CS412/CS413

Introduction to Compilers

Tim Teitelbaum

Lecture 10: LR Parsing

February 11, 2008

LR(0) Parsing Summary

- LR(0) item = a production with a dot in RHS
- LR(0) state = set of LR(0) items valid for a set of viable prefixes
- Compute LR(0) states and build DFA:
  - Start state: $V(\varepsilon) = \{ [S' \rightarrow .S] \}\downarrow^{*}$
  - Other states: $V(\alpha X) = V(\alpha)\rightarrow_{X}\downarrow^{*}$
- Build the LR(0) parsing table from the DFA
- Use the LR(0) parsing table to determine whether to reduce or to shift

LR(0) Limitations

- An LR(0) machine only works if each state with a reduce action has only one possible reduce action and no shift action
- With some grammars, construction gives states with shift/reduce or reduce/reduce conflicts
- Need to use look-ahead to choose

ok

[L → L,S.]

shift /reduce

[L → L,S.]
[S → S.,L]

reduce / reduce

[L → S,L .]
[L → S.]

LR(0) Parsing Table

	(	)	id	,	ε	S	L
1	s3		s2			g4	
2	S→id	S→id	S→id	S→id	S→id		
3	s3		s2			g7	g5
4					accept		
5		s6		s8			
6	S→(L)	S→(L)	S→(L)	S→(L)	S→(L)		
7	L→S	L→S	L→S	L→S	L→S		
8	s3		s2			g9	
9	L→L,S	L→L,S	L→L,S	L→L,S	L→L,S		

A Non-LR(0) Grammar

- Grammar for addition of numbers:

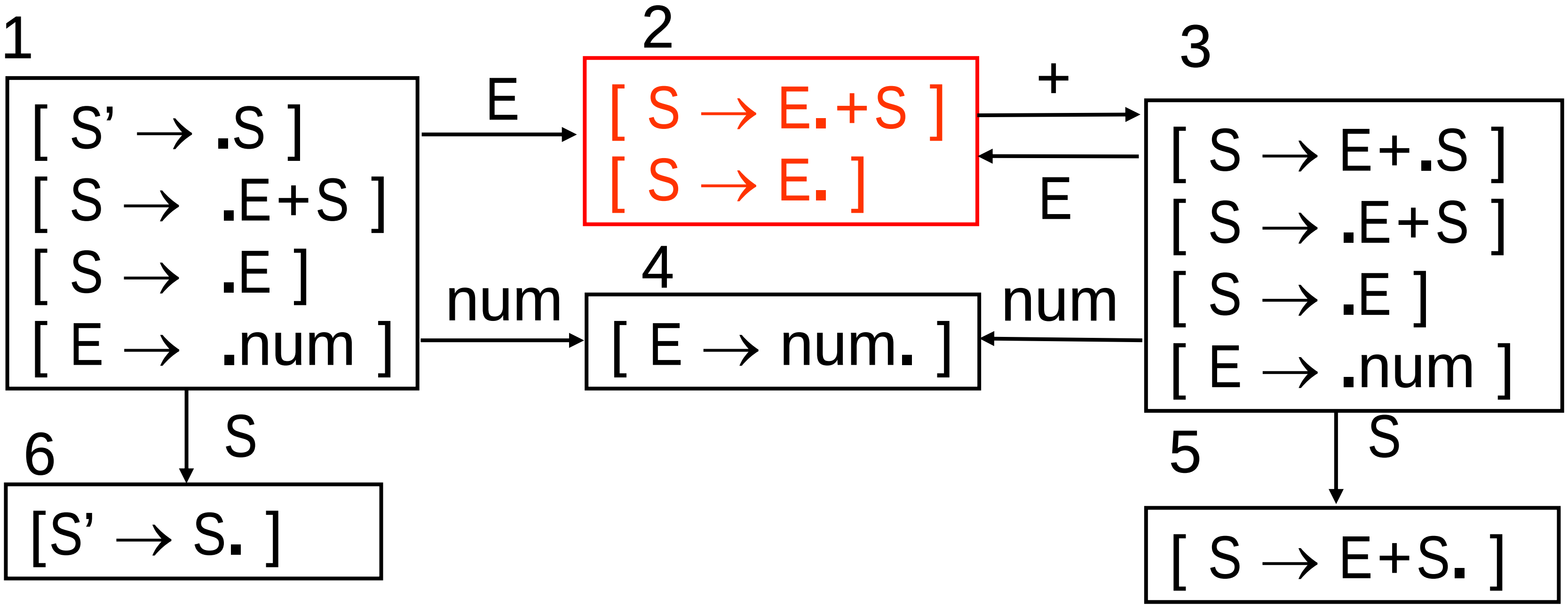
  S → S + E | E
  E → num

- Left-associative version is LR(0)

- Right-associative version is not LR(0)

  S → E + S | E
  E → num

LR(0) Parsing Table



What to do
in state 2:
shift or reduce?

	num	+	ε	E	S
1	s4			g2	g6
2	S→E	s3/S→E	S→E		

SLR(1) Parsing

- SLR Parsing = easy extension of LR(0)
  - For each reduction $A \rightarrow \beta$, look at the next symbol c
  - Apply reduction only if c is in FOLLOW(A), or c= ε and $S \Rightarrow^* \gamma A$
- SLR parsing table eliminates some conflicts
  - Same as LR(0) table except reduction rows
  - Adds reductions $A \rightarrow \beta$ only to columns of symbols in FOLLOW(A), or to column ε if $S \Rightarrow^* \gamma A$
- Example:

  FOLLOW(S)={}

  but $S \Rightarrow^* \gamma E$

	num	+	ε	E	S
1	s4			g2	g6
2		s3	$S \rightarrow E$		

SLR Parsing Table

- Reductions do not fill entire rows
- Otherwise, same as LR(0)

	num	+	ε	E	S
1	s4			g2	g6
2		s3	S→E		
3	s4			g2	g5
4			S→E		
5			S→E+S		
6			s7		
7			accept		

SLR(k)

- Use the LR(0) machine states as rows of table
- Let Q be a state and u be a lookahead string
  - Action(Q,u) = <u>shift Goto(Q,b)</u>

    if Q contains an item of the form $[A\rightarrow \beta_1 \mathbf{.} b\beta_3]$, with $u \in \text{FIRST}_k(b\beta_3 \text{FOLLOW}_k(A))$
  - Action(Q,u) = <u>accept</u>

    if Q = { $[S'\rightarrow S\mathbf{.}]$ } and $u=\varepsilon$
  - Action(Q,u) = <u>reduce</u> i

    if Q contains the item $[A\rightarrow \beta\mathbf{.}]$, where $A\rightarrow \beta$ is the i<u>th</u> production of G and $u \in \text{FOLLOW}_k(A)$, or $u=\varepsilon$ and $S \Rightarrow^* \gamma A$
  - Action(Q,u) = <u>error</u> otherwise
- G is SLR(k) iff the Action function given above is single-valued for all Q and u, i.e, there are no shift-reduce or reduce-reduce conflicts.

LR(1) Parsing

- Get as much power as possible out of 1 look-ahead symbol parsing table
- LR(1) grammar = recognizable by a shift/reduce parser with 1-symbol look-ahead
- LR(1) parsing uses concepts similar to LR(0)
  - Parser states = sets of items
  - LR(1) item = LR(0) item + look-ahead symbol following the production

LR(0) item : $\boxed{[\ S \rightarrow .S+E\ ]}$

LR(1) item : $\boxed{[\ S \rightarrow .S+E \quad +]}$

LR(1) States

- LR(1) state = set of LR(1) items
- LR(1) item = $[\ A \to \alpha.\beta \quad b]$, where b in $\Sigma \cup \{\varepsilon\}$
- Meaning: α already matched at top of the stack;
  next expect to see βb

- Shorthand notation

  $[\ A \to \alpha \,.\, B \quad b_1, \ldots, b_n]$

  means:

  $[\ A \to \alpha \,.\, \beta \quad b_1\ ]$

  ...

  $[\ A \to \alpha \,.\, \beta \quad b_n\ ]$

$[\ S \to S.{+}E \qquad +,\varepsilon]$

$[\ S \to S{+}.E \qquad \mathrm{num}]$

- Extend closure and goto operations

LR(1) Closure

- LR(1) closure operation on set of items S
  - For each item in S:

    $[A \rightarrow \alpha . B\beta \quad b]$

    and for each production $B \rightarrow \gamma$, add the following item to S:

    $[B \rightarrow .\gamma \quad \mathrm{FIRST}(\beta b)]$, or

    $[B \rightarrow .\gamma \quad \varepsilon]$ if $\mathrm{FIRST}(\beta b)=\{\}$
  - Repeat until nothing changes

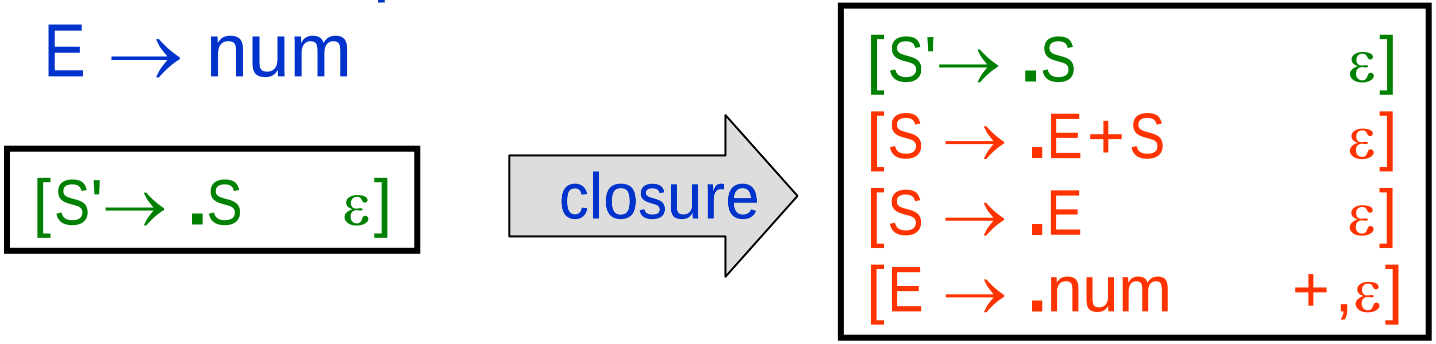
- Similar to LR(0) closure, but also keeps track of the look-ahead symbol

LR(1) Start State

- Initial state: start with [S’ → .S   ε], then apply the closure operation

- Example: sum grammar

  S’ → S
  S → E+S | E
  E → num



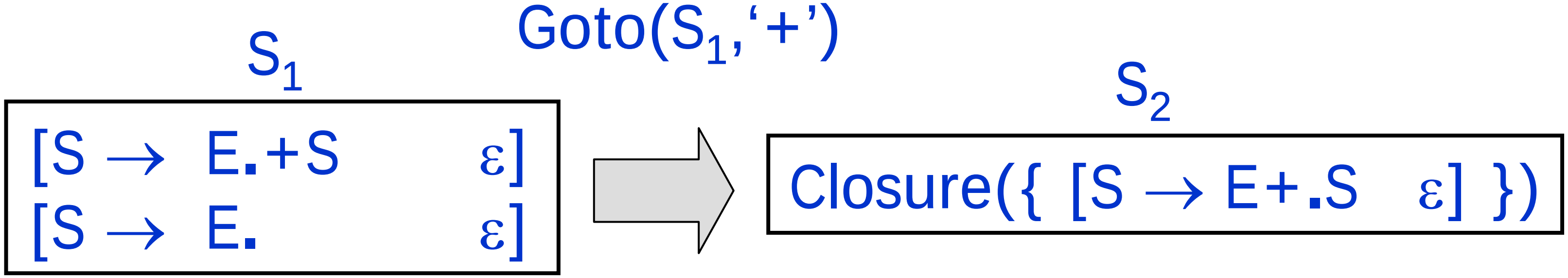
[S'→ .S   ε]

closure

[S'→ .S   ε]
[S → .E+S   ε]
[S → .E   ε]
[E → .num   +,ε]

LR(1) Goto Operation

- LR(1) goto operation = describes transitions between LR(1) states
- Algorithm: for a state S and a symbol Y
  - $S' = \{\ [A \to \alpha Y.\beta \quad b] \mid [A \to \alpha.Y\beta \quad b] \in S\}$
  - Goto(S, Y) = Closure(S')



S_1

$[S \to E.+S \quad \varepsilon]$

$[S \to E. \quad \varepsilon]$

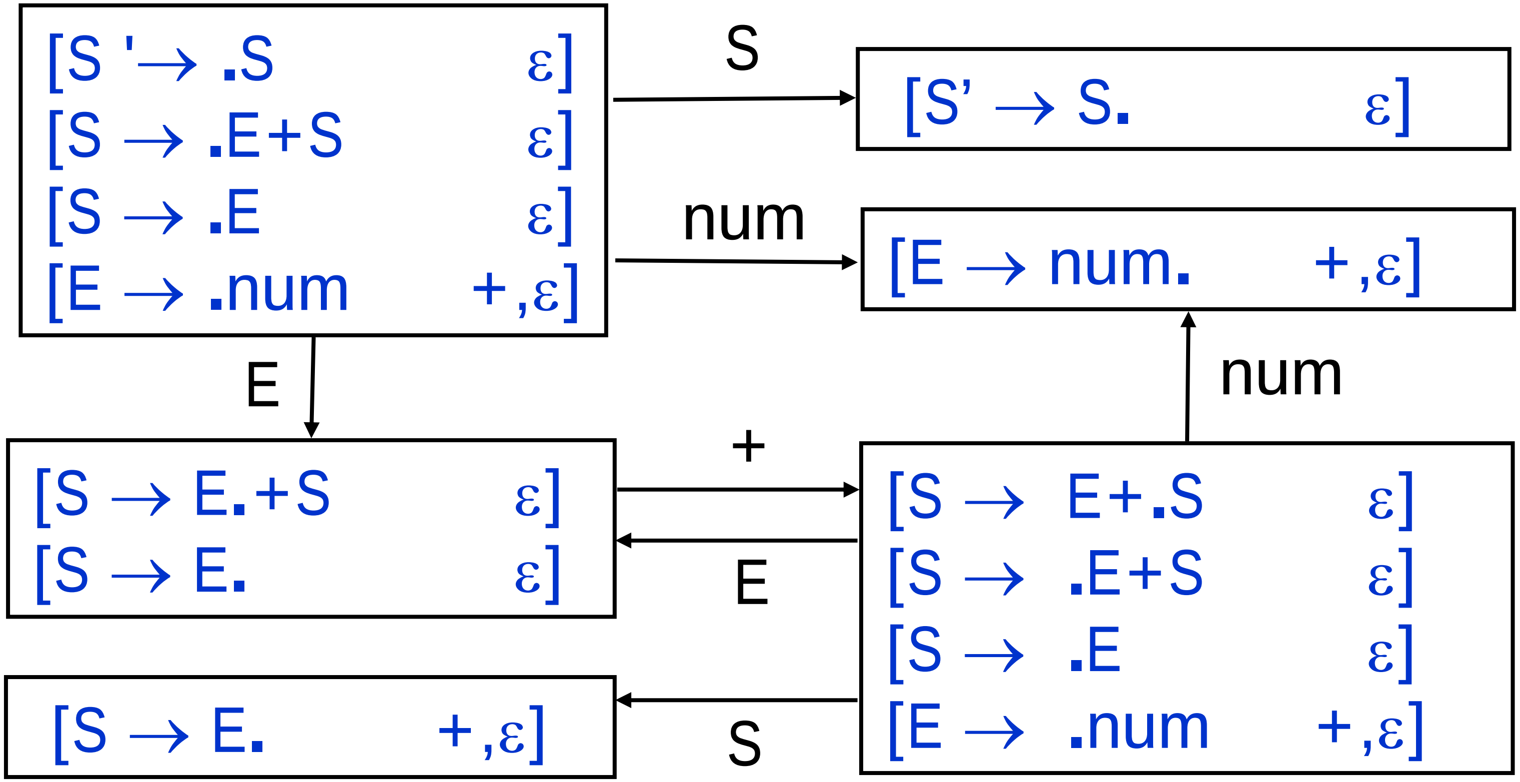
Goto(S_1,'+')

S_2

Closure({ $[S \to E+.S \quad \varepsilon]$ })

LR(1) DFA Construction

- If S’ = Goto(S,X) then add an edge labeled X from S to S’



[S '→ .S ε]
[S → .E+S ε]
[S → .E ε]
[E → .num +,ε]

[S’ → S. ε]

[E → num. +,ε]

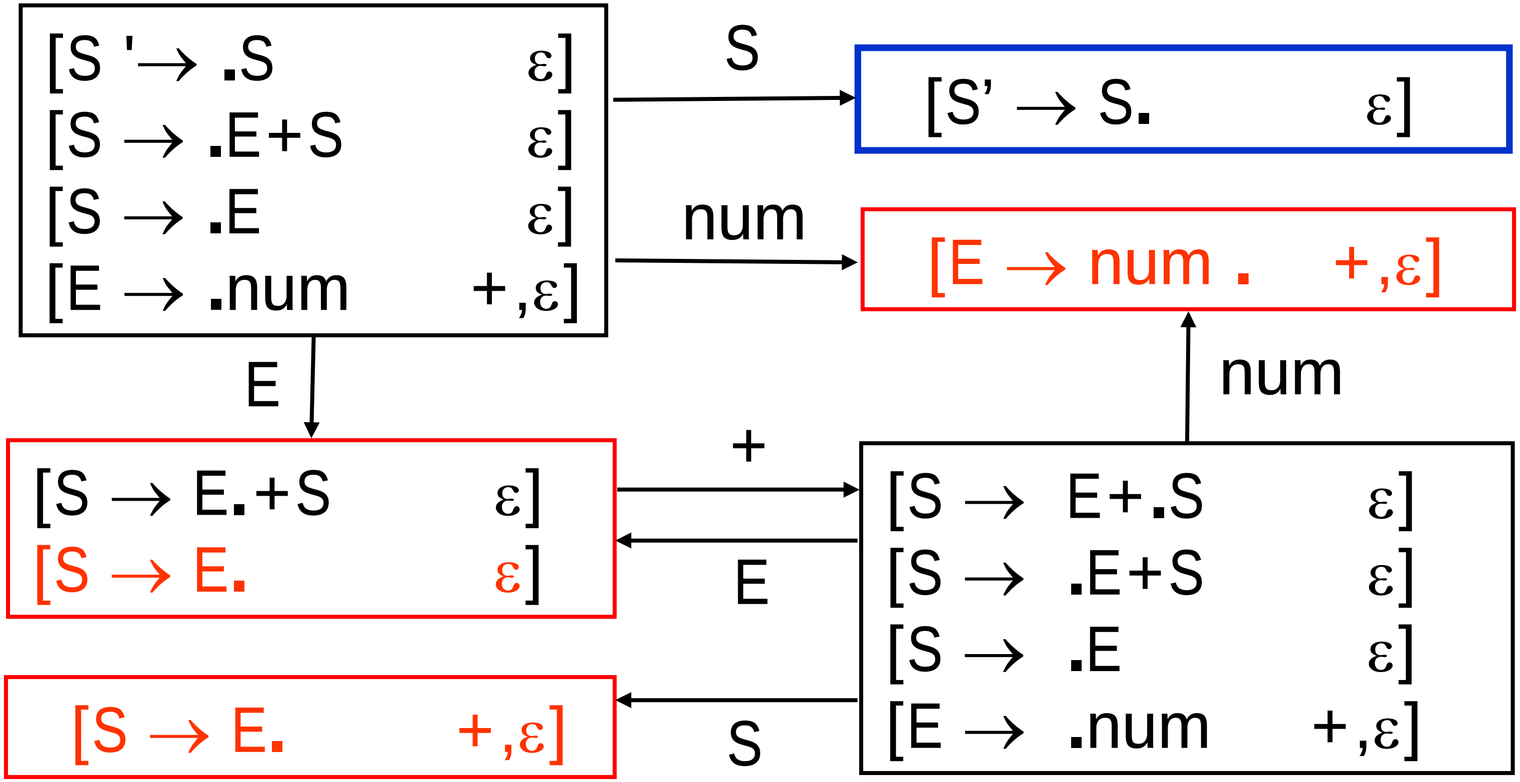
[S → E.+S ε]
[S → E. ε]

[S → E+.S ε]
[S → .E+S ε]
[S → .E ε]
[E → .num +,ε]

[S → E. +,ε]

LR(1) Reductions

- Reductions correspond to LR(1) items of the form [A → β. x]



[S '→ .S ε]
[S → .E+S ε]
[S → .E ε]
[E → .num +,ε]

S

[S' → S. ε]

num

[E → num . +,ε]

E

[S → E.+S ε]
[S → E. ε]

+

E

num

[S → E+.S ε]
[S → .E+S ε]
[S → .E ε]
[E → .num +,ε]

S

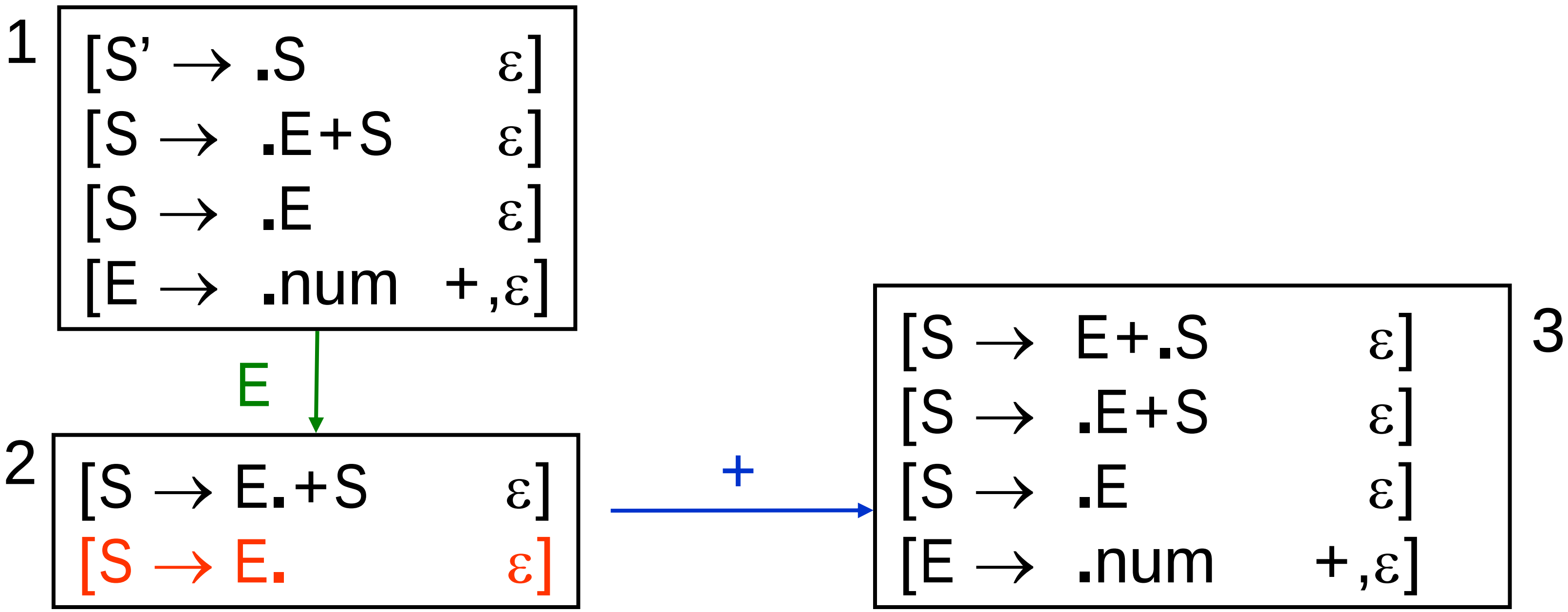
[S → E. +,ε]

LR(1) Parsing Table Construction

- Same as construction of LR(0) parsing table, except for reductions
- If $[A \rightarrow \beta.\quad b] \in$ state Q, then:

  Action(Q,b) is Reduce($A \rightarrow \beta$)

LR(1) Parsing Table Example



Fragment of the Parsing table:

	+	ε	E
1			2
2	s3	S→E	

LR(1) but not SLR(1)

- Let G have productions

  $S \rightarrow aAb \mid Ac$

  $A \rightarrow a \mid \varepsilon$

- V(a) = {

  [$S \rightarrow a.Ab$]
  [$A \rightarrow a.$]
  [$A \rightarrow .a$]
  [$A \rightarrow .$]

  }

FOLLOW(A) = {b,c}

reduce-reduce conflict

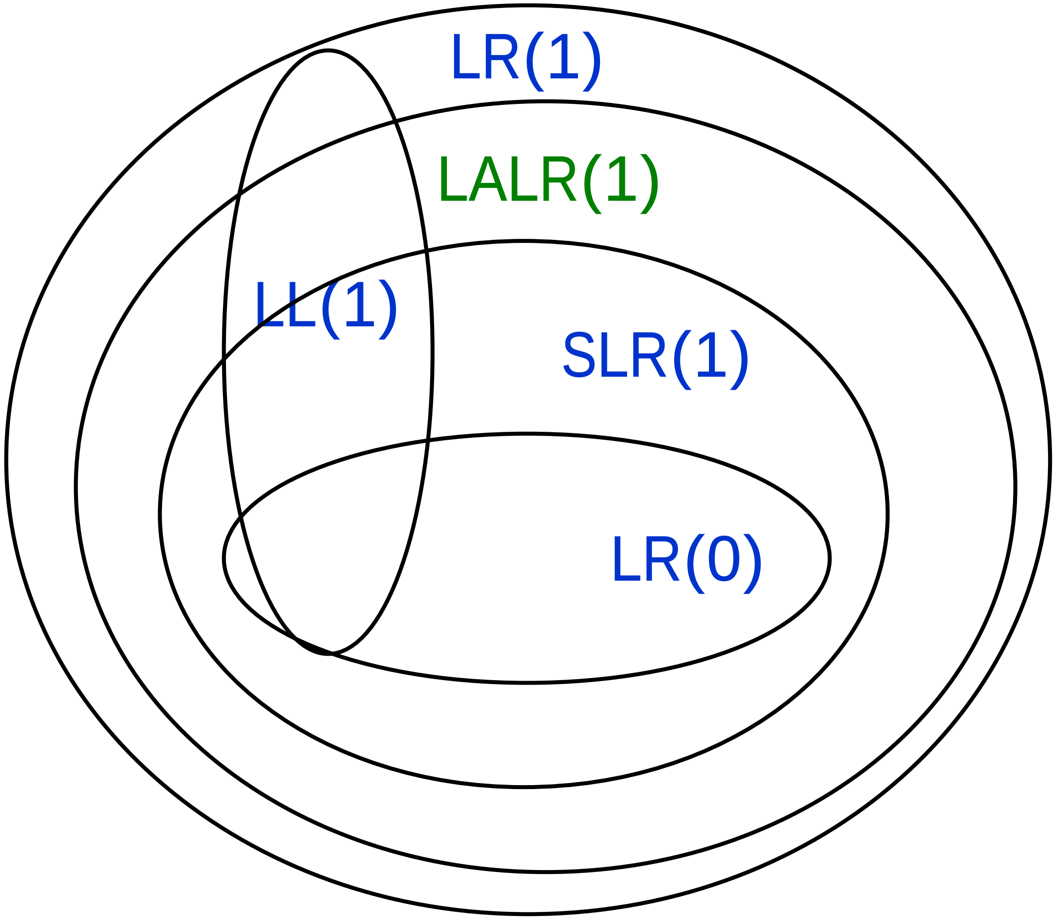
LALR(1) Grammars

- Problem with LR(1): too many states
- LALR(1) Parsing (Look-Ahead LR)
  - Construct LR(1) DFA and then merge any two LR(1) states whose items are identical except look-ahead
  - Results in smaller parser tables
  - Theoretically less powerful than LR(1)

$$\boxed{\begin{array}{ll} [S \rightarrow \text{id}\,\mathbf{.} & +] \\ [S \rightarrow E\,\mathbf{.} & \varepsilon] \end{array}} + \boxed{\begin{array}{ll} [S \rightarrow \text{id}\,\mathbf{.} & \varepsilon] \\ [S \rightarrow E\,\mathbf{.} & +] \end{array}} = \ ?$$

- LALR(1) Grammar = a grammar whose LALR(1) parsing table has no conflicts

Classification of Grammars



$\mathrm{LR}(k) \subseteq \mathrm{LR}(k+1)$
$\mathrm{LL}(k) \subseteq \mathrm{LL}(k+1)$
$\mathrm{LL}(k) \subseteq \mathrm{LR}(k)$

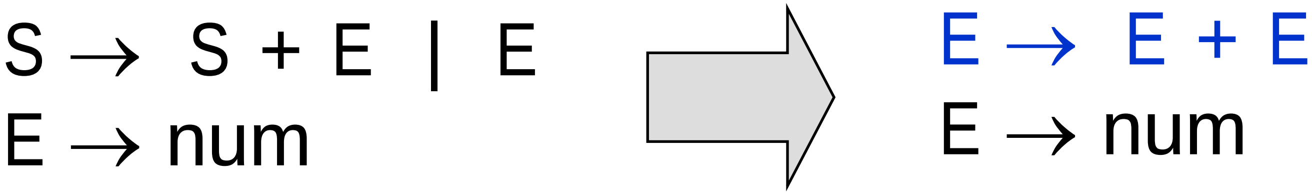
$\mathrm{LR}(0) \subseteq \mathrm{SLR}(1)$

$\mathrm{LALR}(1) \subseteq \mathrm{LR}(1)$

Automate the Parsing Process

- Can automate:
  - The construction of LR parsing tables
  - The construction of shift-reduce parsers based on these parsing tables
- Automatic parser generators: yacc, bison, CUP
- LALR(1) parser generators
  - Not much difference compared to LR(1) in practice
  - Smaller parsing tables than LR(1)
  - Augment LALR(1) grammar specification with declarations of precedence, associativity
- output: LALR(1) parser program

Associativity



S → S + E | E
E → num

E → E + E
E → num

What happens if we run this grammar through LALR construction?

Shift/Reduce Conflict

E → E + E
E → num

[E → E+E.   +]
[E → E.+E   +,ε]

$\xrightarrow{+}$

shift/reduce conflict

shift: 1+(2+3)
reduce: (1+2)+3

```
1+2+3
   ^
```

Grammar in CUP

```
nonterminal E; terminal PLUS, LPAREN...
precedence left PLUS;
```

“when shifting a ‘+’ conflicts with reducing a production, choose reduce”

```
E ::= E  PLUS  E
    |   LPAREN  E  RPAREN
    |   NUMBER ;
```

Precedence

- CUP can also handle operator precedence

$$
\begin{aligned}
E &\to E + E \;\mid\; T \\
T &\to T \times T \;\mid\; \mathrm{num} \;\mid\; (\,E\,)
\end{aligned}
$$

⇓

$$
\begin{aligned}
E \to\;& E + E \;\mid\; E \times E \\
\mid\;& \mathrm{num} \;\mid\; (\,E\,)
\end{aligned}
$$

Conflicts without Precedence

$$E \rightarrow E + E \;\mid\; E \times E$$

$$\mid\; \text{num} \;\mid\; (E)$$

[E → E.+E ...]

[E → E×E. +]

[E → E+E. ×]

[E → E.×E ...]

Predecence in CUP

precedence left PLUS;
precedence left TIMES; // TIMES > PLUS
E ::= E PLUS E | E TIMES E | …

RULE: in conflict, choose reduce if last terminal of production has higher precedence than symbol to be shifted; choose shift if vice-versa. In tie, use associativity (left or right) given by precedence rule

[E → E .+E …]
[E → E×E. +]

reduce E→E×E

[E → E+E. ×]
[E → E.×E …]

Shift ×

Summary

- Look-ahead information makes SLR(1), LALR(1), LR(1) grammars expressive
- Automatic parser generators support LALR(1) grammars
- Precedence, associativity declarations simplify grammar writing