Parsing VI
The LR(1) Table Construction

Building the Canonical Collection

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

Repeatedly construct new states, until all are found

The algorithm

$$
\begin{array}{l}
s_0 \leftarrow closure(\,[S' \rightarrow S, \underline{EOF}]\,) \\
S \leftarrow \{\, s_0 \,\} \\
k \leftarrow 1 \\
while\ (\,S \text{ is still changing}\,) \\
\quad \forall s_j \in S \text{ and } \forall x \in (\,T \cup NT\,) \\
\qquad s_k \leftarrow goto(s_j, x) \\
\qquad record\ s_j \rightarrow s_k \text{ on } x \\
\quad if\ \ s_k \notin S \text{ then} \\
\qquad S \leftarrow S \cup s_k \\
\qquad k \leftarrow k + 1
\end{array}
$$

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

Example from SheepNoise

Starts with S_0

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

$s_0 \leftarrow closure(\ [S' \rightarrow S, \underline{EOF}]\ )$
$S \leftarrow \{\ s_0\ \}$
$k \leftarrow 1$

while (S *is still changing*)
  $\forall s_j \in S$ *and* $\forall x \in (\ T \cup NT)$
    $s_k \leftarrow goto(s_j, x)$
    record $s_j \rightarrow s_k$ *on* x
  if $s_k \notin S$ *then*
    $S \leftarrow S \cup s_k$
    $k \leftarrow k + 1$

Example from SheepNoise

Starts with S_0

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

{ [*Goal*→ • *SheepNoise*, EOF], [*SheepNoise*→ • baa *SheepNoise*, EOF],
[*SheepNoise*→ • baa, EOF]}

Example from SheepNoise

Starts with S_0

S_0 : { [***Goal*** → • ***SheepNoise***, EOF], [***SheepNoise*** → • baa ***SheepNoise***, EOF],
[***SheepNoise*** → • baa, EOF]}

Iteration 1 computes

S_1 = *Goto*(S_0 , ***SheepNoise***)

Example from SheepNoise

Starts with S_0

S_0 : { [*Goal*→ • ***SheepNoise***, <u>EOF</u>], [***SheepNoise***→ • <u>**baa**</u> ***SheepNoise***, <u>EOF</u>],
[***SheepNoise***→• <u>**baa**</u>, <u>EOF</u>]}

Iteration 1 computes

S_1 = *Goto*(S_0 , ***SheepNoise***) =
{ [*Goal*→ ***SheepNoise*** •, <u>EOF</u>]}

S_2 = *Goto*(S_0 , <u>**baa**</u>)

Example from SheepNoise

Starts with S_0

S_0 : { [*Goal*→ • *SheepNoise*, EOF], [*SheepNoise*→ • baa *SheepNoise*, EOF],
[*SheepNoise*→ • baa, EOF]}

Iteration 1 computes

S_1 = *Goto*(S_0 , *SheepNoise*) =
{ [*Goal*→ *SheepNoise* •, EOF]}

S_2 = *Goto*(S_0 , baa) = { [*SheepNoise*→ baa •, EOF],
[*SheepNoise*→ baa • *SheepNoise*, EOF],
[*SheepNoise*→ • baa, EOF],
[*SheepNoise*→ • baa *SheepNoise*, EOF]}

Example from SheepNoise

Starts with S_0

S_0 : { [*Goal*→ • ***SheepNoise***, <u>EOF</u>], [***SheepNoise***→ • <u>baa</u> ***SheepNoise***, <u>EOF</u>],
[***SheepNoise***→ • <u>baa</u>, <u>EOF</u>]}

Iteration 1 computes

S_1 = *Goto*(S_0 , ***SheepNoise***) =
{ [*Goal*→ ***SheepNoise*** •, <u>EOF</u>]}

S_2 = *Goto*(S_0 , <u>baa</u>) = { [***SheepNoise***→ <u>baa</u> •, <u>EOF</u>],
[***SheepNoise***→ <u>baa</u> • ***SheepNoise***, <u>EOF</u>],
[***SheepNoise***→ • <u>baa</u>, <u>EOF</u>],
[***SheepNoise***→ • <u>baa</u> ***SheepNoise***, <u>EOF</u>]}

Iteration 2 computes

Goto(S_2,<u>baa</u>) creates S_2

S_3 = *Goto*(S_2,<u>SheepNoise</u>) = {[***SheepNoise***→ <u>baa</u> SheepNoise •, <u>EOF</u>]}

Example from SheepNoise

Starts with S_0

S_0 : { [*Goal*→ • *SheepNoise*, EOF], [*SheepNoise*→ • baa S
[*SheepNoise*→• baa, EOF]}

Nothing more to compute, since • is at the end of the item in S_3.

Iteration 1 computes

S_1 = *Goto*(S_0 , *SheepNoise*) =
{ [*Goal*→ *SheepNoise* •, EOF]}

S_2 = *Goto*(S_0 , baa) = { [*SheepNoise*→ baa •, EOF],
[*SheepNoise*→ baa • *SheepNoise*, EOF],
[*SheepNoise*→ • baa, EOF],
[*SheepNoise*→ • baa *SheepNoise*, EOF]}

Iteration 2 computes

Goto(S_2,baa) creates S_2

S_3 = *Goto*(S_2,SheepNoise) = {[*SheepNoise*→ baa SheepNoise •, EOF]}

Example

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

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

Example (*building the collection*)

Initialization Step

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

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

Example (*building the collection*)

Initialization Step

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

$\{ [Goal \rightarrow \cdot Expr \text{ , EOF}], [Expr \rightarrow \cdot Term - Expr \text{ , EOF}],$
$[Expr \rightarrow \cdot Term \text{ , EOF}], [Term \rightarrow \cdot Factor * Term \text{ , EOF}],$
$[Term \rightarrow \cdot Factor * Term \text{ , –}], [Term \rightarrow \cdot Factor \text{ , EOF}],$
$[Term \rightarrow \cdot Factor \text{ , –}], [Factor \rightarrow \cdot \underline{\text{ident}} \text{ , EOF}],$
$[Factor \rightarrow \cdot \underline{\text{ident}} \text{ , –}], [Factor \rightarrow \cdot \underline{\text{ident}} \text{ , *}] \}$

$S \leftarrow \{ s_0 \}$

Example (*building the collection*)

Initialization Step

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

$\{ [Goal \rightarrow \cdot\, Expr \text{ , EOF}], [Expr \rightarrow \cdot\, Term - Expr \text{ , EOF}],$
$[Expr \rightarrow \cdot\, Term \text{ , EOF}], [Term \rightarrow \cdot\, Factor * Term \text{ , EOF}],$
$[Term \rightarrow \cdot\, Factor * Term \text{ , –}], [Term \rightarrow \cdot\, Factor \text{ , EOF}],$
$[Term \rightarrow \cdot\, Factor \text{ , –}], [Factor \rightarrow \cdot\, \underline{\text{ident}} \text{ , EOF}],$
$[Factor \rightarrow \cdot\, \underline{\text{ident}} \text{ , –}], [Factor \rightarrow \cdot\, \underline{\text{ident}} \text{ , *}] \}$

$S \leftarrow \{ s_0 \}$

$s_0 \leftarrow closure([S' \rightarrow S, \underline{\text{EOF}}])$
$S \leftarrow \{ s_0 \}$
$k \leftarrow 1$

while (S *is still changing*)
  $\forall s_j \in S$ *and* $\forall x \in (T \cup NT)$
    $s_k \leftarrow goto(s_j, x)$
    record $s_j \rightarrow s_k$ *on* x
  if $s_k \notin S$ *then*
    $S \leftarrow S \cup s_k$
    $k \leftarrow k + 1$

Example (*building the collection*)

Iteration 1

$s_1 \leftarrow goto(s_0, Expr)$

$s_2 \leftarrow goto(s_0, Term)$

$s_3 \leftarrow goto(s_0, Factor)$

$s_4 \leftarrow goto(s_0, \underline{\text{ident}})$

Iteration 2

$s_5 \leftarrow goto(s_2, -)$

$s_6 \leftarrow goto(s_3, *)$

Iteration 3

$s_7 \leftarrow goto(s_5, Expr)$

$s_8 \leftarrow goto(s_6, Term)$

Let's just create sets s_1 *through* s_4

$s_0 \leftarrow closure([S' \rightarrow S, \underline{\text{EOF}}])$
$S \leftarrow \{ s_0 \}$
$k \leftarrow 1$

while (S *is still changing*)
  $\forall s_j \in S$ and $\forall x \in (T \cup NT)$
    $s_k \leftarrow goto(s_j, x)$
    record $s_j \rightarrow s_k$ *on* x
  if $s_k \notin S$ *then*
    $S \leftarrow S \cup s_k$
    $k \leftarrow k + 1$

Example (*Summary*)

S_0 : { [*Goal* → • *Expr* , EOF], [*Expr* → • *Term* – *Expr* , EOF],
[*Expr* → • *Term* , EOF], [*Term* → • *Factor* * *Term* , EOF],
[*Term* → • *Factor* * *Term* , –], [*Term* → • *Factor* , EOF],
[*Term* → • *Factor* , –], [*Factor* → • <u>ident</u> , EOF],
[*Factor* → • <u>ident</u> , –], [*Factor* → • <u>ident</u>, *] }

S_1 : { [*Goal* → *Expr* •, EOF] }

S_2 : { [*Expr* → *Term* • – *Expr* , EOF], [*Expr* → *Term* •, EOF] }

S_3 : { [*Term* → *Factor* • * *Term* , EOF],[*Term* → *Factor* • * *Term* , –],
[*Term* → *Factor* •, EOF], [*Term* → *Factor* •, –] }

S_4 : { [*Factor* → <u>ident</u> •, EOF],[*Factor* → <u>ident</u> •, –], [*Factor* → <u>ident</u> •, *] }

S_5 : { [*Expr* → *Term* – • *Expr* , EOF], [*Expr* → • *Term* – *Expr* , EOF],
[*Expr* → • *Term* , EOF], [*Term* → • *Factor* * *Term* , –],
[*Term* → • *Factor* , –], [*Term* → • *Factor* * *Term* , EOF],
[*Term* → • *Factor* , EOF], [*Factor* → • <u>ident</u> , *],
[*Factor* → • <u>ident</u> , –], [*Factor* → • <u>ident</u> , EOF] }

Example (*Summary*)

S_6 : { [*Term* → *Factor* * • *Term* , EOF], [*Term* → *Factor* * • *Term* , –],
[*Term* → • *Factor* * *Term* , EOF], [*Term* → • *Factor* * *Term* , –],
[*Term* → • *Factor* , EOF], [*Term* → • *Factor* , –],
[*Factor* → • <u>ident</u> , EOF], [*Factor* → • <u>ident</u> , –], [*Factor* → • <u>ident</u> , *] }

S_7: { [*Expr* → *Term* – *Expr* •, EOF] }

S_8 : { [*Term* → *Factor* * *Term* •, EOF], [*Term* → *Factor* * *Term* •, –] }

Example (*Summary*)

The Goto Relationship *(from the construction)*

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

Iteration 1

$s_1 \leftarrow goto(s_0\, , Expr)$

$s_2 \leftarrow goto(s_0\, , Term)$

$s_3 \leftarrow goto(s_0\, , Factor)$

$s_4 \leftarrow goto(s_0\, , \underline{\text{ident}}\ )$

Example

(Summary)

The Goto Relationship *(from the construction)*

State	Expr	Term	Factor	-	*	Ident
0	1	2	3			4
1						
2				5		
3					6	
4						

Iteration 2

$s_5 \leftarrow goto(s_2, -)$

$s_6 \leftarrow goto(s_3, *)$

Example (*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

$s_7 \leftarrow goto(s_5, Expr)$

$s_8 \leftarrow goto(s_6, Term)$

Iteration also creates duplicate states 2, 3, and 4.

Filling in the ACTION and GOTO Tables

The algorithm

$\forall$ *set* $s_x \in S$

$\forall$ *item* $i \in s_x$

if i *is* $[A \rightarrow \beta \bullet \underline{a}d, \underline{b}]$ *and* $goto(s_x, \underline{a}) = s_k$, $\underline{a} \in T$

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

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

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

else if i *is* $[S' \rightarrow S \bullet, \text{EOF}]$

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

$\forall$ $n \in NT$

if $goto(s_x, n) = s_k$

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

x **is the state number**

Many items generate no table entry

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

Example (*Filling in the tables*)

The algorithm produces the following table

	ACTION				GOTO		
	<u>Ident</u>	-	*	EOF	*Expr*	*Term*	*Factor*
0	s 4				1	2	3
1				acc			
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,$\underline{a}$] — cannot do both actions
- This is a fundamental ambiguity, called a *shift/reduce error*
- Modify the grammar to eliminate it *(if-then-else)*
- Shifting will often resolve it correctly

EaC includes a worked example

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

- Each generates "reduce", but with a different production
- Both define ACTION[s,$\underline{a}$] — 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)

Shrinking the Tables

Three options:

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