LR Parsing pt. 3: Canonical LR(1) Parsers

CS 440: Programming Languages and Translators, Spring 2020

1. LR(1) Parsers

- LR(1) parsers are also known as "full" or "canonical" LR(1) parsers. The abbreviation CLR(1) means "canonical LR(1) parser", but it's just a synonym for "LR(1) parser".
- Recall that LR(1) parsers use LR(1) items, which have the form $A \to \alpha \bullet \beta$, L, where $L \subseteq$ Follow(A) is the *lookahead set* for this item. If $\beta \equiv \varepsilon$, then we reduce $A \to \alpha \bullet$, L iff next input symbol $\in L$.
- **Splitting the LR(0) states**: The number of possible lookahead sets L for ($A \to \alpha \bullet \beta$, L) depends on the number of times A appears in the rhs of any of the rules. The size of the set L depends on the size of the local follow set for A.
  - E.g., the rule $B \to A$ `a` A `b` A `c` A `d` A `e`, generates five different lookahead sets of one symbol each. The rule $B \to A\ Digit$ generates a lookahead set of size 10 (assuming *Digit* means the decimal digits).
- **Calculating a lookahead set**: Given an LR(1) item ($B \to \gamma \bullet A\ \delta$, L_B) we can calculate the lookahead set L for $A \to \bullet\ \alpha\ \beta$, L as (1) If $\delta \not\to^* \varepsilon$, then L = First(δ). (2) If $\delta \to^* \varepsilon$, then L = (First(δ) - ε) $\cup L_B$.
  - The justification for this is that the A in ($B \to \gamma \bullet A\ \delta$, L_B) can be followed by any symbol that begins a δ. In addition, if δ can generate ε, then in effect we're looking at ($B \to \gamma \bullet A$, L_B), so in that case, A can be followed by anything that can follow the B, which is exactly what L_B means.
  - A shorthand that combines the two cases is L = First($\delta\ L_B$), where for $L_B = \{x_1, x_2, \ldots x_n\}$ we use $(x_1 \mid x_2 \mid \ldots \mid x_n)$.

2. Example 1: LR(1) Parser that finds errors before reducing

- Let's go back to the grammar with rules 0: $S' \to S$ `$`, 1: $S \to A$ `b` A `c`, 2: $A \to$ `a`.
- (This was the grammar where an SLR(1) parser does parse the language, but it delays announcing parse errors for inputs like `a c` … or `a b a b`.)
  - The SLR(1) parser behaves as if we had LR(1) items 2a: $A \to \bullet$ `a`, {`b`, `c`}, and 2a: $A \to$ `a` $\bullet$, {`b`, `c`}.
  - For the LR(1) parser, since A appears twice in $S \to A$ `b` A `c`, with different local follow sets, we get two versions of the rule 2a items; I'll call them 2a.1: $A \to \bullet$ `a`, {`b`} and 2a.2: $A \to \bullet$ `a`, {`c`}. Shifting an `a` takes us to 2b.1: $A \to$ `a` $\bullet$, {`b`} and 2b.2: $A \to$ `a` $\bullet$, {`c`}. Both items indicate possible reductions, limited now by the lookhead sets.
- Because Follow(S) = {`$`}, the rule 1 items all have `$` as their lookahead symbol:
  - 1a: $S \to \bullet A$ `b` A `c`, {`$`}, 1b: $S \to A \bullet$ `b` A `c`, {`$`}, …, 1d: $S \to A$ `b` $\bullet A$ `c`, {`$`}, 1e: $S \to A$ `b` A `c` $\bullet$, {`$`}
  - Since 1a has $\bullet A$ followed by `b`, the deterministic state that includes 1a also has to include 2a.1: $A \to \bullet$ `a`, {`b`}; similarly, a state with item 1c: $S \to A$ `b` $\bullet A$ `c`, {`$`} has to include 2a.2: $A \to \bullet$ `a`, {`c`}.

- The shift actions in the Action/Go-To tables for the LR(1) grammar are the same as in the SLR(1) parser

State #	State Items	Actions				GoTo	
		a	b	c	$	*S*	*A*
0	{0a: $S' \to \bullet\ S$ $, ∅, 1a: $S \to \bullet\ A$ b A c, {$} 2a.1: $A \to \bullet$ a, {b}}	s6: {2b.1}				1: 0b	2: 1b
1	{0b: $S' \to S \bullet$ $, ∅}				accept		
2	{1b: $S \to A \bullet$ b A c, {$}		s3: {1c}				
3	{1c: $S \to A$ b $\bullet\ A$ c, {$}, 2a.2: $A \to \bullet$ a, {c}}	s7: {2b.2}					4: 1d
4	{1d: $S \to A$ b $A \bullet$ c, {$}			s5: {1e}			
5	{1e: $S \to A$ b A c $\bullet$, {$}				r1		
6	{2b.1: : $A \to$ a •, {b}}		r2				
7	{2b.2: : $A \to$ a •, {c}}			r2			

LR(1) Parser for S → A b A c, A → c

- Here are some execution traces using the full LR(1) parser. In the unsuccessful parses, an LR(1) parser announces an error because it can't reduce $A \to$ a $\bullet$ because the next symbol isn't in the lookahead set.

LR(1) Parse of a b a c $

Stack (top at right)	Input	Action
0	a b a c $	s6
0 a 6	b a c $	r2
0 *A* 2	b a c $	s3
0 *A* 3 b 3	a c $	s7
0 *A* 3 b 3 a 7	c $	r2
0 *A* 3 b 3 *A* 4	c $	s5
0 *A* 3 b 3 *A* 5 c 5	$	r1
0 *S* 1	$	accept

Unsuccessful Parse of a c $

Stack (top at right)	Input	Action
0	a c $	s6
0 a 6	c $	error*

Unsuccessful Parse of a b a b $

Stack (top at right)	Input	Action
0	a b a b $	s6
0 a 6	b a b $	r2
0 *A* 2	b a b $	s3
0 *A* 3 b 3	a b $	s7
0 *A* 3 b 3 a 7	b $	error*

- Recall the SLR(1) parser combines the reductions in states 7 and 8 into one reduction for one state. On the inputs a c $ and a b a b $, the SLR(1) parser reduces and announces an error at the next shift operation.

State #	State Items	Actions				GoTo	
		a	b	c	$	*S*	*A*
…	…						
…	{2b:: $A \to$ a •}		r2	r2			

SLR(1) Parser for S → A b A c, A → c

3. *Handling ε-rules*

- An ε-rule like $A \to \varepsilon$ has LR(0) items $A \to \bullet\ \varepsilon$ and $A \to \varepsilon\ \bullet$, but these items indicate the same situation, since we can always choose to reduce an ε-rule.
  - LR(0) parsers: These can't handle ε-rules well because we can always reduce $A \to \varepsilon\ \bullet$, so all states including an ε-rule item have shift-reduce or reduce-reduce conflicts. One can try arbitrating a reduce-reduce conflict by doing the non-ε reduction (assuming there is one).
  - SLR(1) parsers: These handle ε-rules by checking the next input symbol and reducing $A \to \varepsilon\ \bullet$ if the symbol is ∈ Follow(A).
    - This doesn't resolve conflicts in states that include $A \to \varepsilon\ \bullet$ and (e.g.) $B \to \alpha \bullet \beta$ or $C \to \gamma\ \bullet$ if the next symbol ∈ Follow(A) and also First(β) or Follow(C) (or, if $\beta \to^* \varepsilon$ is possible, Follow(B)).
  - Full LR(1): These handle ε-rules by checking the next input symbol and reduce $A \to \varepsilon\ \bullet, L$ if the next symbol ∈ L. (So this is like the SLR(1) rule but checks the local follow set of A, not the global one.)
    - As with SLR(1), this doesn't resolve the conflict between $A \to \varepsilon\ \bullet, L$ and $B \to \alpha \bullet \beta, L_B$ or $C \to \gamma\ \bullet, L_C$ if the next symbol ∈ L and also First(β) or Follow(C) (or, if $\beta \to^* \varepsilon$ is possible, L_B.)

Example 2: Add ε-rule to earlier grammar

- **Example 2**: Say we add an ε rule 3: $A \to \varepsilon$ to 0: $S' \to S$ `$`, 1: $S \to A$ `b` A `c`, 2: $A \to$ `a`.
  - In that case, we add $A \to \varepsilon\ \bullet$, {`b`} to the state containing $S \to \bullet\ A$ `b` A `c`, {`$`}; there's no conflict because First(`b` A `c`) ∩ {`$`} = {`b`} ∩ {`$`} = ∅. Similarly, we add $A \to \varepsilon\ \bullet$, {`c`} to the state containing $S \to A$ `b` $\bullet$ A `c`, {`$`} with no conflict because First(`c`) ∩ {`$`} = {`c`} ∩ {`$`} = ∅.
  - The resulting Action and Go-To tables get only a few changes, shown in dark red below:

State #	State Items	Actions				GoTo	
		a	b	c	$	S	A
0	{0a: $S' \to \bullet\ S$ `$`, ∅, 1a: $S \to \bullet\ A$ `b` A `c`, {$} 2a.1: $A \to \bullet$ `a`, {b}, 3b: $A \to \varepsilon\ \bullet$, {`b`}}	s6: {2b.1}	r3			1: 0b	2: 1b
1	{0b: $S' \to S \bullet$ `$`, ∅}				accept		
2	{1b: $S \to A \bullet$ `b` A `c`, {$}}		s3: {1c}				
3	{1c: $S \to A$ `b` $\bullet\ A$ `c`, {$}, 2a.2: $A \to \bullet$ `a`, {`c`}, 3b: $A \to \varepsilon\ \bullet$, {`c`}}	s7: {2b.2}		r3			4: 1d
4	{1d: $S \to A$ `b` $A \bullet$ `c`, {$}}			s5: {1e}			
5	{1e: $S \to A$ `b` A `c` $\bullet$, {$}}				r1		
6	{2b.1: : $A \to$ `a` •, {b}}		r2				
7	{2b.2: : $A \to$ `a` •, {c}}			r2			

LR(1) Parser for $S \to A$ `b` A `c`, $A \to$ `c`, $A \to \varepsilon$

- The SLR(1) parser can also handle $A \to \varepsilon$, but it reduces $A \to \varepsilon\ \bullet$ if the next symbol is b or c, since we're checking the full Follow(A).

State #	State Items	Actions				GoTo	
		a	b	c	$	S	A
0	{0a: $S' \to \bullet\, S$ \$, 1a: $S \to \bullet\, A$ b A c} 2a: $A \to \bullet$ a}, 3b: $A \to \varepsilon\, \bullet$}	s6: {2b}	r3	r3		1: 0b	2: 1b
1	{0b: $S' \to S \bullet$ \$, ∅}				accept		
2	{1b: $S \to A \bullet$ b A c		s3: {1c}				
3	{1c: $S \to A$ b $\bullet\, A$ c, 2a: $A \to \bullet$ a}, 3b: $A \to \varepsilon\, \bullet$}	s6: {2b}	r3	r3			4: 1d
4	{1d: $S \to A$ b $A \bullet$ c			s5: {1e}			
5	{1e: $S \to A$ b A c $\bullet$				r1		
6	{2b: : $A \to$ a $\bullet$}		r2	r2			

SLR(1) Parser for $S \to A$ b A c, $A \to$ c, $A \to \varepsilon$

Example 3: Grammar with multiple ε-rules

- Let's study the grammar with rules 0: $S' \to S$ \$, 1: $S \to D\, E$, 2: $D \to$ d, 3: $D \to \varepsilon$, 4: $E \to$ e, 5: $E \to \varepsilon$.
- This is more complicated because D and E both can generate ε, plus they form a sequence at the end of a rule.
- The LR(1) Items:
  - 0a: $S' \to \bullet\, S$ \$, ∅, 0b: $S' \to S \bullet$ \$, ∅, 0c: $S' \to S$ \$ $\bullet$, ∅
  - 1a: $S \to \bullet\, D\, E$, {\$}, 1b: $S \to D \bullet E$, {\$}, 1c: $S \to D\, E\, \bullet$, {\$}
  - 2a: $D \to \bullet$ d, {e, \$}, 2b: $D \to$ d $\bullet$, {e, \$},
  - 3b: $D \to \varepsilon\, \bullet$, {e, \$}
  - 4a: $E \to \bullet$ e, {\$}, 4b: $E \to$ e $\bullet$, {\$}
  - 5b: $E \to \varepsilon\, \bullet$, {\$}

State #	Items	Actions			GoTo		
		d	e	$	S	D	E
0	{0a: $S' \to \bullet\, S$ \$, ∅, 1a: $S \to \bullet\, D\, E$, {\$}, 2a: $D \to \bullet$ d, {e, \$}, 3b: $D \to \varepsilon\, \bullet$, {e, \$}}	2b	r3	r3	0b	1b	
1	{0b: $S' \to S \bullet$ \$, ∅}			accept			
2	{1b: $S \to D \bullet E$, {\$}, 4a: $E \to \bullet$ e, {\$}, 5b: $E \to \varepsilon\, \bullet$, {\$}}		4b	r5			1c
3	{1c: $S \to D\, E\, \bullet$, {\$}}			r1			
4	{2b: $D \to$ d $\bullet$, {e, \$}		r2	r2			
5	{4b: $E \to$ e $\bullet$, {\$}}			r4			

LR(1) Parser for 0: $S' \to S$ \$, 1: $S \to D\, E$, 2: $D \to$ d, 3: $D \to \varepsilon$, 4: $E \to$ e, 5: $E \to \varepsilon$

Activity Problems For Lecture 19

Lecture 19: LR(1) Parsers

1. Study the LR(1) Action/Go-To table from Example 2 for the grammar $S \to A$ `b` A `c`, $A \to$ `c`, $A \to \varepsilon$. Use it to trace the execution of the parser on inputs `a b c $` and `b a c $`.

2. Study the LR(1) Action/Go-To table from Example 3 for the grammar 0: $S' \to S$ `$`, 1: $S \to D\ E$, 2: $D \to$ `d`, 3: $D \to \varepsilon$, 4: $E \to$ `e`, 5: $E \to \varepsilon$. Use it to trace the execution of the parser on inputs `d e $`, `d $`, `e $`, and `$`.

3. Study the grammar with rules 0: $S' \to S$ `$`, 1: $S \to D$, 2: $D \to$ `d` E, 3: $D \to E$, 4: $E \to$ `e`, 5: $E \to \varepsilon$. (This is similar to the grammar in Problem 2 but adds recursion.)
   a. What are the LR(1) items for this grammar?
   b. Show the LR(1) Action/Go-to Table for this grammar.
   c. Trace the execution of the parser on inputs `d d e e $`, `d d $`, `e e $`, and `$`.

Solutions to Activity Problems for Lecture 19

1. (Trace grammar with rules $S \to A$ `b` A `c`, $A \to$ `c`, $A \to \varepsilon$.) Notice that the parses are similar to each other and to a parse of `a b a c $`. The differences come in when reducing $A \to \varepsilon$ versus shifting `a` and then reducing $A \to$ `a`. Either way, after the reduction, we end up in state 2 or 4, depending on where we are in the rule.

LR(1) Parse of `a b c $`

Stack (top at right)	Input	Action
0	a b c $	s6
0 a 6	b c $	r2
0 *A* 2	b c $	s3
0 *A* 3 b 3	c $	r3
0 *A* 3 b 3 *A* 4	c $	s5
0 *A* 3 b 3 *A* 4 c 5	$	r1
0 *S* 1	$	accept

LR(1) Parse of `b a c $`

Stack (top at right)	Input	Action
0	b a c $	r3
0 *A* 2	b a c $	s3
0 *A* 3 b 3	a c $	s7
0 *A* 3 b 3 a 7	c $	r2
0 *A* 3 b 3 *A* 4	c $	s5
0 *A* 3 b 3 *A* 4 c 5	$	r1
0 *S* 1	$	accept

2. (Trace grammar with rules 0: $S' \to S$ `$`, 1: $S \to D$, 2: $D \to$ `d` E, 3: $D \to E$, 4: $E \to$ `e`, 5: $E \to \varepsilon$.)

LR(1) Parse of `d e $`

Stack (top at right)	Input	Action
0	d e $	s4
0 d 4	e $	r2
0 *D* 2	$	r5
0 *D* 2 *E* 3	$	r1
0 *S* 1	$	accept

LR(1) Parse of `d $`

Stack (top at right)	Input	Action
0	d $	s4
0 d 4	$	r2
0 *D* 2	$	r5
0 *D* 2 *E* 3	$	r1
0 *S* 1	$	accept

LR(1) Parse of `e $`

Stack (top at right)	Input	Action
0	e $	r3
0 *D* 2	e $	s5
0 *D* 2 e 5	$	r4
0 *D* 2 *E* 3	$	r1
0 *S* 1	$	accept

LR(1) Parse of `$`

Stack (top at right)	Input	Action
0	$	r3
0 *D* 2	$	r5
0 *D* 2 *E* 3	$	r1
0 *S* 1	$	accept

3. Study the grammar with rules 0: $S' \to S$ \$, 1: $S \to D\ E$, 2: $D \to$ d D, 3: $D \to \varepsilon$, 4: $E \to$ e E, 5: $E \to \varepsilon$. (This is similar to the grammar in Example 3.)

a. (LR(1) items)

- 0a: $S' \to \bullet\ S$ \$, $\varnothing$, 0b: $S' \to S \bullet$ \$, $\varnothing$, 0c: $S' \to S$ \$ $\bullet$, $\varnothing$
- 1a: $S \to \bullet\ D\ E$, {\$}, 1b: $S \to D \bullet E$, {\$}, 1c: $S \to D\ E \bullet$, {\$}
- 2a: $D \to \bullet$ d D, {e, \$}, 2b: $D \to$ d $\bullet\ D$, {e, \$}, 2c: $D \to$ d $D \bullet$, {e, \$}
- 3b: $D \to \varepsilon \bullet$, {e, \$}
- 4a: $E \to \bullet$ e E, {\$}, 4b: $E \to$ e $\bullet\ E$, {\$}, 4c: $E \to$ e $E \bullet$, {\$}
- 5b: $E \to \varepsilon \bullet$, {\$}

b. (LR(1) Action/Go-to Table for this grammar.)

State #	Items	Actions			GoTo		
		d	**e**	**\$**	***S***	***D***	***E***
0	{0a: $S' \to \bullet\ S$ \$, $\varnothing$, 1a: $S \to \bullet\ D\ E$, {\$}, 2a: $D \to \bullet$ d D, {e, \$}, 3b: $D \to \varepsilon \bullet$, {e, \$}}	s4: 2b …	r3	r3	1: 0b	2: 1b	
1	{0b: $S' \to S \bullet$ \$, $\varnothing$}			accept			
2	{1b: $S \to D \bullet E$, {\$}, 4a: $E \to \bullet$ e E, {\$}, 5b: $E \to \varepsilon \bullet$, {\$}}		s6: 4b …	r5			3: 1c
3	{1c: $S \to D\ E \bullet$, {\$}}			r1			
4	2b: $D \to$ d $\bullet\ D$, {e, \$}, 2a: $D \to \bullet$ d D, {e, \$}, 3b: $D \to \varepsilon \bullet$, {e, \$}}	s4: 2b …	r3	r3		5: 2c	
5	{2c: $D \to$ d $D \bullet$, {e, \$}}		r2	r2			
6	{4b: $E \to$ e $\bullet\ E$, {\$}, 4a: $E \to \bullet$ e E, {\$}, 5b: $E \to \varepsilon \bullet$, {\$}}		s6: 4b …	r5			7: 4c
7	{4c: $E \to$ e $E \bullet$, {\$}}			r4			

c. Trace the execution of the parser on inputs d d e e \$, d d \$, e e \$, and \$.

LR(1) Parse of d d e e \$

Stack (top at right)	Input	Action
0	d d e e \$	s4
0 d 4	d e e \$	s4
0 d 4 d 4	e e \$	r3
0 d 4 d 4 *D* 5	e e \$	r2
0 d 4 *D* 5	e e \$	r2
0 *D* 2	e e \$	s6
0 *D* 2 e 6	e \$	s6
0 *D* 2 e 6 e 6	\$	r5
0 *D* 2 e 6 e 6 *E* 7	\$	r4
0 *D* 2 e 6 *E* 7	\$	r4
0 *D* 2 *E* 3	\$	r1
0 *S* 1	\$	accept

LR(1) Parse of d d \$

Stack (top at right)	Input	Action
0	d d \$	s4
0 d 4	d \$	s4
0 d 4 d 4	\$	r3
0 d 4 d 4 *D* 5	\$	r2
0 d 4 *D* 5	\$	r2
0 *D* 2	\$	r5
0 *D* 2 *E* 3	\$	r1
0 *S* 1	\$	accept

LR(1) Parse of `e e $`

Stack (top at right)	Input	Action
0	e e $	r3
0 *D* 2	e e $	s5
0 *D* 2 e 6	e $	s6
0 *D* 2 e 6 e 6	$	r5
0 *D* 2 e 6 e 6 *E* 7	$	r5
0 *D* 2 e 6 *E* 7	$	r5
0 *D* 2 *E* 3	$	r1
0 *S* 1	$	accept

LR(1) Parse of `$`

Stack (top at right)	Input	Action
0	$	r3
0 *D* 2	$	r5
0 *D* 2 *E* 3	$	r1
0 *S* 1	$	accept

