# *LR Parsing pt. 4: LALR(1) Parsers*

*CS 440: Programming Languages and Translators, Spring 2020*

4/8 p.3

## *1. Review - LR(1) Parsers*

- LR(1) parsers (a.k.a. full or canonical LR(1) parsers) are more exacting in their reduction decisions than SLR(1) parsers: If an LR(1) parser reduces, the corresponding SLR(1) parser will also reduce, but not necessarily vice versa.
  - An LR(1) parser reduces $A \rightarrow \alpha \bullet, L$ if the next symbol $\in L$, which is a subset of Follow($A$).
  - An SLR(1) parser handles $A \rightarrow \alpha \bullet$ the same way an LR(1) parser treats $A \rightarrow \alpha \bullet$, Follow($A$).
- LR(1) parsers use states that extend LR(0) items by including the lookahead set $L$.
  - Since the unique start symbol never gets reduced, its rule has an empty lookahead set, e.g. $S' \rightarrow S\ \$, \varnothing$.
  - A grammar rule $A \rightarrow \alpha$ generates different sets of LR(1) items $A \rightarrow \bullet\ \alpha, L$ where $L$ depends on where we were in the parse before starting the parse of $A \rightarrow \alpha$.
  - If "$\bullet\ A$" appears in the an item for a rule for $B$, say $(B \rightarrow \gamma \bullet A\ \delta, L')$, then we get an LR(1) item $A \rightarrow \bullet\ \alpha, L$ where $L = \text{First}(\delta\ L')$. (Or, spelling this out, if $\delta \nrightarrow^* \varepsilon$, then $L = \text{First}(\delta)$, otherwise $L = (\text{First}(\delta) - \varepsilon) \cup L'$.) Note we get an $L$ for each different $L'$ of the set of items $(B \rightarrow \gamma \bullet A\ \delta, L')$.
- The process of creating different LR(1) items from a single LR(0) item is called "splitting an LR(0) state."
  - Where an SLR(1) parser has just one item $A \rightarrow \bullet\ \alpha$, an LR(1) parser will have $A \rightarrow \bullet\ \alpha, L$ for each occurrence of $A$ in the rhs of any rule in the grammar.
  - This can be a large number and can easily cause the number of states to explode exponentially.
  - As a result, full LR(1) parsers are generally not used because their Action/Go-To tables are too large to be practical.

## *2. State Merging and LALR(1) Parsing*

- In LALR(1) parsing ("LA" = "Look Ahead", "LR" = "LR parsing", certain sets of LR(1) states are merged together into one new state.
- Specifically, two LR(1) states are merged if their LR(0) "core items" must match.
  - I.e., temporarily ignore the lookahead part of the LR(1) items. The results are the core LR(0) items.
  - If two states have exactly the same set of core items, we merge their corresponding LR(1) states together (so the new state will have the same core items but with possibly more lookahead symbols attached).
- LALR(1) parsers have the advantage that their parsing tables are generally quite a bit smaller than a full LR(1) parser's tables, but LALR(1) parsers can retain enough of the separation of follow sets to be more stringent in their reductions than SLR(1) parsers.
- In addition, LALR(1) grammars are expressive enough to handle modern programming languages (possibly with some extra hacks tossed in). As a result, LALR(1) parsers are what a parser-generator typically produces. (A parser generator takes a grammar description and builds a parser for it.)

 

- Note that an LALR(1) parser doesn't have to be smaller than a full LR(1) parser, nor does it have to be larger than an SLR(1) parser.
  - If no state merging can be done, then the LALR(1) parser is the same as the starting LR(1) parser.
  - If states can be merged to the point where all the lookahead sets equal the follow set of the lhs nonterminal, then the LALR(1) parser is the same as the SLR(1) parser.

### *3. Example 1: LALR(1) parser from LR(1) parser*

- The grammar has rules 0: $S' \rightarrow S$ `$`, 1: $S \rightarrow A$ `a` $A$ `a`, 2: $S \rightarrow A$ `b` $A$ `b`, 3: $A \rightarrow$ `c`
- Here's the LR(1) parsing table:

*LR(1) Action/Go-To Tables*

| | | *Action* | | | | *Go-To* | |
|---|---|---|---|---|---|---|---|
| **Nbr** | **Item** | `a` | `b` | `c` | `$` | *S* | *A* |
| 0 | {0a: $S' \rightarrow \bullet S$ `$`, ∅,<br>1a: $S \rightarrow \bullet A$ `a` $A$ `a`, {`$`},<br>2a: $S \rightarrow \bullet A$ `b` $A$ `b`, {`$`},<br>3.1a: $A \rightarrow \bullet$ `c`, {`a`},<br>3.2a: $A \rightarrow \bullet$ `c`, {`b`} } | | | s9 | | 1 | 2 |
| 1 | {0b: $S' \rightarrow S \bullet$ `$`, ∅} | | | | accept | | |
| 2 | {1b: $S \rightarrow A \bullet$ `a` $A$ `a`, {`$`},<br>2b: $S \rightarrow A \bullet$ `b` $A$ `b`, {`$`} } | s3 | s6 | | | | |
| 3 | {1c: $S \rightarrow A$ `a` $\bullet A$ `a`, {`$`},<br>3.1a: $A \rightarrow \bullet$ `c`, {`a`}} | | | s9 | | | 4 |
| 4 | {1d: $S \rightarrow A$ `a` $A \bullet$ `a`, {`$`}} | s5 | | | | | |
| 5 | {1e: $S \rightarrow A$ `b` $A$ `a` $\bullet$, {`$`}} | | | | r1 | | |
| 6 | {2b: $S \rightarrow A$ `b` $\bullet A$ `b`, {`$`}<br>3.2a: $A \rightarrow \bullet$ `c`, {`b`}} | | | s10 | | | 7 |
| 7 | {2c: $S \rightarrow A$ `b` $A \bullet$ `b`, {`$`}} | | s8 | | | | |
| 8 | {2d: $S \rightarrow A$ `b` $A$ `b` $\bullet$, {`$`}} | | | | r2 | | |
| 9 | {3.1b: $A \rightarrow$ `c` $\bullet$, {`a`}<br>3.2b: $A \rightarrow$ `c` $\bullet$, {`b`}} | r3 | r3 | | | | |

- The LR(1) to LALR(1) transformation does nothing because there aren't any states with matching LR(0) core items.
- This grammar is actually SLR(1): If we look at state 9, we see the reduction is done if the next symbol $\in$ {a, b}, which is Follow(A). The table row contains the same reductions for all members of Follow(A).

### *LR(1) > LALR(1) > SLR(1) Parsers*

- Let's take the grammar with rules 0: $S' \rightarrow S$ $, 1: $S \rightarrow A$ a $A$ b $C$, 2: $A \rightarrow$ c, 3: $C \rightarrow A$ d, 4: $C \rightarrow$ c c and compare the LR(1), LALR(1), and SLR(1) parsers for it .
- We'll find that the LR(1) parser detects errors earlier than the LALR(1) parser, which in turn detects errors earlier than the SLR(1) parser.
- **Example 2:** First let's look at the LR(1) parser's Action and Go-To tables. One thing to note is that rule 2: $A \rightarrow$ c comes with three different lookaheads; we get 2.1a: $A \rightarrow \bullet$ c, {a}, 2.2a: $A \rightarrow \bullet$ c, {b}, and 2.3a: $A \rightarrow \bullet$ c, {d} and three similar versions for $A \rightarrow$ c $\bullet$.

  [4/8: states 3, 6 below]

*First & Follow sets*

| NonT. | First | Follow |
|---|---|---|
| $S'$ | {c} | ∅ |
| $S$ | {c} | {$} |
| $A$ | {c} | {a, b, d} |
| $C$ | {c} | {$} |

*LR(1) Action/GoTo Tables*

| State | Items | a | b | c | d | $ | $S$ | $A$ | $C$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | {0a: $S' \rightarrow \bullet S$ $, ∅,<br>1a: $S \rightarrow \bullet A$ a $A$ b $C$, {$}<br>2.1a: $A \rightarrow \bullet$ c, {a}} | | | s11: 2.1b | | | 1: 0b | 2: 1b | |
| 1 | {0b: $S' \rightarrow S \bullet$ $, ∅} | | | | | accept | | | |
| 2 | {1b: $S \rightarrow A \bullet$ a $A$ b $C$, {$}} | s3: 1c | | | | | | | |
| 3 | {1c: $S \rightarrow A$ a $\bullet A$ b $C$, {$},<br>2.2a: $A \rightarrow \bullet$ c, {b}} | | | s12: 2.3b | | | | 4: 1d | |
| 4 | {1d: $S \rightarrow A$ a $A \bullet$ b $C$, {$}} | | s5: 1e | | | | | | |
| 5 | {1e: $S \rightarrow A$ a $A$ b $\bullet C$, {$},<br>3a: $C \rightarrow \bullet A$ d, {$},<br>2.3a: $A \rightarrow \bullet$ c, {d},<br>4a: $C \rightarrow \bullet$ c c, {$}} | | | s9: 2.3b | r2 | | | 7: 3b | 6: 1f |
| 6 | {1f: $S \rightarrow A$ a $A$ b $C \bullet$, {$}} | | | | | r1 | | | |
| 7 | {3b: $C \rightarrow A \bullet$ d, {$}} | | | | s8: 3c | | | | |
| 8 | {3c: $C \rightarrow A$ d $\bullet$, {$}} | | | | | r3 | | | |
| 9 | {2.3b: $A \rightarrow$ c $\bullet$, {d},<br>4b: $C \rightarrow$ c $\bullet$ c, {$}} | | | s10: 4c | r2 | | | | |
| 10 | {4c: $C \rightarrow$ c c $\bullet$, {$}} | | | | | r4 | | | |
| 11 | {2.1b: $A \rightarrow$ c $\bullet$, {a}} | r2 | | | | | | | |
| 12 | {2.2b: $A \rightarrow$ c $\bullet$, {b}} | | r2 | | | | | | |

- Here are a couple of sample successful parses. Since the two strings `c a c b c c` and `c a c b c d` only differ in their final symbol, their parses are the same until we reach the ending `c` or `d`:

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

| Step | Stack (top at right) | Input | Action |
|---|---|---|---|
| 1 | 0 | c a c b c c $ | s11 |
| 2 | 0 c 11 | a c b c c $ | r2 |
| 3 | 0 *A* 2 | a c b c c $ | s3 |
| 4 | 0 *A* 2 a 3 | c b c c $ | s12 |
| 5 | 0 *A* 2 a 3 c 12 | b c c $ | r2 |
| 6 | 0 *A* 2 a 3 *A* 4 | b c c $ | s5 |
| 7 | 0 *A* 2 a 3 *A* 4 b 5 | c c $ | s9 |
| 8 | 0 *A* 2 a 3 *A* 4 b 5 c 9 | c $ | s10 |
| 9 | 0 *A* 2 a 3 *A* 4 b 5 c 8 c 10 | $ | r4 |
| 10 | 0 *A* 2 a 3 *A* 4 b 5 *C* 6 | $ | r1 |
| 11 | 0 *S* 1 | $ | accept |

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

| Step | Stack (top at right) | Input | Action |
|---|---|---|---|
| 1 | 0 | c a c b c d $ | s11 |
| | *... Like parse of* `c a c b c c $` *...* | | |
| 8 | 0 *A* 2 a 3 *A* 4 b 5 c 9 | d $ | r2 |
| 9 | 0 *A* 2 a 3 *A* 4 b 5 *A* 7 | d $ | s8 |
| 10 | 0 *A* 2 a 3 *A* 4 b 5 *A* 7 d 8 | $ | r3 |
| 11 | 0 *A* 2 a 3 *A* 4 b 5 *C* 6 | $ | r1 |
| 12 | 0 *S* 1 | $ | accept |

- **Example 3:** To get the LALR(1) parser, we have to merge states with identical LR(0) items; the only ones here are states 11: {2.1b: $A \rightarrow$ `c` •, {`a`}} and 12: {2.2b: $A \rightarrow$ `c` •, {`b`}}, which we can merge into a new state 11*: {2.1b, 2.2b: $A \rightarrow$ `c` •, {`a`, `b`}}. Only the changed states are shown below, with the changes in red.

| State | Items | a | b | c | d | $ | *S* | *A* | *C* |
|---|---|---|---|---|---|---|---|---|---|
| 0 | {0a: $S' \rightarrow \bullet S$ $, ∅,<br>1a: $S \rightarrow \bullet A$ a $A$ b $C$, {$}<br>2.1a: $A \rightarrow \bullet$ c, {a}} | | | s11′ | | | 1: 0b | 2: 1b | |
| … | … | | | | | | | | |
| 3 | {1c: $S \rightarrow A$ a • $A$ b $C$, {$},<br>2.3a: $A \rightarrow \bullet$ c, {b}} | | | s11′ | | | | 4: 1d | |
| … | … | | | | | | | | |
| 11′ | {2.1b, 2.2b: $A \rightarrow$ c •, {a, b}} | r2 | r2 | | | | | | |

- **Example 4:** The SLR(1) parser reduces if the next symbol is in the follow of the lhs nonterminal.
  - For this grammar, the only rule with different LR(1) and SLR(1) reductions is $A \rightarrow$ `c`; instead of the LR(1) lookahead sets {`a`}, {`b`}, or {`d`}, the SLR(1) parser in effect has a lookahead set of {`a`, `b`, `d`}.

 

- The SLR(1) parser below is written using LR(1) items (normally the lookahead sets are omitted) with changes relative to the LR(1) parser marked in red. State 11* combines the LR(1) states 11 and 12.
- Since there are no conflicts, the grammar is SLR(1).

| State | Items | a | b | c | d | $ | *S* | *A* | *C* |
|---|---|---|---|---|---|---|---|---|---|
| 0 | {0a: $S' \to \bullet\ S\ \$, \varnothing$,<br>1a: $S \to \bullet\ A\ \mathtt{a}\ A\ \mathtt{b}\ C$, {$}<br>2.1a: $A \to \bullet\ \mathtt{c}$, {a, b, d}} | | | s11* | | | 1: 0b | 2: 1b | |
| 1 | {0b: $S' \to S \bullet\ \$, \varnothing$} | | | | | accept | | | |
| 2 | {1b: $S \to A \bullet\ \mathtt{a}\ A\ \mathtt{b}\ C$, {$}} | s3: 1c | | | | | | | |
| 3 | {1c: $S \to A\ \mathtt{a} \bullet A\ \mathtt{b}\ C$, {$},<br>2.3a: $A \to \bullet\ \mathtt{c}$, {a, b, d}} | | | s11* | | | | 4: 1d | |
| 4 | {1d: $S \to A\ \mathtt{a}\ A \bullet\ \mathtt{b}\ C$, {$}} | | s5: 1e | | | | | | |
| 5 | {1e: $S \to A\ \mathtt{a}\ A\ \mathtt{b} \bullet C$, {$},<br>3a: $C \to \bullet\ A\ \mathtt{d}$, {$},<br>2.4a: $A \to \bullet\ \mathtt{c}$, {a, b, d},<br>4a: $C \to \bullet\ \mathtt{c}\ \mathtt{c}$, {$}} | | | s9: 2.4b | | | | 7: 3b | 6: 1f |
| 6 | {1f: $S \to A\ \mathtt{b}\ A\ \mathtt{b}\ C \bullet$, {$}} | | | | | r1 | | | |
| 7 | {3b: $C \to A \bullet\ \mathtt{d}$, {$}} | | | | s8: 3c | | | | |
| 8 | {3c: $C \to A\ \mathtt{d} \bullet$, {$}} | | | | | r3 | | | |
| 9 | {2.4b: $A \to \mathtt{c} \bullet$, {a, b, d},<br>4b: $C \to \mathtt{c} \bullet\ \mathtt{c}$, {$}} | r2 | r2 | s10: 4c | r2 | | | | |
| 10 | {4c: $C \to \mathtt{c}\ \mathtt{c} \bullet$, {$}} | | | | | r4 | | | |
| 11* | {2.1b, 2.2b: $A \to \mathtt{c} \bullet$, {a, b, d}} | r2 | r2 | | r2 | | | | |

 

# *Activity Problems for Lecture 20*

For Problems 1 and 2, take the grammar used in Examples 2 – 4 and trace the execution of various parses to verify some properties of the different parsers

1. The LR(1) parser is stricter (finds the error sooner) than the SLR(1) and LALR(1) parsers.
   a. The LR(1) parser flags input `c a c a` as an error on the second `a` because the next input is `a` instead of `b`.
   b. On the same input, the SLR(1) and LALR(1) parsers reduce $A \rightarrow$ `c` and generate the error when they can't shift `a`.

2. The LALR(1) parser is stricter than the SLR(1) parser.
   a. On input `c a c b c a`, the LALR(1) parser generates an error on the second `a` because it can't reduce $A \rightarrow$ `c` here when the next symbol is `a` (it wants `d`).
   b. On the same input, the SLR(1) parser reduces $A \rightarrow$ `c` and generates an error when it can't shift the `a`.

## *(Solutions omitted)*

 