态是愿望之路 (目标 A 、许愿积跬步 γ 现栈顶) 上的一个节点, 不管该愿望是否达成。
- ▶ 若可用项目的愿望之路出现在一条成功的规范归约路径中, 就表明它是 ρ 的有效项目。 ρ 在 $[A \rightarrow \gamma^{\text{past}} \cdot \gamma^{\text{future}}]$ 状态肯定存在
- ▶ 项目 $A \rightarrow \gamma^{\text{past}} \cdot \gamma^{\text{future}}$ 的愿望之路:
 - 起点: 目标是“归约为 A ”; 愿望是“积跬步到 γ 成为栈顶”; 起点状态是 $[A \rightarrow \cdot \gamma]$ 。 终点状态是 $[A \rightarrow \gamma \cdot]$
- ▶ 愿望之路到规范归约步骤有一一对应关系:
$$\delta \cdot \dots x \xrightarrow{\text{green}} \rho \cdot x = \delta \gamma^{\text{past}} \cdot yz \xrightarrow{\text{red}} \delta \gamma \cdot z \xrightarrow{\text{blue}} \delta A \cdot z$$
- ▶ 愿望之路出现在一条成功的规范归约路径里:
$$w \xrightarrow{\text{blue}} \rho \cdot x = \delta \gamma^{\text{past}} \cdot yz \xrightarrow{\text{red}} \delta \gamma \cdot z \xrightarrow{\text{green}} \delta A \cdot z \xrightarrow{\text{green}} S \cdot \varepsilon$$

李永强

例：活前缀的有效项目

李永亮

$Z_0 \bullet (()) \#$ 0, 2, 6
 $\Rightarrow Z_0 (\bullet ()) \#$ 3, 2, 6, 7
 $\Rightarrow Z_0 ((\bullet)) \#$ 3, 2, 6, 7
 $\Rightarrow Z_0 () (\bullet) \#$ 8
 $\Rightarrow Z_0 (X \bullet) \#$ 4
 $\Rightarrow Z_0 (X) \bullet \#$ 5
 $\Rightarrow Z_0 X \bullet \#$ 1

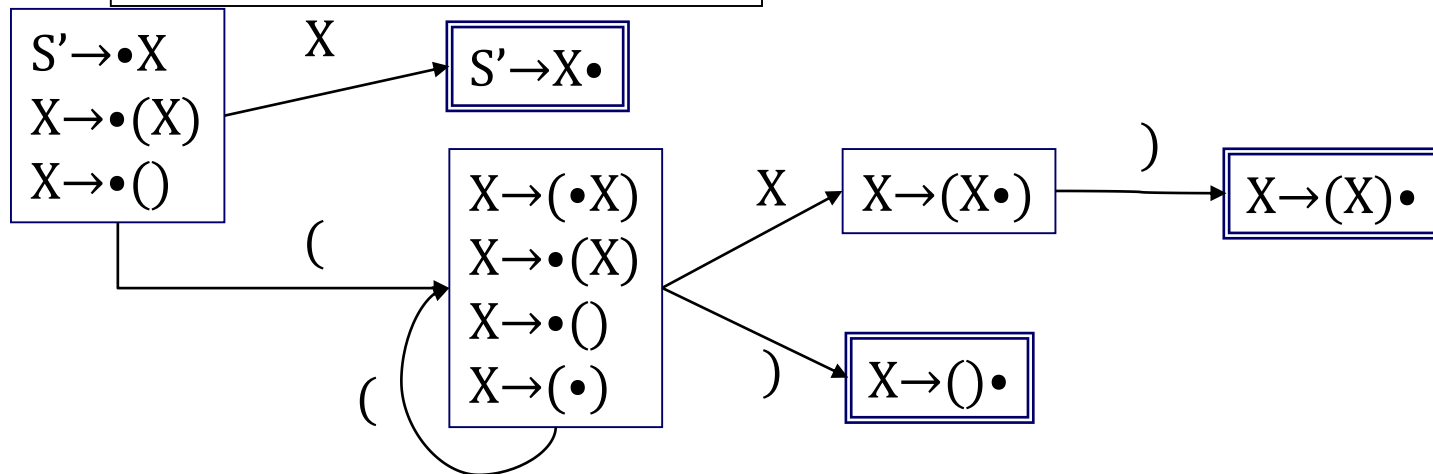
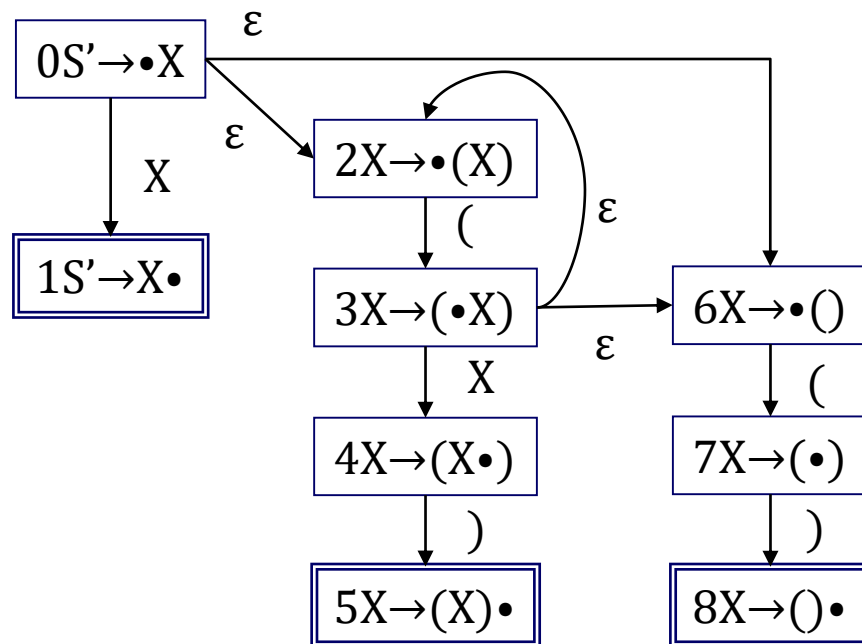


CFG:
 $S' \rightarrow X$
 $X \rightarrow (X)$
 $X \rightarrow ()$

李永亮

例：活前缀的有效项目

$Z_0 \bullet ((())) \#$ 0, 2, 6
 $\Rightarrow Z_0 (\bullet ((())) \#$ 3, 2, 6, 7
 $\Rightarrow Z_0 ((\bullet (())) \#$ 3, 2, 6, 7
 $\Rightarrow Z_0 (((\bullet)) \#$ 3, 2, 6, 7
 $\Rightarrow Z_0 (((()) \bullet) \#$ 8
 $\Rightarrow Z_0 ((X \bullet)) \#$ 4
 $\Rightarrow Z_0 ((X) \bullet) \#$ 5
 $\Rightarrow Z_0 (X \bullet) \#$ 4
 $\Rightarrow Z_0 (X) \bullet \#$ 5
 $\Rightarrow Z_0 X \bullet \#$ 1



CFG:
 $S' \rightarrow X$
 $X \rightarrow (X)$
 $X \rightarrow ()$

2016.12.2



有效项目集与规范簇

王

- ▶ 一个活前缀的所有有效项目构成的集合被称为识别该活前缀的**有效项目集**。
- ▶ 文法 G 的所有有效项目集构成的集合称为该文法的**LR(0)项目集规范簇**，简称规范簇或LR(0)规范簇。
- ▶ 考察 $L(G)$ 的每个元素（句子）的规范推导，收集到所有活前缀；再基于 $\text{itemset}(G)$ 文法确定每个活前缀的有效项目集，这些项目集构成的集合就是该文法的LR(0)规范簇。
- ▶ 任意一个活前缀都有唯一一个有效项目集；
- ▶ 规范归约过程中，活前缀 ρ 增长为活前缀 ρ' ，其中 $|\rho'|=|\rho|+1$ ，是因为 ρ 的有效项目集到 ρ' 的有效项目集有状态转移关系；
- ▶ itemNFA 和 itemDFA 都给出了有效项目集间的转移关系。
- ▶ itemDFA 的状态集合就是**LR(0)项目集规范簇**。

直接构造itemDFA

赵银亮

- ▶ 给定CFG $G=(V, T, \mathcal{P}, S)$, 构造itemDFA(Q, Σ, v, q_0, F)。
- ▶ itemDFA 的状态是项目集合
- ▶ 初始状态 $q_0 = \omega([S' \rightarrow \bullet S])$
 - 定义项目集 q 的 ε 闭包 $\omega(q)$ 如下:
如果 $[A \rightarrow \alpha \bullet N \beta] \in \omega(q)$, 那么 $[N \rightarrow \bullet \gamma] \in \omega(q)$,
其中 $(A, \alpha N \beta), (N, \gamma) \in \mathcal{P}$ 。
 - 直观地, itemNFA 中, $[A \rightarrow \alpha \bullet N \beta]$ 到 $[N \rightarrow \bullet \gamma]$ 有 ε 转移。
- ▶ 转移函数: 对于 $q \in Q$, 令 $p = v(q, X)$, $X \in V \cup T$, 那么,
 - $p = \omega(\{[A \rightarrow \alpha X \bullet \eta] \mid [A \rightarrow \alpha \bullet X \eta] \in q\})$,
 - $p \in Q$,
 - 若 p 仅含完全项目, 那么 $p \in F$ 。

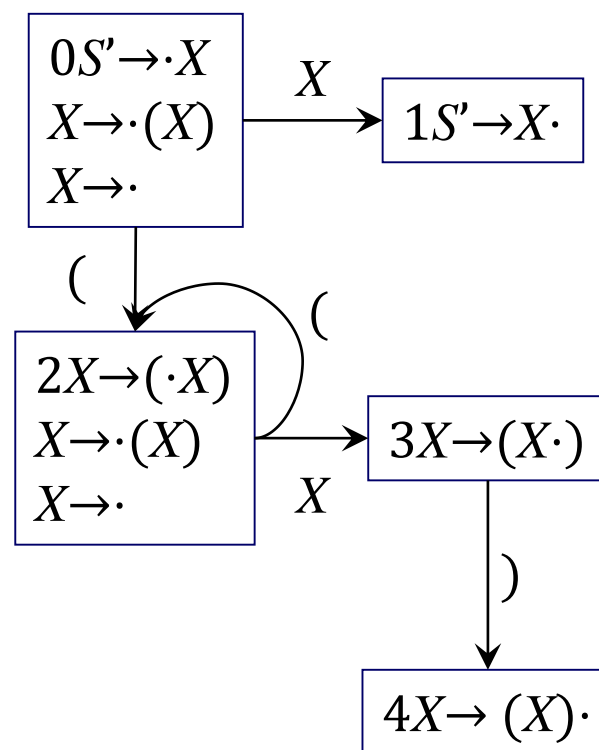
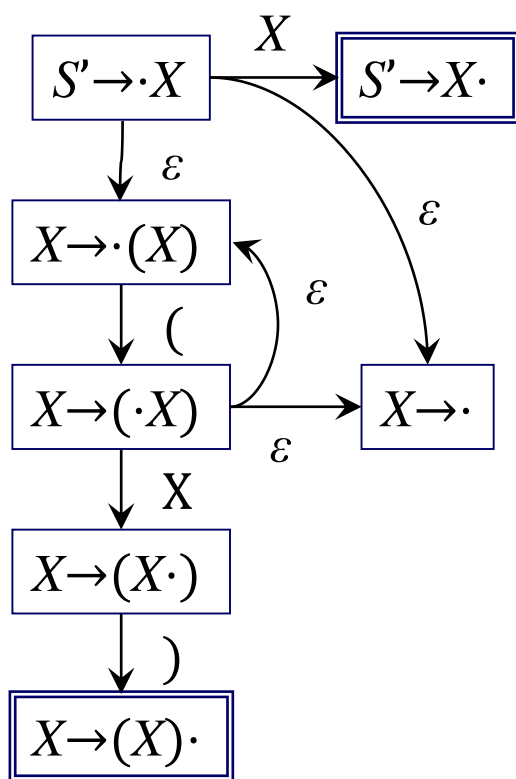
赵银亮

例: $X \rightarrow (X)$ $X \rightarrow \varepsilon$

构建itemNFA、itemDFA都使用增广文法。

建议直接构建itemDFA;

也可先构建itemNFA, 再确定化为itemDFA。



注意, ε 产生式的项目只有1个, 其愿望之路的起点即终点, 长度为0

构造itemDFA算法

王冬冬

输入：CFG $G=(V, T, P, S)$

输出：识别活前缀的DFA($Q, V \cup T, v, q_0, F$)。

$Q=\emptyset; F=\emptyset; v=\emptyset;$

将 $q_0=\omega([S' \rightarrow \bullet S])$ 加入 Q ;

while (Q 中存在未标记元素 q) {

 标记 q ;

 for ($X \in V \cup T$) { $p=\emptyset$;

 for ($s \in q \ \&\& \ s == [A \rightarrow \alpha \bullet X \eta]$) 将 $[A \rightarrow \alpha X \bullet \eta]$ 加入 p ;

 if ($p \neq \emptyset$) { $p = \omega(p)$; 将 p 加入 Q ;

 将 $((q, X), p)$ 加入 v ; } //即令 $v(q, X)=p$

 若 p 含完全项目，那么将 p 加入 F ; }

}

王冬冬

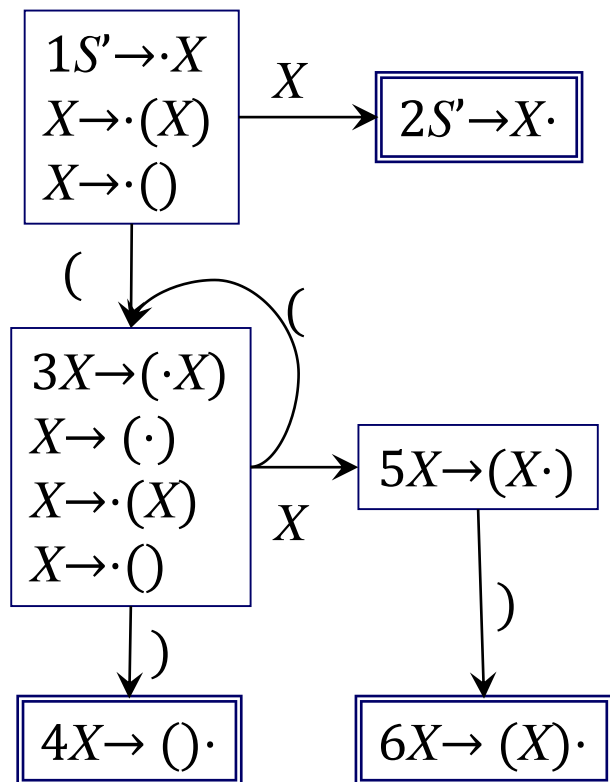
从itemDFA构造LR(0)分析表

王

- LR(0)分析表由ACTION[]和GOTO[]两个表组成：
- ACTION[m, a]=? 其中m为状态，a为当前输入符号
 - **sn**: 若itemDFA中有尾m头n权a的弧；
 - **acc**: m状态包含有接受项目；
 - **rk**: m状态含编号k的产生式的完全项目（非接受项目）。
- GOTO[m, A]=?
 - **n**: 若itemDFA中有尾m头n权a的弧。
- LR(0)分析表看起来是itemDFA的转移表表示，其中状态被排序并使用索引值替代状态引用，另外对终结符列添加了额外信息。
- 但是与用DFA判定输入串的运行方式不同，注意这里输入串是终结字符串。（相关理论：项目PDA）

例：LR(0)分析表

李永亮



状态	ACTION			GOTO
	()	#	X
1	s3			2
2			acc	
3	s3	s4		5
4	r3	r3	r3	
5		s6		
6	r2	r2	r2	

$1S' \rightarrow X$
 $2X \rightarrow (X)$
 $3X \rightarrow ()$

- 给定CFG;
- 先求出itemDFA;
- 再填ACTION表和填GOTO表;

李永亮

基于LR(0)分析表的分析过程

状态	ACTION			GOTO
	()	#	X
1	s3			2
2			acc	
3	s3	s4		5
4	r3	r3	r3	
5		s6		
6	r2	r2	r2	

ACTION[m, a]:

- sn : bi-push(n, a)
- rn : 按产生式 $nA \rightarrow \alpha$ 归约:
bi-pop($|\alpha|$); bi-top($m', X]$
bi-push((GOTO[m', A], A)).
- acc: 分析成功。

状态栈	符号栈	剩余串	动作
1	#	(())#	shift
13	#(()#	shift
133	#(())#	shift
1334	#(())#	reduce $X \rightarrow ()$
135	#(X)#	shift
1356	#(X)	#	reduce $X \rightarrow (X)$
12	#X	#	acc

1 $S' \rightarrow X$

2 $X \rightarrow (X)$

3 $X \rightarrow ()$

基于分析表的LR(0)分析算法

王

► 输入：分析表， w ，双栈空栈。 输出：分析成功与否。

bi-stack(1,#); //双栈初始化分别为初始状态1和符号#

scan(); //读入当前符号赋给全局变量tok

while(1){

bi-top(m , c); //将双栈栈顶符号分别赋给 m 和 c

if (ACTION[m , tok]==sn)bi-push(n , tok);

else if(ACTION[m , tok]==rk){

bi-pop(body-size(k)); //弹出产生式 k 的体长那么多

A=head(k); //产生式的头，即左部变元

bi-top(m' , c); bi-push(GOTO[m' , A], A);

}else if(ACTION[m , tok]==acc)break; //接受

else error(); //其它情况都属于错误

}

王

5.4 SLR(1)分析法

- ▶ LR(0)分析要求基于LR(0)文法进行,
- ▶ LR(0)文法对它的LR(0)规范簇的每一个有效项目集都要求没有冲突, 即:
 - 含有一个完全项目, 就不能再含有任何别的项目。
- ▶ 那么, SLR(1)分析的思想?
- ▶ 查看当前输入符号来消解LR(0)规范簇中的冲突:
 - 移进-归约冲突;
 - 归约-归约冲突。

2018

是SLR(1)?

1 $S' \rightarrow X$

2 $X \rightarrow (X)$

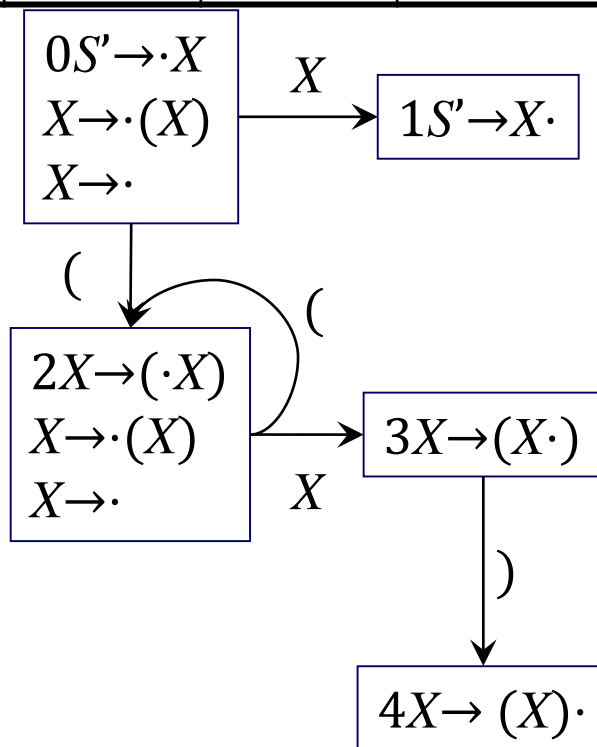
3 $X \rightarrow \varepsilon$

$\text{FIRST}(X) = \{ (, \varepsilon, \# \}$

$\text{FOLLOW}(X) = \{ \#,) \}$

	ACTION			GOTO
状态	()	#	X
0	s2/r3	r3	r3	1
1			acc	
2	s2/r3	r3	r3	3
3		s4		
4		r2	r2	

状态栈	符号栈	剩余串	动作
0	Z_0	$(()) \#$	shift
02	$Z_0($	$() \#$	shift
022	$Z_0(($	$) \#$	reduce $X \rightarrow \varepsilon$
0223	$Z_0((X$	$) \#$	shift
02234	$Z_0((X)$	$\#$	reduce $X \rightarrow (X)$
023	$Z_0(X$	$\#$	shift
0234	$Z_0(X)$	$\#$	reduce $X \rightarrow (X)$
01	Z_0X	$\#$	acc



李永强

有冲突的例子很常见

► 对LR(0)文法若 $\tau(\rho) \cap \text{ran}(\mathcal{P}) = \emptyset$ 则移进否则归约。

$$S \rightarrow En \quad S \rightarrow n \quad S \rightarrow \varepsilon \quad E \rightarrow n +$$

#•n+n#

\Rightarrow #•n+n# //移进归约冲突；选移进

\Rightarrow #n•+n# //移进归约冲突、归约归约冲突；选移进

\Rightarrow #n+•n# //移进归约冲突；选归约 $E \rightarrow n +$

\Rightarrow #E•n# //移进归约冲突；选移进

\Rightarrow #En•# //归约归约冲突；选归约 $S \rightarrow En$

\Rightarrow #S•# //分析成功

冲突消解思路

李永亮

$S' \rightarrow S$

$S \rightarrow En | n | \varepsilon$

$E \rightarrow n +$

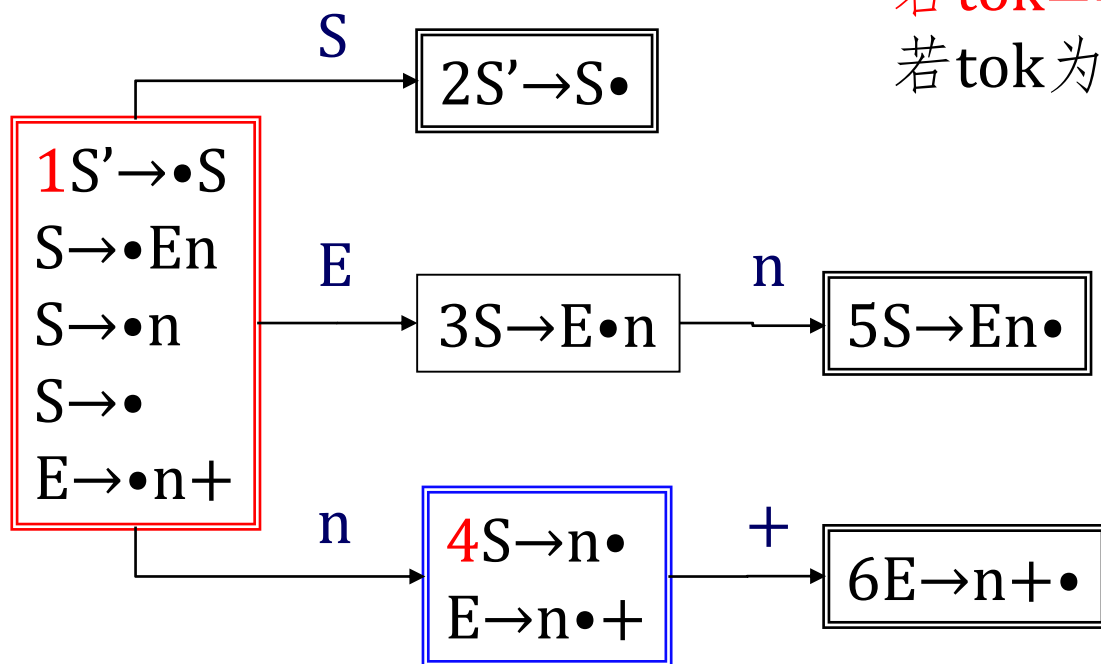
状态1移进归约冲突。

冲突消解思路：

若 $\text{tok} \in \text{FOLLOW}(S)$ 则归约；

若 $\text{tok} == n$ 则移进(推迟消解)；

若 tok 为其它则出错。



状态4移进归约冲突。

消解：

若 $\text{tok} == \#$ 则归约；

若 $\text{tok} == +$ 则移进；

若 tok 为其它则出错。

以上都属于SLR(1)消解方法，因此，这种文法是SLR(1)文法。

李永亮

举例：SLR(1)冲突消解规则

赵永亮

- ▶ itemDFA的当前状态包含 $[A \rightarrow \gamma \bullet]$
 - 如果当前输入符号 $a \in \text{FOLLOW}(A)$ ，那么用 $A \rightarrow \gamma$ 归约。
- ▶ itemDFA的当前状态包含 $\{X \rightarrow \alpha \bullet c \beta, A \rightarrow \alpha \bullet\}$ 且 $c \notin \text{FOLLOW}(A)$ ，
 - 若 $a = c$ 则移进；
 - 若 $a \in \text{FOLLOW}(A)$ 则用 $A \rightarrow \alpha$ 归约；
- ▶ itemDFA当前状态包含 $\{X \rightarrow \alpha \bullet c \beta, A \rightarrow \alpha \bullet, B \rightarrow \alpha \bullet\}$ 且 $c \notin \text{FOLLOW}(A)$ ， $c \notin \text{FOLLOW}(B)$ ， $\text{FOLLOW}(A) \cap \text{FOLLOW}(B) = \emptyset$ ，
 - 若 $a = c$ 则移进；
 - 若 $a \in \text{FOLLOW}(A)$ 则用 $A \rightarrow \alpha$ 归约；
 - 若 $a \in \text{FOLLOW}(B)$ 则用 $B \rightarrow \alpha$ 归约；
- ▶

赵永亮

例：构造SLR(1)分析表

$S \rightarrow E$

$E \rightarrow E+T$

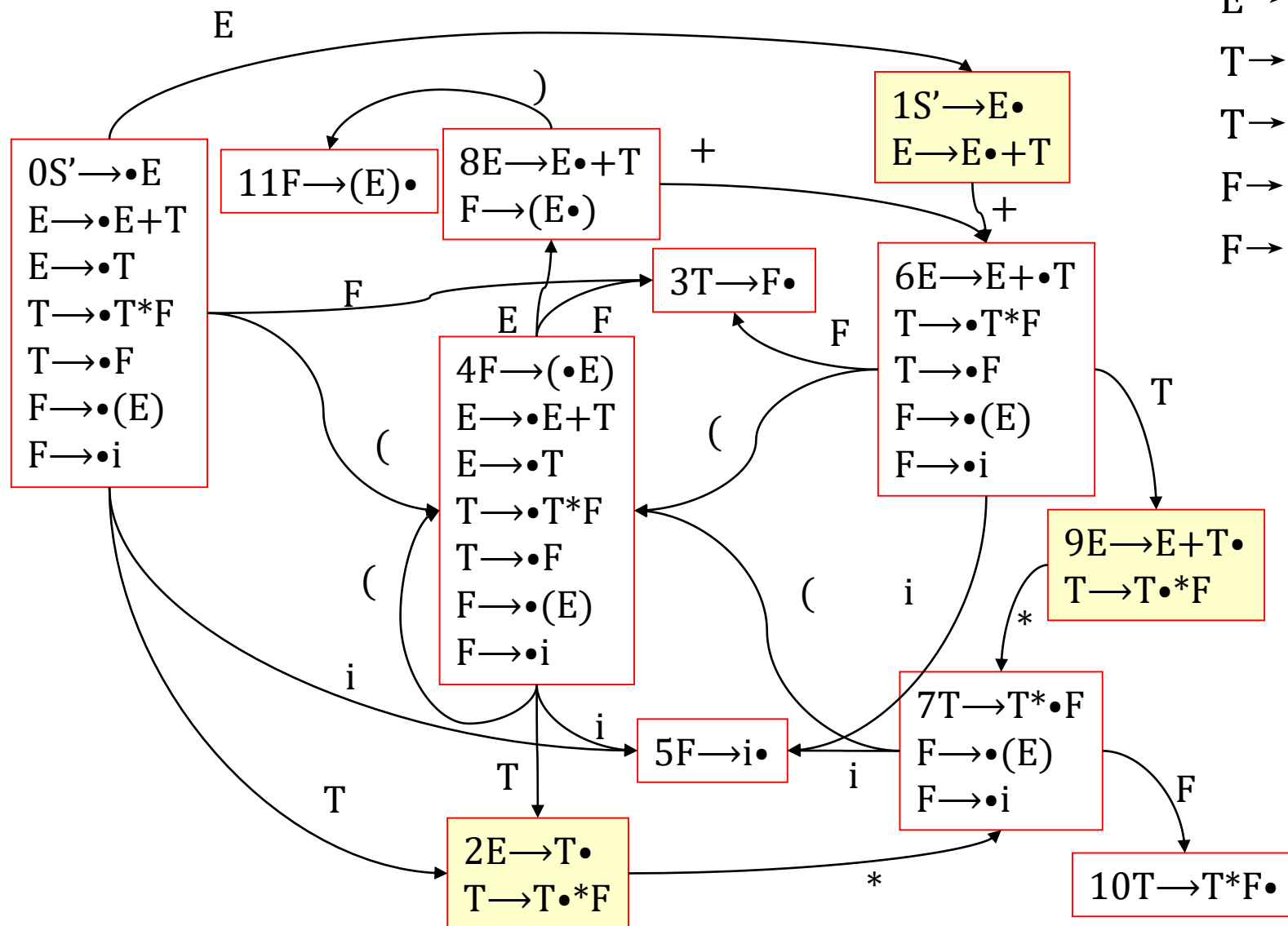
$E \rightarrow T$

$T \rightarrow T^*F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow i$



例：冲突消解与填表

0: $S \rightarrow E$

1: $E \rightarrow E + T$

2: $E \rightarrow T$

3: $T \rightarrow T * F$

4: $T \rightarrow F$

5: $F \rightarrow (E)$

6: $F \rightarrow i$

- 规范集0没有冲突，直接填表
- $ACTION[0,()] = s5$; $ACTION[0,i] = s4$
- $GOTO[0,E] = 1$; $GOTO[0,T] = 2$; $GOTO[0,F] = 3$
- 规范集1移进-归约冲突
- $+ \notin FOLLOW(S)$, $ACTION[1,+]=s6$,
- $FOLLOW(S) = \{\#\}$, $ACTION[1,\#]=acc$
- 规范集2移进-归约冲突
- $FOLLOW(E) = \{\#,+,)\}$,
 $ACTION[2,\#]=ACTION[2,+]=ACTION[2,)] = r2$;
- $ACTION[2,*]=s7$
- 规范集3无冲突
- $FOLLOW(T) = \{+,*),\#\}$,
 $ACTION[3,+]=ACTION[3,*]=ACTION[3,)] = ACTION[3,\#]=r4$

例：冲突消解与填表

0: $S \rightarrow E$

1: $E \rightarrow E+T$

2: $E \rightarrow T$

3: $T \rightarrow T * F$

4: $T \rightarrow F$

5: $F \rightarrow (E)$

6: $F \rightarrow i$

- 项目集4没有冲突，直接填表
- $\text{ACTION}[4,()] = s4$; $\text{ACTION}[4,i] = s5$
- $\text{GOTO}[4,E] = 8$; $\text{GOTO}[4,T] = 2$; $\text{GOTO}[4,F] = 3$
- 项目集5无冲突
- $\text{FOLLOW}(F) = \{+, *,), \#\}$,
 $\text{ACTION}[5,+] = \text{ACTION}[5,*] = \text{ACTION}[5,)] = \text{ACTION}[5,\#] = r6$
- 项目集6无冲突
- $\text{ACTION}[6,()] = s4$; $\text{ACTION}[6,i] = s5$
- $\text{GOTO}[6,T] = 9$; $\text{GOTO}[6,F] = 3$
- 项目集7无冲突
- $\text{ACTION}[7,()] = s4$; $\text{ACTION}[7,i] = s5$
- $\text{GOTO}[7,F] = 10$

例：冲突消解与填表

0: $S \rightarrow E$

1: $E \rightarrow E+T$

2: $E \rightarrow T$

3: $T \rightarrow T * F$

4: $T \rightarrow F$

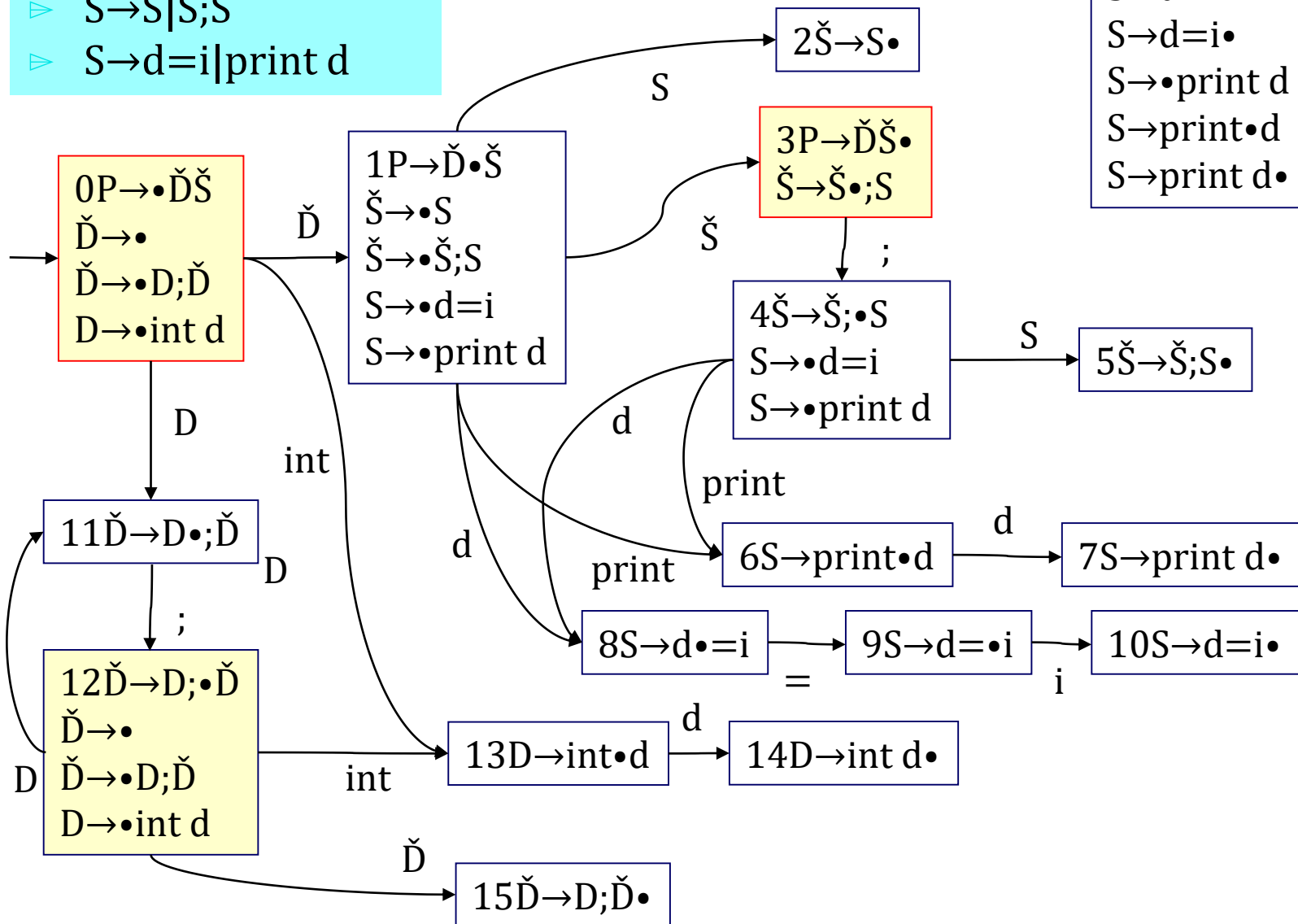
5: $F \rightarrow (E)$

6: $F \rightarrow i$

- 项目集8没有冲突，直接填表
- $ACTION[8,)] = s11$; $ACTION[4,+]=s6$
- $GOTO[4,E]=8$; $GOTO[4,T]=2$; $GOTO[4,F]=3$
- 项目集9移进-归约冲突
- $FOLLOW(E) = \{+,), \#\}$, $ACTION[9,*] = s7$
- $ACTION[9,+]=ACTION[9,)] = ACTION[9,\#] = r1$
- 项目集10无冲突
- $FOLLOW(T) = \{+,*), \#\}$, $ACTION[10,+]=ACTION[10,*] = ACTION[10,)] = ACTION[10,\#] = r3$
- 项目集11无冲突
- $FOLLOW(T) = \{+,*), \#\}$, $ACTION[11,+]=ACTION[11,*] = ACTION[11,)] = ACTION[11,\#] = r5$

- $P \rightarrow \check{D}\check{S}$
- $\check{D} \rightarrow \varepsilon | D; \check{D}$
- $D \rightarrow \text{int } d$
- $\check{S} \rightarrow S | \check{S}; S$
- $S \rightarrow d=i | \text{print } d$

例：构造DFA



$S \rightarrow \bullet d=i$
 $S \rightarrow d \bullet =i$
 $S \rightarrow d = \bullet i$
 $S \rightarrow d = i \bullet$
 $S \rightarrow \bullet \text{print } d$
 $S \rightarrow \text{print} \bullet d$
 $S \rightarrow \text{print } d \bullet$

$P \rightarrow \bullet \check{D}\check{S}$
 $P \rightarrow \check{D} \bullet \check{S}$
 $P \rightarrow \check{D}\check{S} \bullet$
 $\check{D} \rightarrow \bullet$
 $\check{D} \rightarrow \bullet D; \check{D}$
 $\check{D} \rightarrow D \bullet; \check{D}$
 $\check{D} \rightarrow D; \bullet \check{D}$
 $\check{D} \rightarrow D; \check{D} \bullet$
 $D \rightarrow \bullet \text{int } d$
 $D \rightarrow \text{int} \bullet d$
 $D \rightarrow \text{int } d \bullet$
 $\check{S} \rightarrow \bullet S$
 $\check{S} \rightarrow S \bullet$
 $\check{S} \rightarrow \bullet \check{S}; S$
 $\check{S} \rightarrow \check{S} \bullet; S$
 $\check{S} \rightarrow \check{S}; \bullet S$
 $\check{S} \rightarrow \check{S}; S \bullet$

例：SLR(1)分析表

	;	int	d	=	i	pri	#	P	Ď	Š	D	S
0		s13	r2			r2			1		11	
1			s8			s6				3		2
2	r5						r5					
3	s4						acc					
4			s8			s6						5
5	r6						r6					
6			s7									
7	r8						r8					
8				s9								
9					s10							
10	r7						r7					
11	s12											
12		s13	r2			r2			15		11	
13			s14									
14	r4											
15			r3			r3						

0: $P \rightarrow \bullet \check{D} \check{S}$
 $\check{D} \rightarrow \bullet$
 $\check{D} \rightarrow \bullet D; \check{D}$
 $D \rightarrow \bullet \text{int } d$

3: $P \rightarrow \check{D} \check{S} \bullet$
 $\check{S} \rightarrow \check{S} \bullet; S$

12: $\check{D} \rightarrow D; \bullet \check{D}$
 $\check{D} \rightarrow \bullet$
 $\check{D} \rightarrow \bullet D; \check{D}$
 $D \rightarrow \bullet \text{int } d$

1: $P \rightarrow \check{D} \check{S}$

2: $\check{D} \rightarrow \epsilon$

3: $\check{D} \rightarrow D; \check{D}$

4: $D \rightarrow \text{int } d$

5: $\check{S} \rightarrow S$

6: $\check{S} \rightarrow \check{S}; S$

7: $S \rightarrow d=i$

8: $S \rightarrow \text{print } d$

DFA当前状态包含

$\{X \rightarrow \alpha \bullet c \beta, A \rightarrow \alpha \bullet, B \rightarrow \alpha \bullet\}$ 且

$c \notin \text{FOLLOW}(A)$,

$c \notin \text{FOLLOW}(B)$,

$\text{FOLLOW}(A) \cap \text{FOLLOW}(B) = \emptyset$,

那么,

若 $a=c$ 则移进;

若 $a \in \text{FOLLOW}(A)$ 则用 $A \rightarrow \alpha$ 归约;

若 $a \in \text{FOLLOW}(B)$ 则用 $B \rightarrow \alpha$ 归约。

FIRST:

P {d,int,print}

Ď {ε,int}

Š {d,print}

D {int}

S {d,print}

FOLLOW:

P {#}

Ď {d,print}

Š {#,,}

D {;}

S {#,,}

SLR(1)消解简单冲突的通用规则

王

- ▶ 假定LR(0)规范簇的某个规范集中含有 m 个移进项目和 n 个归约项目：
 - $X_i \rightarrow \alpha \bullet c_i \beta_i, i=1, \dots, m,$
 - $A_i \rightarrow \alpha \bullet, i=1, \dots, n。$
- ▶ 该规范集满足：
 - 所有移进项目的点右终结符组成的集合 C 与任意完全项目的左部变元的FOLLOW集都不相交；
 - 任意两个完全项目的左部变元的FOLLOW集都不相交。
- ▶ 当前状态为该规范集、当前输入符号为 a 时，分析过程为：
 - 若 $a \in C$ 那么移进；否则，
 - 若 $a \in FOLLOW(A_k), k=1, \dots, \text{或} n,$ 则用产生式 $A_k \rightarrow \alpha$ 进行归约；否则，
 - 其它情况报错。

王

如何在SLR(1)中容忍其它冲突

赵银亮

- ▶ 定理：任何有歧义性的文法都不是LR文法。
- ▶ 证明略。
- ▶ 歧义性文法带来许多好处：
 - 规范簇更小；
 - 表示更简单；
 - 分析过程更简短等。
- ▶ 那么，考虑容忍歧义性的途径：
 - 让更多的冲突消解规则起作用；
 - 通过修剪文法推迟归约来回避风头（可能暂时难以解决）。

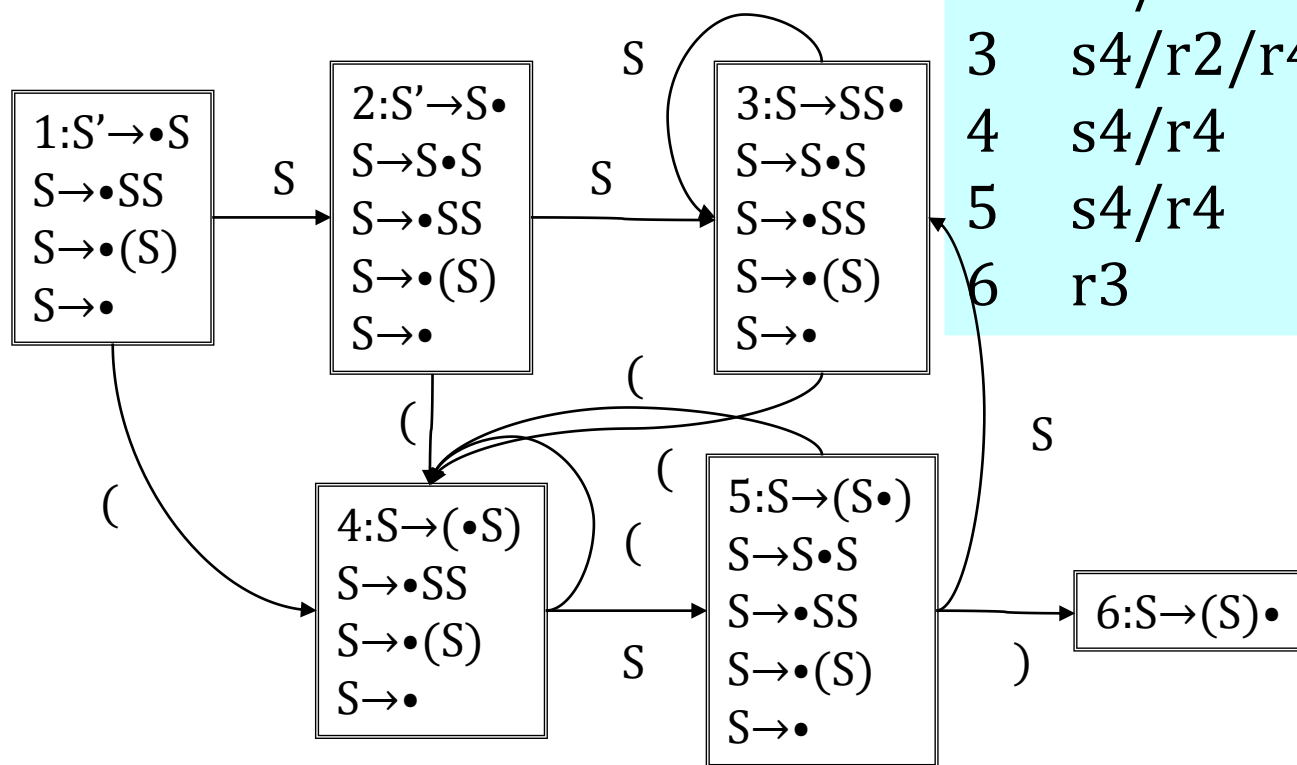
赵银亮

例：SLR(1)中其它冲突

2018

➤ $S' \rightarrow S$

➤ $S \rightarrow SS|(S)|\varepsilon$



	()	#	S
1	s4/r4	r4	r4	2
2	s4/r4	r4	acc	3
3	s4/r2/r4	r2/r4	r2/r4	3
4	s4/r4	r4	r4	5
5	s4/r4	s6/r4	r4	3
6	r3	r3	r3	

1: $S' \rightarrow S$
 2: $S \rightarrow SS$
 3: $S \rightarrow (S)$
 4: $S \rightarrow \varepsilon$

S'	$\{(\varepsilon\}$	$\{\#\}$
S	$\{(\varepsilon\}$	$\{(),\#\}$

2018

是文法歧义性导致的冲突

李永亮

1	#	()()#	s4
14	#()()#	r4
145	#(S)()#	s6
1456	#(S)	()#	r3
12	#S	()#	s4
124	#S()()#	r4
1245	#S(S)()#	s6
12456	#S(S))#	r3
123	#SS)#	r2
12	#S	()#	s4
124	#S()#	r4
1245	#S(S)#	s6
12456	#S(S)	#	r3
123	#SS	#	r2
12	#S	#	acc

	()	#	S
1	s4/r4	r4	r4	2
2	s4/r4	r4	acc	3
3	s4/r2/r4	r2/r4	r2/r4	3
4	s4/r4	r4	r4	5
5	s4/r4	s6/r4	r4	3
6	r3	r3	r3	

1: $S' \rightarrow S$
 2: $S \rightarrow SS$
 3: $S \rightarrow (S)$
 4: $S \rightarrow \varepsilon$

李永亮

为什么 $S \rightarrow SS$ 在程序设计语言中少见?

2018

- 靠LR(1)解决, 或,
- 文法中避免同一符号并列出现

1	#	((()))#	s4
14	#(((()))#	s4
144	#((((()))#	s4
1444	#(((((()))#	r4
14445	#(((S	((()))#	s4
144456	#(((S)	((()))#	r3
1445	#((S	((()))#	s6
12456	#((S)	((()))#	r3
123	#(S	((()))#	r2

	()	#	S
1	s4/r4	r4	r4	2
2	s4/r4	r4	acc	3
3	s4/r2/r4	r2/r4	r2/r4	3
4	s4/r4	r4	r4	5
5	s4/r4	s6/r4	r4	3
6	r3	r3	r3	

- 1: $S' \rightarrow S$
- 2: $S \rightarrow SS$
- 3: $S \rightarrow (S)$
- 4: $S \rightarrow \epsilon$

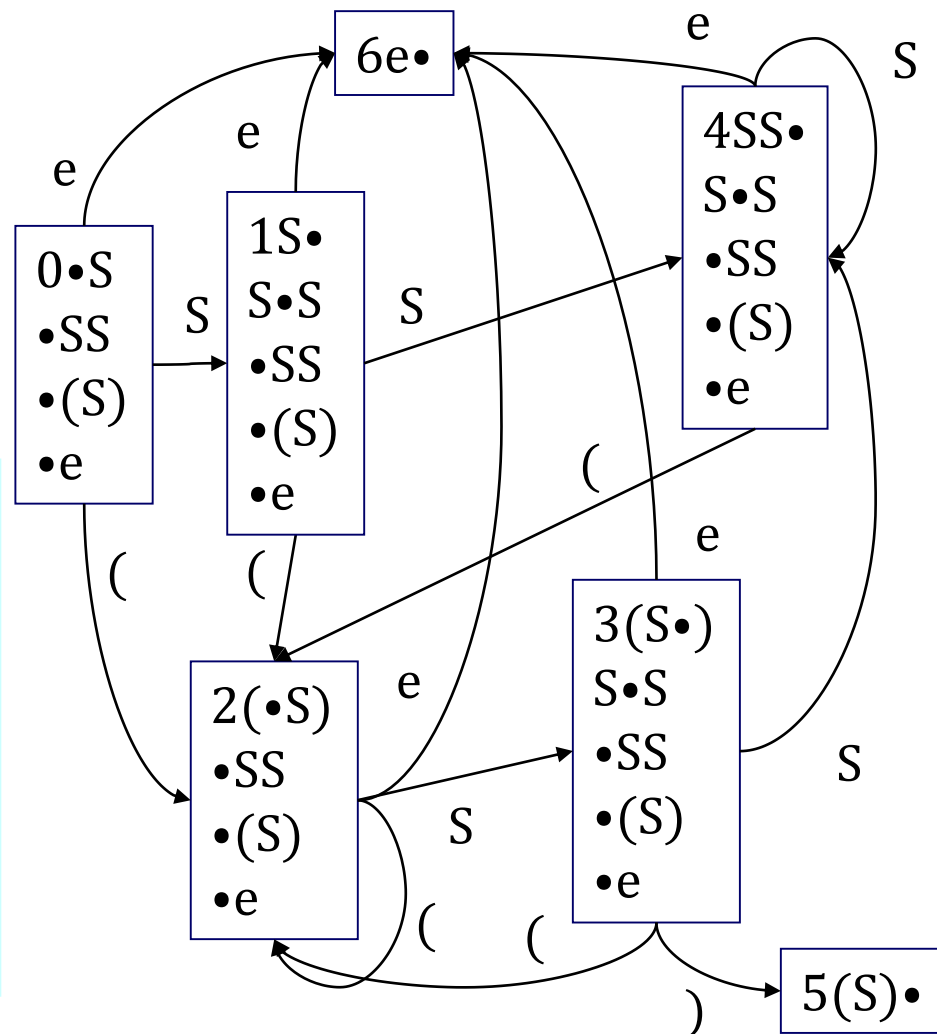
2018

文法 $G_{(e)}$

2018

$S' \rightarrow S$
 $S \rightarrow SS$
 $S \rightarrow (S)$
 $S \rightarrow e$

	e	()	#	S
0	s6	s2			1
1	s6	s2		acc	4
2	s6	s2			3
3	s6	s2	s5		4
4	s6	s2/r2	r2	r2	4
5		r3	r3	r3	
6	r4	r4	r4	r4	

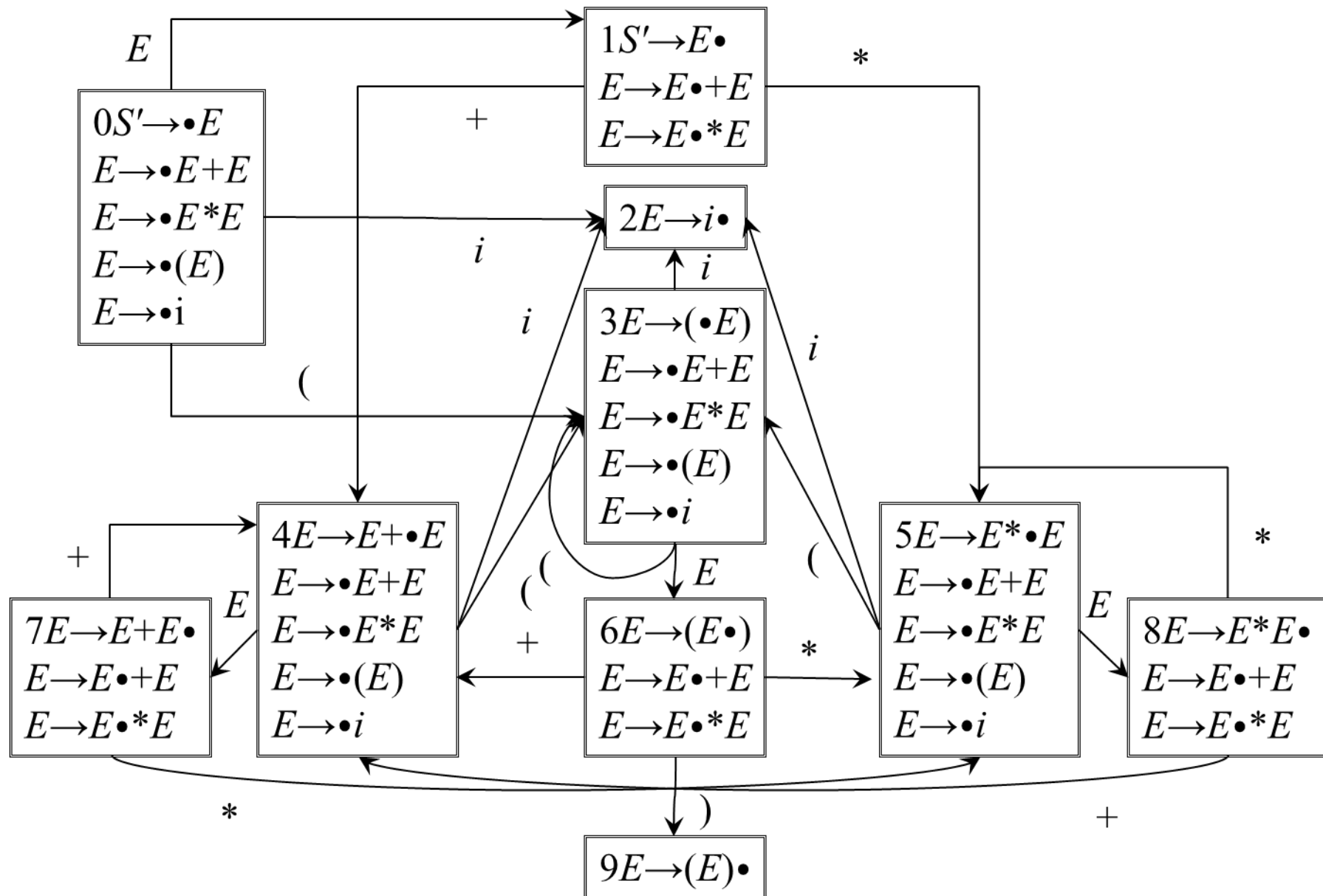


通过计算 $FOLLOW(S) = \{\#,), (, e\}$, 虽然SLR(1)默认冲突消解规则失效, 但是容易添加额外的规则, 就是有关左、右结合性之规则。具体来说, 如果消解为归约, 就表示句型 $\alpha SS \cdot (y$ 归约为 $\alpha S \cdot (y$, 这是从左往右的运算次序。如果消解为移进, 就表示从句型 $\alpha SS \cdot (y$ 到 $\alpha SS(\cdot y$, 其中活前缀由 αSS 变为 $\alpha SS($, 这是从右往左的运算次序。

李永强

例: $E \rightarrow E + E \mid E * E \mid (E) \mid i$

李永亮



李永亮

利用优先次序规则来消解冲突

赵永亮

- ▶ 状态1移进-归约冲突。用SLR(1)规则来消解。
- ▶ $E' \rightarrow E \bullet$ $E \rightarrow E \bullet + E \mid E \bullet * E$
- ▶ $FOLLOW(E') = \{\#\}$, tok为#时归约, 为+或*时移进。
- ▶ 状态7移进-归约冲突。用优先次序规则来消解。
- ▶ $E \rightarrow E + E \bullet \mid E \bullet + E \mid E \bullet * E$
- ▶ $FOLLOW(E) = \{\#, +, *,)\}$, tok为+、)、#时都归约, 为*时移进
- ▶ 等价于这个状态中没有第二个项目。第二个项目不起作用
- ▶ 状态8移进-归约冲突。用优先次序规则来消解。
- ▶ $E \rightarrow E * E \bullet \mid E \bullet + E \mid E \bullet * E$
- ▶ tok $\in FOLLOW(E)$ 时归约。
- ▶ 等价于这个状态中没有第二、三个项目。

赵永亮

悬挂else歧义性文法

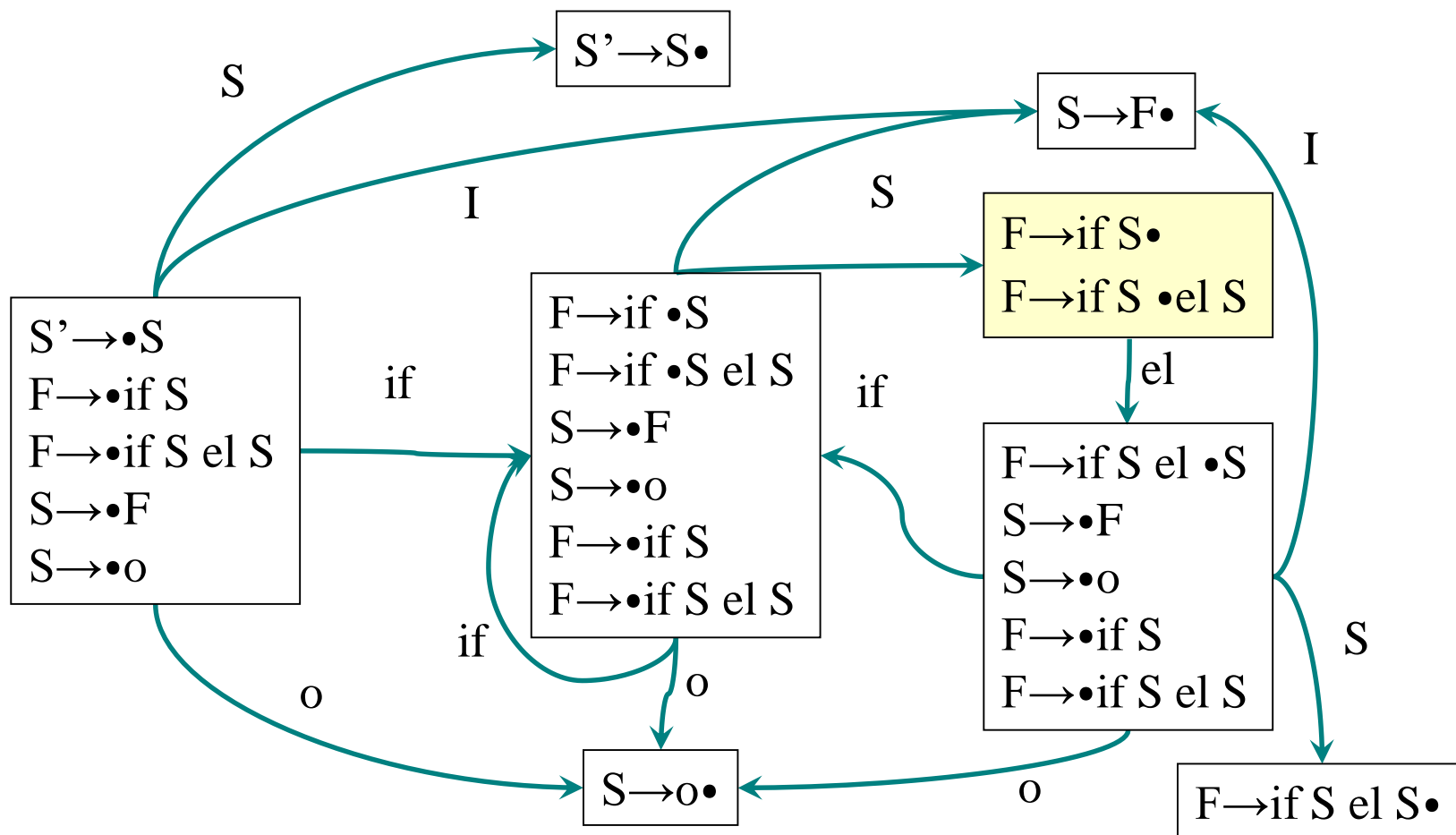
赵银亮

- ▶ $S \rightarrow F \mid o$
 - ▶ $F \rightarrow \text{if } S \mid \text{if } S \text{ el } S$
 - ▶ $\text{FOLLOW}(F) = \{\text{el}, \#\}$
-
- ▶ $\check{S} \rightarrow \check{S} ; S \mid S$
 - ▶ $S \rightarrow F \mid o$
 - ▶ $F \rightarrow \text{if } S \mid \text{if } S \text{ el } S$
 - ▶ $\text{FOLLOW}(F) = \{\text{el}, \#, ;\}$

用最近匹配原则来消解悬挂ELSE冲突

李永亮

- ▶ $FOLLOW(F) = \{el, \#\}$
- ▶ 在该状态时，若tok为el则移进，为#则归约



李永亮

其他容忍冲突方法：修剪文法推迟归约

2018

- ▶ 遇到移进-归约冲突不能消解时，修剪文法推迟归约，从而使
得移进项目和归约项目不同时在一个项目集里。
- ▶ 如： $A \rightarrow \alpha N \beta \mid \alpha \gamma \delta \quad N \rightarrow \gamma \mid \xi$
- ▶ 活前缀 $\alpha \gamma$ 的有效项目集含有： $N \rightarrow \gamma \bullet$ 和 $A \rightarrow \alpha \gamma \bullet \delta$
- ▶ 那么，当 $\text{FIRST}(\delta) \cap \text{FOLLOW}(N) = C \neq \emptyset$ 时，可能存在移进-归
约冲突，具体地，对于 $\text{tok} \in C$ 可以移进也可以按 $N \rightarrow \gamma$ 归约，如
果 δ 的首符集含有 c 的话。
- ▶ 修剪文法以消除此类冲突，
- ▶ $A \rightarrow \alpha \gamma \beta \mid \alpha \xi \beta \mid \alpha \gamma \delta$
- ▶ 此时，针对于活前缀 $\alpha \gamma$ 的冲突不存在了。事实上推后到归约
项目 $A \rightarrow \alpha \gamma \beta \bullet$ 和 $A \rightarrow \alpha \gamma \delta \bullet$ 了

李永强

SLR(1)局限性

赵银亮

- ▶ SLR(1)+冲突消解可处理几乎所有实用语言结构。
- ▶ 局限：
 - itemset= $\{X \rightarrow \alpha \bullet c \beta, Y \rightarrow \alpha \bullet b \gamma, A \rightarrow \alpha \bullet, B \rightarrow \alpha \bullet\}$
 - 无法满足 $\{c, b\} \cap \text{FOLLOW}(A) \cap \text{FOLLOW}(B) = \emptyset$

$S \rightarrow \text{id} \mid V := E$

$V \rightarrow \text{id}$

$E \rightarrow V \mid n$

$\text{Follow}(S) = \{\#\}$

$\text{Follow}(V) = \{:=, \#\}$

$S' \rightarrow \bullet S$

$S \rightarrow \bullet \text{id}$

$S \rightarrow \bullet V := E$

$V \rightarrow \bullet \text{id}$

id

$S \rightarrow \text{id} \bullet$

$V \rightarrow \text{id} \bullet$

赵银亮

- ▶ 规范句型，规范推导，规范归约，句柄，短语，直接短语；
- ▶ 活前缀，LR(0)规范簇，规范集；
- ▶ 自下而上分析面临的问题；
- ▶ itemDFA（识别活前缀的DFA）；
- ▶ 移进-归约冲突，归约-归约冲突；
- ▶ 冲突消解规则；
- ▶ LR(0)分析表，SLR(1)分析表；
- ▶ SLR(1)默认的冲突消解；
- ▶ SLR(1)的三种额外冲突消解规则；
- ▶ 自下而上分析的算法描述。
- ▶ 作业：习题8.1(2)、8.2(1)、8.3、大作业3（下页）

大作业（三）

2016

- ▶ 对下页的文法分别写出itemDFA、冲突消解规则、以及SLR(1)分析表，并指出不能消解的冲突都是哪些。
- ▶ 选择一种完成方式：（1）手工完成并上传雨课堂（2）写程序完成，即基于SLR(1)分析表的语法分析器代码。将代码、运行结果和文档合成一个文件发送到zyl9910@xjtu.edu.cn，附件名称为<群内名><报告名>。
- ▶ 对于本次大作业，建议原则上2人一组按照方式（1）进行，或3人一组按照方式（2）进行。3人组由1名组长和2名组员构成。

李银亮

$$P \rightarrow \check{D} \check{S}$$

$$\check{D} \rightarrow \varepsilon \mid \check{D} D ;$$

$$D \rightarrow T d \mid T d[i] \mid T d(\check{A}) \{ \check{D} \check{S} \}$$

$$T \rightarrow \text{int} \mid \text{void}$$

$$\check{A} \rightarrow \varepsilon \mid \check{A} A ;$$

$$A \rightarrow T d \mid d[] \mid T d()$$

$$\check{S} \rightarrow S \mid \check{S} ; S$$

$$S \rightarrow d = E \mid \text{if } (B) S \mid \text{if } (B) S \text{ else } S \mid \text{while } (B) S \mid \text{return } E \mid \{ \check{S} \} \mid d(\check{R})$$

$$B \rightarrow B \wedge B \mid B \vee B \mid E r E \mid E$$

$$E \rightarrow d = E \mid i \mid d \mid d(\check{R}) \mid E + E \mid E * E \mid (E)$$

$$\check{R} \rightarrow \varepsilon \mid \check{R} R ,$$

$$R \rightarrow E \mid d[] \mid d()$$

构造分析器示例

2016年

①写出识别活前缀的DFA;

②确定有哪些冲突并给出冲突消解规则;

③写出分析表及分析器代码,并测试完善。

```
11413: E→i*
11414: E→d•|d•[Ē]|d•(Ē)
115: S→while•(B)S
1151: S→while•(B)S    B→•ErE|•E    E→•i|•d|•d[Ē]|•d(Ē)|•E+E|•E*E
11511: S→while(B•)S
115111: S→while(B)•S    S→•d=E|•d[Ē]=E|•if(B)S|•if(B)SelseS|•while(B)S|•returnE|•{Ŝ}
1151111: S→while(B)S•
1151112: S→d•=E|d•[Ē]=E    等于 113
1151113: S→if•(B)S|if•(B)SelseS    等于 114
1151114: S→while•(B)S    等于 115
1151115: S→return•E    E→•i|•d|•d[Ē]|•d(Ē)|•E+E|•E*E    等于 116
1151116: S→{•Ŝ}    Ŝ→•S|Ŝ•S    S→•d=E|•d[Ē]=E|•if(B)S|•if(B)SelseS|•while(B)S|•returnE|•{Ŝ}    等于 117
11512: B→E•rE|E•    E→E•+E|E•*E    等于 11412
11513: E→i•    等于 1162
11514: S→d•|d•[Ē]|d•(Ē)    等于 11313
116: S→return•E    E→•i|•d|•d[Ē]|•d(Ē)|•E+E|•E*E
1161: S→returnE•    E→E•+E|E•*E
11611: E→E•+E    E→•i|•d|•d[Ē]|•d(Ē)|•E+E|•E*E
116111: E→E+E•    E→E•+E|E•*E
1161111: E→E•+E    E→•i|•d|•d[Ē]|•d(Ē)|•E+E|•E*E    等于 11611
1161112: E→E*•E    E→•i|•d|•d[Ē]|•d(Ē)|•E+E|•E*E    等于 11612
116112: E→i•    等于 1162
116113: E→d•|d•[Ē]|d•(Ē)    等于 1163
11612: E→E*•E    E→•i|•d|•d[Ē]|•d(Ē)|•E+E|•E*E
116121: E→E*E•    E→E•+E|E•*E
1161211: E→E•+E    E→•i|•d|•d[Ē]|•d(Ē)|•E+E|•E*E    等于 11611
1161212: E→E*•E    E→•i|•d|•d[Ē]|•d(Ē)|•E+E|•E*E    等于 11612
116122: E→i•    等于 1162
116123: E→d•|d•[Ē]|d•(Ē)    等于 1163
1162: E→i•
1163: E→d•|d•[Ē]|d•(Ē)
11631: E→d[Ē]•    Ē→•E|Ē•E    E→•i|•d|•d[Ē]|•d(Ē)|•E+E|•E*E
116311: E→d[Ē]•    Ē→Ē•E
1163111: E→d[Ē]•
1163112: Ē→Ē•E    E→•i|•d|•d[Ē]|•d(Ē)|•E+E|•E*E    等于 1132:
```

等于 1163

1163112: Ē→Ē•E E→•i|•d|•d[Ē]|•d(Ē)|•E+E|•E*E 等于 1132:

2016年

- ▶ LR(0)文法允许：
- ▶ ϵ -产生式
- ▶ 左递归变元
- ▶ 候选式有公共前缀的变元
- ▶ 单位产生式
- ▶ 无用符号
- ▶ 多候选式初始符号
- ▶ ϵ -产生式的项目是什么？既是完全项目又是初始项目
- ▶ 必须是增广文法

关于SLR(1)的结论

- ▶ 能消解的冲突都已经有了消解规则，并已经体现在分析表中
 - SLR(1)默认的冲突及消解；
 - 优先级冲突；悬挂**else**冲突；运算次序冲突；
 - 文法中没有变元并列的候选式。
- ▶ SLR(1)分析表已经消解了所有能消解的冲突。
- ▶ 仍然存在不为SLR(1)能消解的冲突，但已经充分了对于PL而言？

基于分析表的SLR(1)分析算法

赵银亮

► 输入：分析表， w ，双栈空栈。输出：分析成功与否。

```
bi-stack(1,#);
```

```
scan();
```

```
while(1){
```

```
    bi-top(m,c);
```

```
    if (ACTION[m,tok]==sn)bi-push(n,tok);
```

```
    else if(ACTION[m,tok]==rk){
```

```
        bi-pop(body-size(k));
```

```
        A=head(k);
```

```
        bi-top(m',c); bi-push(GOTO[m',A],A);
```

```
    }else if(ACTION[m,tok]==acc)break;
```

```
    else error();
```

```
}//与LR(0)的没有区别
```

赵银亮