

Parsing VI The LR(1) Table Construction

Building the Canonical Collection

Start from $s_0 = closure([S' \rightarrow S, EOF])$

Repeatedly construct new states, until all are found

The algorithm

 $s_{0} \leftarrow closure([S' \rightarrow S, EOF])$ $S \leftarrow \{ s_{0} \}$ $k \leftarrow 1$ while (S is still changing) $\forall s_{j} \in S \text{ and } \forall x \in (T \cup NT)$ $s_{k} \leftarrow goto(s_{j}, x)$ $record s_{j} \rightarrow s_{k} \text{ on } x$ if $s_{k} \notin S$ then $S \leftarrow S \cup s_{k}$ $k \leftarrow k + 1$

- Fixed-point computation
- > Loop adds to S
- > $S \subseteq 2^{(LR \text{ ITEMS})}$, so S is finite



Starts with S₀

$$s_0 \leftarrow closure(\{ [Goal \rightarrow \cdot Expr , EOF] \})$$

$$s_{0} \leftarrow closure([S' \rightarrow S, EOF])$$

$$S \leftarrow \{ s_{0} \}$$

$$k \leftarrow 1$$
while (S is still changing)

$$\forall s_{j} \in S \text{ and } \forall x \in (T \cup NT)$$

$$s_{k} \leftarrow goto(s_{j}, x)$$

$$record s_{j} \rightarrow s_{k} \text{ on } x$$
if $s_{k} \notin S$ then

$$S \leftarrow S \cup s_{k}$$

$$k \leftarrow k + 1$$





Starts with S₀

S₀ ← closure({ [Goal → • Expr , EOF] })
{ [Goal → • SheepNoise, EOF], [SheepNoise → • baa SheepNoise, EOF],
[SheepNoise → • baa, EOF]}

UNIVERSITY OF ELAWARE

Starts with S₀
S₀: { [Goal→ · SheepNoise, EOF], [SheepNoise→ · baa SheepNoise, EOF],
[SheepNoise→ · baa, EOF]}

Iteration 1 computes $S_1 = Goto(S_0, SheepNoise)$

UNIVERSITY OF ELAWARE

Starts with S₀
S₀: { [Goal→ · SheepNoise, EOF], [SheepNoise→ · baa SheepNoise, EOF],
[SheepNoise→ · baa, EOF]}

Iteration 1 computes
S₁ = Goto(S₀, SheepNoise) =
 {[Goal→ SheepNoise •, EOF]}

 $S_2 = Goto(S_0, \underline{baa})$



Starts with S₀
S₀: { [Goal→ · SheepNoise, EOF], [SheepNoise→ · baa SheepNoise, EOF],
[SheepNoise→ · baa, EOF]}

Iteration 1 computes
S₁ = Goto(S₀, SheepNoise) =
 { [Goal→ SheepNoise •, EOF]}

Starts with S_0

 $S_0: \{ [Goal \rightarrow \cdot SheepNoise, \underline{EOF}], [SheepNoise \rightarrow \cdot \underline{baa} \ \underline{SheepNoise} \rightarrow \cdot \underline{baa}, \underline{EOF}], \\ [SheepNoise \rightarrow \cdot \underline{baa}, \underline{EOF}] \}$

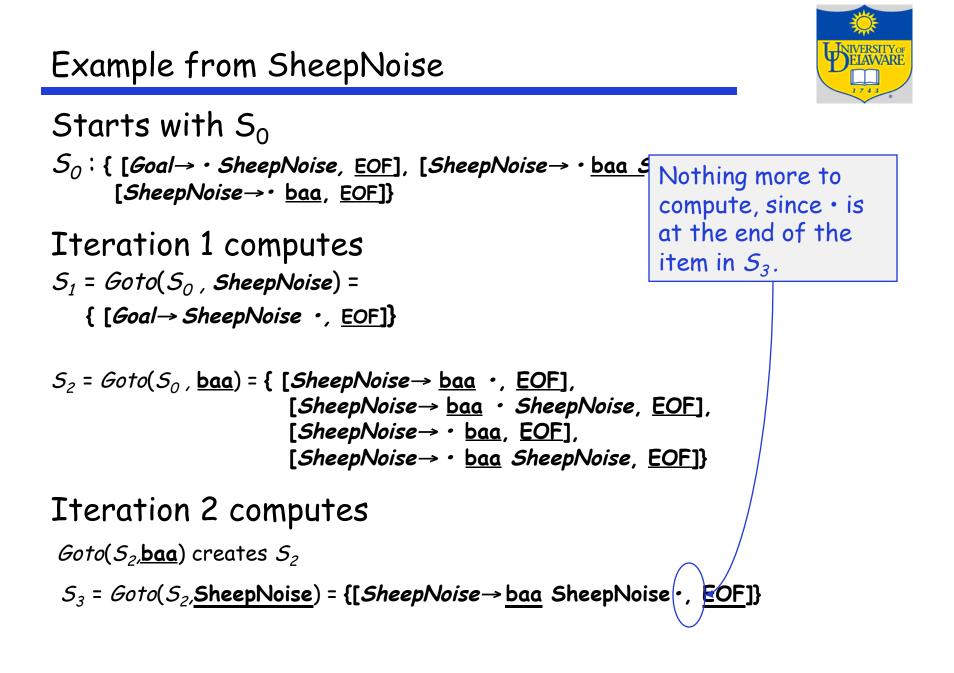
Iteration 1 computes $S_1 = Goto(S_0, SheepNoise) =$ { [Goal \rightarrow SheepNoise \cdot , EOF]}

Iteration 2 computes

 $Goto(S_2, \underline{baa})$ creates S_2

 $S_3 = Goto(S_2, SheepNoise) = \{[SheepNoise \rightarrow baa SheepNoise , EOF]\}$







Simplified, <u>right</u> recursive expression grammar





Initialization Step

 $S_0 \leftarrow closure(\{[Goal \rightarrow \cdot Expr, EOF]\})$

Goal → Expr Expr → Term - Expr Expr → Term Term → Factor * Term Term → Factor Factor → <u>ident</u>



Initialization Step

$$\begin{split} S_{0} \leftarrow closure(\{[Goal \rightarrow \cdot Expr , EOF]\}) \\ \{ [Goal \rightarrow \cdot Expr , EOF], [Expr \rightarrow \cdot Term - Expr , EOF], \\ [Expr \rightarrow \cdot Term , EOF], [Term \rightarrow \cdot Factor * Term , EOF], \\ [Term \rightarrow \cdot Factor * Term , -], [Term \rightarrow \cdot Factor , EOF], \\ [Term \rightarrow \cdot Factor , -], [Factor \rightarrow \cdot ident , EOF], \\ [Factor \rightarrow \cdot ident , -], [Factor \rightarrow \cdot ident , *] \} \end{split}$$

 $\boldsymbol{S} \leftarrow \{\boldsymbol{s}_0\}$



Initialization Step

$$\mathbf{S} \leftarrow \{\mathbf{s}_0\}$$

$$s_{0} \leftarrow closure([S' \rightarrow S, \underline{EOF}])$$

$$S \leftarrow \{ s_{0} \}$$

$$k \leftarrow 1$$
while (S is still changing)
$$\forall s_{j} \in S \text{ and } \forall x \in (T \cup NT)$$

$$s_{k} \leftarrow goto(s_{j}, x)$$

$$record s_{j} \rightarrow s_{k} \text{ on } x$$
if $s_{k} \notin S$ then
$$S \leftarrow S \cup s_{k}$$

$$k \leftarrow k + 1$$



Iteration 1

 $s_{1} \leftarrow goto(s_{0}, Expr)$ $s_{2} \leftarrow goto(s_{0}, Term)$ $s_{3} \leftarrow goto(s_{0}, Factor)$ $s_{4} \leftarrow goto(s_{0}, ident)$

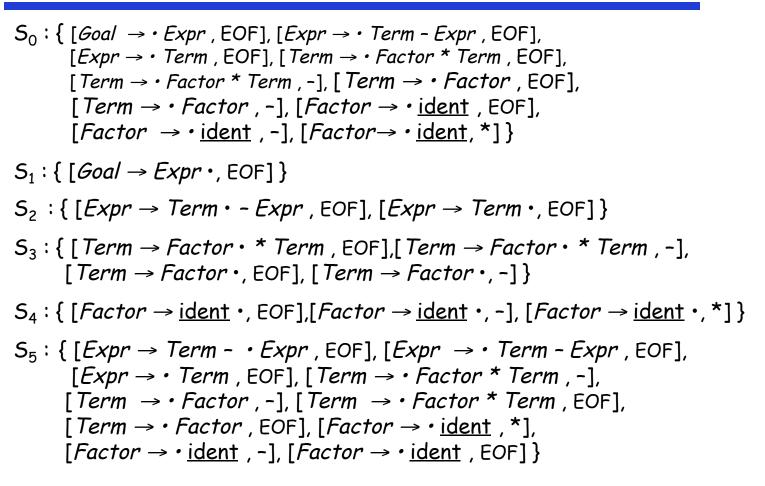
Let's just create sets s_1 through s_4

Iteration 2

Iteration 3

s₇ ← goto(s₅ , Expr) s₈ ← goto(s₆ , Term) $s_{0} \leftarrow closure([S' \rightarrow S, EOF])$ $S \leftarrow \{s_{0}\}$ $k \leftarrow 1$ while (S is still changing) $\forall s_{j} \in S \text{ and } \forall x \in (T \cup NT)$ $s_{k} \leftarrow goto(s_{j}, x)$ $record s_{j} \rightarrow s_{k} \text{ on } x$ if $s_{k} \notin S$ then $S \leftarrow S \cup s_{k}$ $k \leftarrow k + 1$

(Summary)





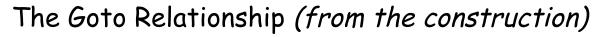
(Summary)



$$\begin{split} & \mathsf{S}_6: \{ [\mathit{Term} \rightarrow \mathit{Factor} * \cdot \mathit{Term} , \mathsf{EOF}], [\mathit{Term} \rightarrow \mathit{Factor} * \cdot \mathit{Term} , -], \\ & [\mathit{Term} \rightarrow \cdot \mathit{Factor} * \mathit{Term} , \mathsf{EOF}], [\mathit{Term} \rightarrow \cdot \mathit{Factor} * \mathit{Term} , -], \\ & [\mathit{Term} \rightarrow \cdot \mathit{Factor} , \mathsf{EOF}], [\mathit{Term} \rightarrow \cdot \mathit{Factor} , -], \\ & [\mathit{Factor} \rightarrow \cdot \mathit{ident} , \mathsf{EOF}], [\mathit{Factor} \rightarrow \cdot \mathit{ident} , -], [\mathit{Factor} \rightarrow \cdot \mathit{ident} , *] \} \\ & \mathsf{S}_7: \{ [\mathit{Expr} \rightarrow \mathit{Term} - \mathit{Expr} \cdot , \mathsf{EOF}] \} \end{split}$$

 $\mathsf{S}_8: \{ [\textit{Term} \rightarrow \textit{Factor} * \textit{Term} \cdot, \mathsf{EOF}], [\textit{Term} \rightarrow \textit{Factor} * \textit{Term} \cdot, -] \}$

(Summary)



State	Expr	Term	Factor	-	*	<u>Ident</u>
0	1	2	3			4

Iteration 1

$$s_{1} \leftarrow goto(s_{0}, Expr)$$

$$s_{2} \leftarrow goto(s_{0}, Term)$$

$$s_{3} \leftarrow goto(s_{0}, Factor)$$

$$s_{4} \leftarrow goto(s_{0}, ident)$$



(Summary)

The Goto Relationship (from the construction)

State	Expr	Term	Factor	-	*	<u>Ident</u>
0	1	2	3			4
1						
2				5		
3					6	
4						

Iteration 2

$$s_5 \leftarrow goto(s_2, -)$$

$$s_6 \leftarrow goto(s_3, *)$$



(Summary)

The Goto Relationship (from the construction)

State	Expr	Term	Factor	-	*	<u>Ident</u>
0	1	2	3			4
1						
2				5		
3					6	
4						
5	7	2	3			4
6		8	3			4
7						
8						

Iteration 3

s₇ ← goto(s₅ , Expr) s₈ ← goto(s₆ , Term) Iteration also creates duplicate states 2, 3, and 4.



Filling in the ACTION and GOTO Tables

VIVERSITY OF ELAWARE

The algorithm

$$\forall set s_{x*} \in S$$

$$\forall item i \in s_{x}$$

$$if i is [A \rightarrow \beta \cdot \underline{a}d, \underline{b}] and goto(s_{x}, \underline{a}) = s_{k}, \underline{a} \in T$$

$$then ACTION[x, \underline{a}] \leftarrow "shift k''$$

$$else if i is [A \rightarrow \beta \cdot, \underline{a}]$$

$$then ACTION[x, \underline{a}] \leftarrow "reduce A \rightarrow \beta''$$

$$else if i is [S' \rightarrow S \cdot, EOF]$$

$$then ACTION[x, \underline{a}] \leftarrow "accept''$$

$$\forall n \in NT$$

$$if goto(s_{x}, n) = s_{k}$$

$$then GOTO[x, n] \leftarrow k$$

Many items generate no table entry

 \rightarrow *Closure()* instantiates FIRST(X) directly for $[A \rightarrow \beta \cdot X \delta, \underline{a}]$



(Filling in the tables)

Example

The algorithm produces the following table

	ACTION				GOTO		
	<u>Ident</u>	-	*	EOF	Expr	Term	Factor
0	s 4				1	2	3
1				مدد			
2		s 5		r 3			
3		r 5	s 6	r 5			
4		r 6	r 6	r 6			
5	s 4				7	2	3
6	s 4					8	3
7				r 2			
8		r 4		r 4			

Plugs into the skeleton LR(1) parser

What can go wrong?

What if set s contains $[A \rightarrow \beta \cdot \underline{a}\gamma, \underline{b}]$ and $[B \rightarrow \beta \cdot, \underline{a}]$?

- First item generates "shift", second generates "reduce"
- Both define ACTION[s,<u>a</u>] cannot do both actions
- This is a fundamental ambiguity, called a *shift/reduce error*
- Modify the grammar to eliminate it
- Shifting will often resolve it correctly

What is set s contains $[A \rightarrow \gamma, \underline{a}]$ and $[B \rightarrow \gamma, \underline{a}]$?

- Each generates "reduce", but with a different production
- Both define ACTION[s,<u>a</u>] cannot do both reductions
- This fundamental ambiguity is called a *reduce/reduce error*
- Modify the grammar to eliminate it

In either case, the grammar is not LR(1)



EaC includes a worked example

(if-then-else)

Shrinking the Tables

Three options:

- Combine terminals such as <u>number</u> & <u>identifier</u>, <u>+</u> & <u>-</u>, <u>*</u> & <u>/</u>
 - \rightarrow Directly removes a column, may remove a row
 - → For expression grammar, 198 (vs. 384) table entries
- Combine rows or columns
 - \rightarrow Implement identical rows once & remap states
 - \rightarrow Requires extra indirection on each lookup
 - \rightarrow Use separate mapping for ACTION & for GOTO
- Use another construction algorithm
 - \rightarrow Both LALR(1) and SLR(1) produce smaller tables
 - → Implementations are readily available

