LR Parsing

Last time, Earley parser:
- Bottom-up parser
- Works for ambiguous CFGs; generates all parses
- Suitable for natural languages
- Overkill for PLs—only one parse tree needed

Also last time, a brief look at LR parser, which makes shift/reduce choices

shift ≈ scan + take prediction closure

reduce $X \to \gamma$ ≈ complete X + - - -

LR(k) parser restricts grammar so that
only one choice is possible given k lookaheads current parser state

LR(0) parser

makes shift/reduce choices based solely on parser state

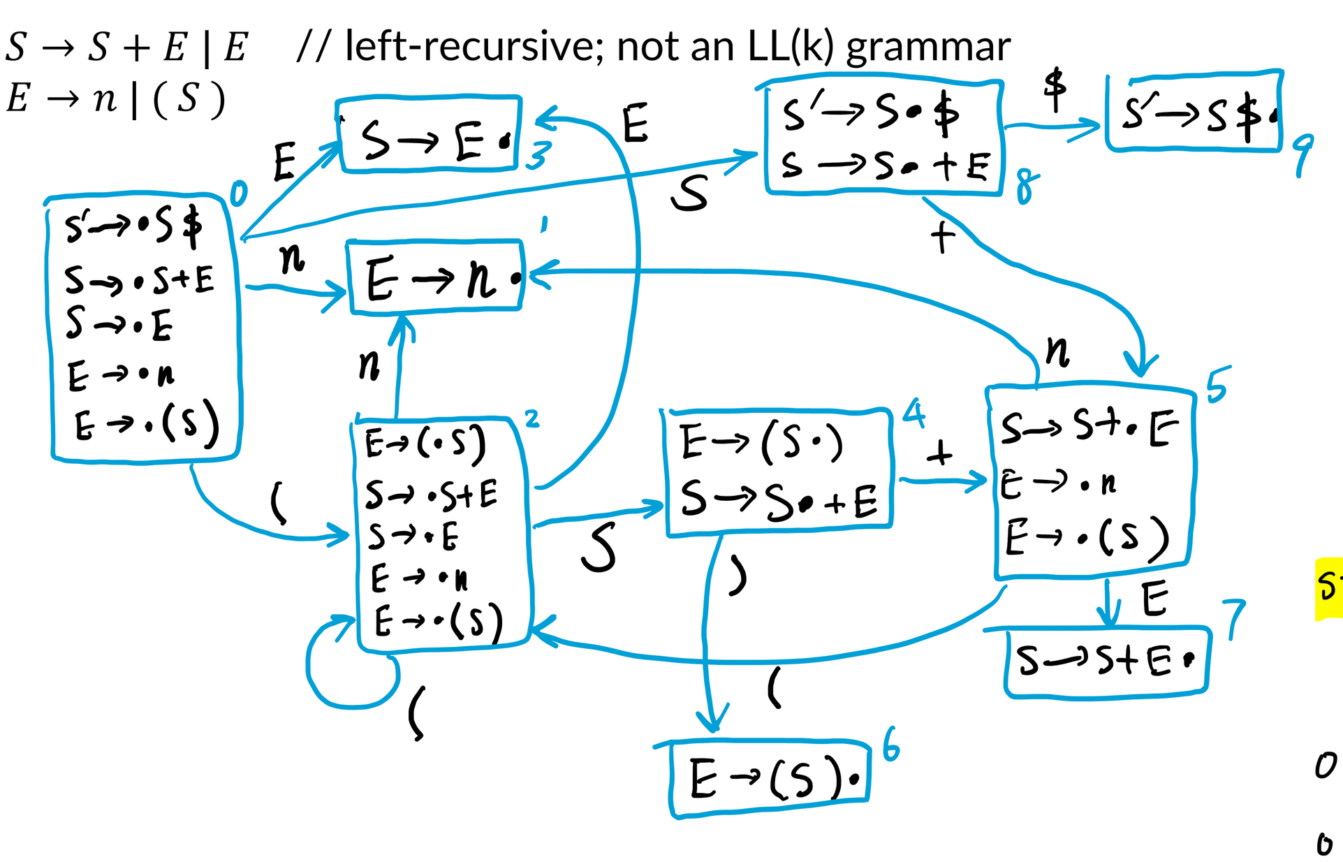
Automaton state : set of LR(0) items of form $[X \to \alpha \cdot \beta]$

Constructing automaton for LR(0) grammar

1. Initial automaton state := closure of item $[S' \to \cdot S \$]$
2. Do until no more changes are possible :
   For each symbol to right of • in some item :
   Add transition to a state w/ all matching items advanced and closure taken

Example

$S \to S + E \mid E$ // left-recursive; not an LL(k) grammar
$E \to n \mid (S)$



action table | goto table

	n	+	(	)	$	S	E
0	1		2			8	3
1	---		E→n	~			
⋮							
9							

Parsing LR(0) grammar

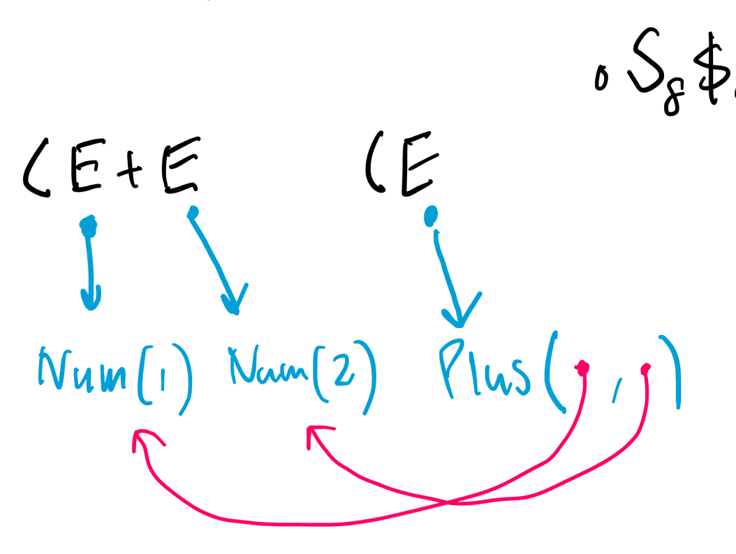
Example

input = (1 + 2)

stack	unconsumed input	next action
0	(1+2) $	shift 2
0 (2	1+2) $	shift 1
0 (2 1₁	+2) $	reduce E→n
0 (2 E	+2) $	goto 3
0 (2 E3	+2) $	reduce S→E
0 (2 S	+2) $	goto 4
0 (2 S4	+2) $	shift 5
0 (2 S4 +5	2) $	
⋮		
0 S8 $9		

```
expr ::= expr:e1 PLUS expr:e2 {:
    RESULT = new Plus(e1, e2);
:}
```

(1+2•) * 3

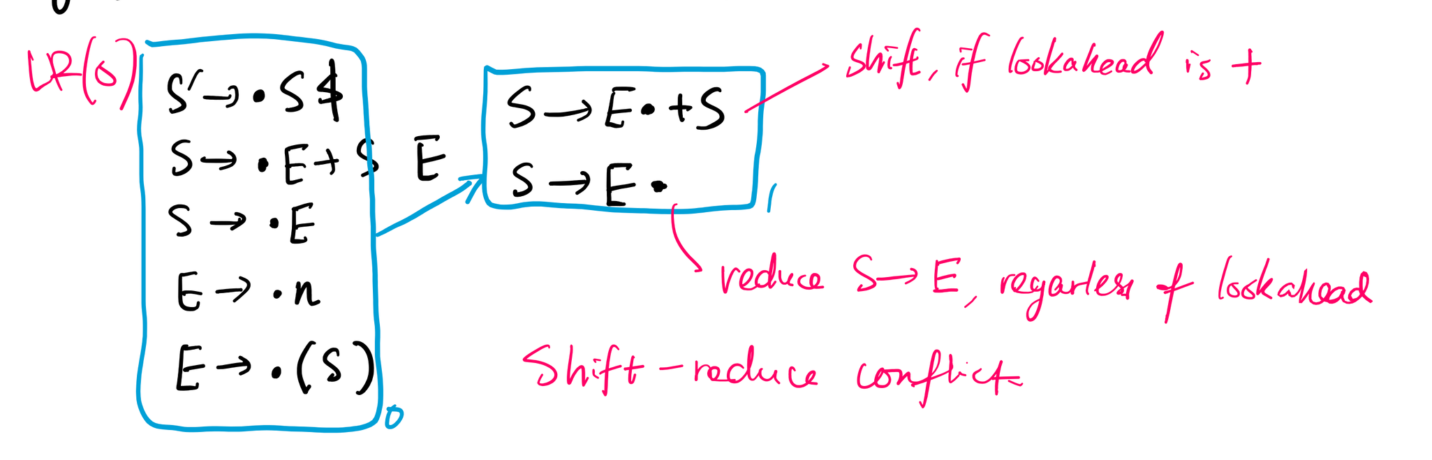


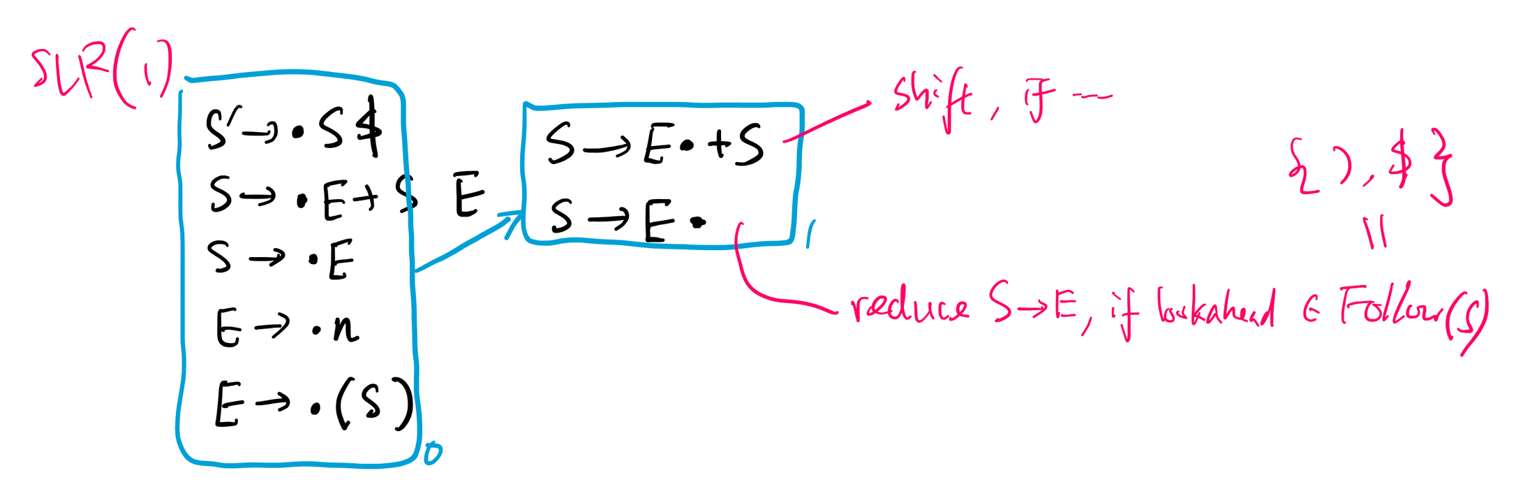
SLR(1)

simple

reduce $X \to \gamma$ only if lookahead in Follow (X)

$S \to E + S \mid E$
$E \to n \mid (S)$





LR(1)

LR(1) item = LR(0) item + 1-lookahead set $[X \to \beta \cdot \gamma, \lambda]$

tokens that can follow γ

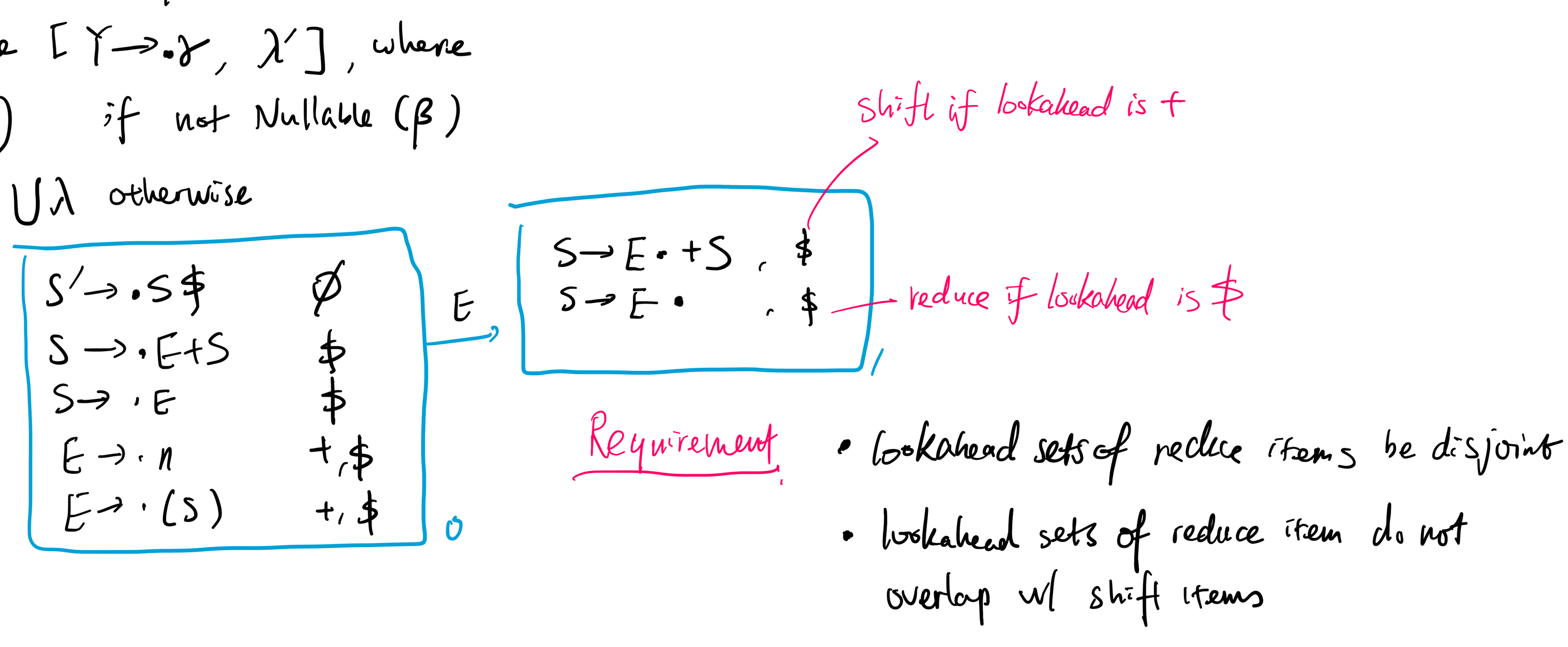
Taking LR(1) closure

If state includes $[X \to \alpha \cdot Y \beta, \lambda]$,
the state should include $[Y \to \cdot \gamma, \lambda']$, where

$\lambda' := First(\beta)$ if not Nullable (β)

$\lambda' := First(\beta) \cup \lambda$ otherwise

$S \to E + S \mid E$
$E \to n \mid (S)$



Requirement
- lookahead sets of reduce items be disjoint
- lookahead sets of reduce item do not overlap w/ shift items

Fixing conflicts using precedence declarations in parser generator

Example $E \to E + E \mid E * E$

E → E+E •, +	reduce on +
E → E • + E, ~	shift on +

```
precedence left PLUS;
---- -- - TIMES;
```

1 * 2• + 3

1 + 2• * 3

LALR(1)

Problem with LR(1): number of automaton states can be large

Relative power of parser technologies