Syntax Analysis: LR(1) and LALR Parsing

Lecture 7

Objectives

By the end of this lecture you should be able to:

1. Identify LR(1) items.
2. Construct an LR(1) automaton for a CFG.
3. Construct the LR(1) parsing table for a CFG.
4. Trace the operation of an LR(1) parser.
5. Construct an LALR automaton for a CFG.
6. Construct the LALR parsing table for a CFG.
7. Trace the operation of an LALR parser.

Outline

1. Canonical LR(1) Parsing
2. LALR Parsing

Outline

Grammars Not SLR

Example

Consider the following grammar G_7:

$$\begin{array}{rcl} S & \longrightarrow & L{=}R \mid R \\ L & \longrightarrow & *R \mid \mathbf{id} \\ R & \longrightarrow & L \end{array}$$

Grammars Not SLR: States

Example

I_0: $S' \rightarrow \cdot S$
$S \rightarrow \cdot L = R$
$S \rightarrow \cdot R$
$L \rightarrow \cdot * R$
$L \rightarrow \cdot \mathbf{id}$
$R \rightarrow \cdot L$

I_1: $S' \rightarrow S \cdot$

I_2: $S \rightarrow L \cdot = R$
$R \rightarrow L \cdot$

I_3: $S \rightarrow R \cdot$

I_4: $L \rightarrow * \cdot R$
$R \rightarrow \cdot L$
$L \rightarrow \cdot * R$
$L \rightarrow \cdot \mathbf{id}$

I_5: $L \rightarrow \mathbf{id} \cdot$

I_6: $S \rightarrow L = \cdot R$
$R \rightarrow \cdot L$
$L \rightarrow \cdot * R$
$L \rightarrow \cdot \mathbf{id}$

I_7: $L \rightarrow * R \cdot$

I_8: $R \rightarrow L \cdot$

I_9: $S \rightarrow L = R \cdot$

Problems with SLR Parsing

One problem with SLR parsers is that it is always possible to reduce $A \longrightarrow \alpha$ if the next input is in $Follow(A)$.

Example

- With G_7 and input **id** $=$ **id**, an SLR parser may shift **id** and then reduce $L \longrightarrow$ **id**.
- Since $= \in Follow(R)$, the parser may decide to reduce $R \longrightarrow L$.
- Clearly, this is not correct; a sentential form starting with $R=$ can never reduce to S.

States should carry more information about when reduction is appropriate.

Problems with SLR Parsing

One problem with SLR parsers is that it is always possible to reduce $A \longrightarrow \alpha$ if the next input is in $Follow(A)$.

Example

- With G_7 and input $\mathbf{id} = \mathbf{id}$, an SLR parser may shift **id** and then reduce $L \longrightarrow \mathbf{id}$.
- Since $= \in Follow(R)$, the parser may decide to reduce $R \longrightarrow L$.
- Clearly, this is not correct; a sentential form starting with $R=$ can never reduce to S.

States should carry more information about when reduction is appropriate.

Problems with SLR Parsing

One problem with SLR parsers is that it is always possible to reduce $A \longrightarrow \alpha$ if the next input is in $Follow(A)$.

Example

- With G_7 and input $\mathbf{id} = \mathbf{id}$, an SLR parser may shift **id** and then reduce $L \longrightarrow \mathbf{id}$.
- Since $= \in Follow(R)$, the parser may decide to reduce $R \longrightarrow L$.
- Clearly, this is not correct; a sentential form starting with $R=$ can never reduce to S.

States should carry more information about when reduction is appropriate.

LR(0) Items: Reprise

Definition

An LR(0) item of CFG $G = \langle V, \Sigma, R, S \rangle$ is a pair $\langle A \rightarrow \alpha, i \rangle$, where $(A \rightarrow \alpha) \in R$ and $0 \leq i \leq |\alpha|$.

- Intuitively, an LR(0) item is a rule and a position in the right side of the rule.
  - Thus, $\langle A \rightarrow \mathtt{a}B\mathtt{b}, 2 \rangle \equiv A \rightarrow \mathtt{a}B.\mathtt{b}$
- An LR(0) item represents a state of the parser: *We have already found the prefix of α before the dot and, if we find the suffix following the dot, we may reduce α to A.*

LR(1) Items

Definition

An LR(1) item of CFG $G = \langle V, \Sigma, R, S \rangle$ is a pair $\langle c, a \rangle$, where c is an LR(0) item and $a \in \Sigma \cup \{\$\}$.

- c is called the core of the item.
- a is the lookahead of the item.
  - Note that a is a single symbol, hence the "1" in "LR(1) item."
- An LR(1) item $[A \to \alpha.\beta, a]$ represents a state of the parser: *We have already found α in the input and, if we find β, we may reduce $\alpha\beta$ to A if the next symbol is a.*

LR(1) Items

Definition

An LR(1) item of CFG $G = \langle V, \Sigma, R, S \rangle$ is a pair $\langle c, a \rangle$, where c is an LR(0) item and $a \in \Sigma \cup \{\$\}$.

- c is called the core of the item.
- a is the lookahead of the item.
  - Note that a is a single symbol, hence the "1" in "LR(1) item."
- An LR(1) item $[A \to \alpha.\beta, a]$ represents a state of the parser: *We have already found α in the input and, if we find β, we may reduce $\alpha\beta$ to A if the next symbol is a.*

LR(1) Items

Definition

An LR(1) item of CFG $G = \langle V, \Sigma, R, S \rangle$ is a pair $\langle c, a \rangle$, where c is an LR(0) item and $a \in \Sigma \cup \{\$\}$.

- c is called the core of the item.
- a is the lookahead of the item.
  - Note that a is a single symbol, hence the "1" in "LR(1) item."
- An LR(1) item $[A \rightarrow \alpha.\beta, a]$ represents a state of the parser: *We have already found α in the input and, if we find β, we may reduce $\alpha\beta$ to A if the next symbol is a.*

LR(1) Items

Definition

An LR(1) item of CFG $G = \langle V, \Sigma, R, S \rangle$ is a pair $\langle c, a \rangle$, where c is an LR(0) item and $a \in \Sigma \cup \{\$\}$.

- c is called the core of the item.
- a is the lookahead of the item.
  - Note that a is a single symbol, hence the "1" in "LR(1) item."
- An LR(1) item $[A \rightarrow \alpha.\beta, a]$ represents a state of the parser: *We have already found α in the input and, if we find β, we may reduce $\alpha\beta$ to A if the next symbol is a.*

The Set of LR(1) Items

The smallest set $\mathcal{I}(G)$ satisfying the following

- $[S' \longrightarrow .S, \$] \in \mathcal{I}$.
- $[A \longrightarrow \alpha X.\beta, a] \in \mathcal{I}$ if $[A \longrightarrow \alpha.X\beta, a] \in \mathcal{I}$, for $X \in \Sigma \cup V$.
- $[B \longrightarrow .\gamma, b] \in \mathcal{I}$ if $[A \longrightarrow \alpha.B\beta, a] \in \mathcal{I}$ where $(B \longrightarrow \gamma) \in R$ and $b \in First(\beta a)$.

The LR(1) NFA

Definition

For the CFG $G = \langle V, \Sigma, R, S \rangle$, the LR(1) NFA is an NFA $N_G^1 = \langle \mathcal{I}(G), V \cup \Sigma, \delta, [S' \rightarrow .S, \$], \mathcal{I}(G) \rangle$, where

- $S' \notin V \cup \Sigma$;
- $\delta([A \rightarrow \alpha.X\beta, a], X) = \{[A \rightarrow \alpha X.\beta, a]\}$;
- $\delta([A \rightarrow \alpha.B\beta, a], \varepsilon) = \{[B \rightarrow .\gamma, b] \mid (B \rightarrow \gamma) \in R$ and $b \in First(\beta a)\}$.

The LR(1) Automaton

Definition

The LR(1) automaton for a CFG G is the DFA M_G^1 which is equivalent to N_G^1 and constructed using the standard subset construction.

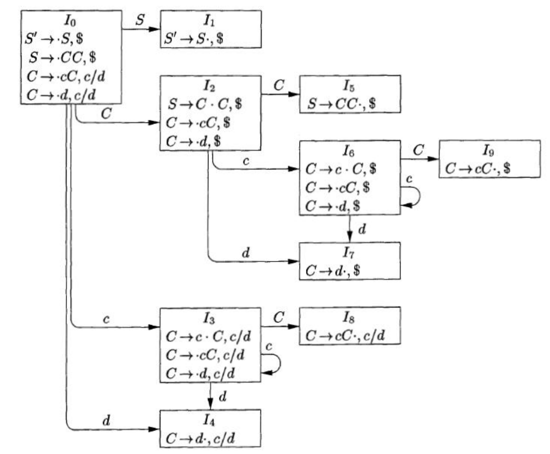
Grammar G_8

Example (The grammar)

$$
\begin{array}{llll}
1. & S & \longrightarrow & CC \\
2. & C & \longrightarrow & \mathtt{c}C \\
3. & C & \longrightarrow & \mathtt{d}
\end{array}
$$

Grammar G_8

Example (The automaton)



©Aho et al. (2007)

The Canonical LR(1) Parsing Table

We are given a CFG $G = \langle V, \Sigma, R, S \rangle$.

1. Construct M_G^1.
2. For all states q of M_G^1
   1. GOTO$(q, A) = \delta(q, A)$, for every $A \in V$.
   2. If $a \in \Sigma$ and $[A \rightarrow \alpha.a\beta, b] \in q$, then ACTION$(q, a) =$ "shift $\delta(q, a)$".
   3. Else if $A \neq S'$ and $[A \rightarrow \alpha., a] \in q$, then ACTION$(q, a) =$ "reduce $A \rightarrow \alpha$".
   4. Else if $[S' \rightarrow S., \$] \in q$, then ACTION$(q, \$)$ = "accept".
   5. Else ACTION(q, a) = "error".

If any conflicting actions result from the above construction, we say that G is not LR(1).

The Canonical LR(1) Parsing Table

We are given a CFG $G = \langle V, \Sigma, R, S \rangle$.

1. Construct M_G^1.
2. For all states q of M_G^1
    1. $\text{GOTO}(q, A) = \delta(q, A)$, for every $A \in V$.
    2. If $a \in \Sigma$ and $[A \rightarrow \alpha.a\beta, b] \in q$, then $\text{ACTION}(q, a) =$ "shift $\delta(q, a)$".
    3. Else if $A \neq S'$ and $[A \rightarrow \alpha., a] \in q$, then $\text{ACTION}(q, a) =$ "reduce $A \rightarrow \alpha$".
    4. Else if $[S' \rightarrow S., \$] \in q$, then ACTION($q$, \$) = "accept".
    5. Else ACTION(q, a) = "error".

If any conflicting actions result from the above construction, we say that G is not LR(1).

Grammar G_8

Example (Table ▸ Automaton)

STATE	ACTION			GOTO	
	c	d	\$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

Grammar G_8

Example (Trace ▸ Automaton)

Stack	Input
0	ccdd$
03	cdd$
033	dd$
0334	d$
0338	d$
038	d$
02	d$
027	$
025	$
01	$

Canonical LR(1) and SLR Parsing

- Note that a canonical LR(1) table can include conflicts only if the corresponding SLR table does.
- Thus, every SLR grammar is a canonical LR(1) grammar, but not vice versa.
- The price, however, is the increased table size due to the increased number of automaton states.

Outline

1. Canonical LR(1) Parsing
2. LALR Parsing

Lookahead LR Parsing

- Lookahead LR parsers (LALR parsers) are often used in practice.
- Most common syntactic constructs in programming languages are representable by LALR grammars.
- LALR tables are considerably smaller than canonical LR(1) tables.
  - SLR and LALR tables always have the same number of states.
  - Such number is typically several hundred states.
  - A canonical LR(1) table typically has several thousand states!

Lookahead LR Parsing

- Lookahead LR parsers (LALR parsers) are often used in practice.
- Most common syntactic constructs in programming languages are representable by LALR grammars.
- LALR tables are considerably smaller than canonical LR(1) tables.
  - SLR and LALR tables always have the same number of states.
  - Such number is typically several hundred states.
  - A canonical LR(1) table typically has several thousand states!

Lookahead LR Parsing

- Lookahead LR parsers (LALR parsers) are often used in practice.
- Most common syntactic constructs in programming languages are representable by LALR grammars.
- LALR tables are considerably smaller than canonical LR(1) tables.
  - SLR and LALR tables always have the same number of states.
  - Such number is typically several hundred states.
  - A canonical LR(1) table typically has several thousand states!

Lookahead LR Parsing

- Lookahead LR parsers (LALR parsers) are often used in practice.
- Most common syntactic constructs in programming languages are representable by LALR grammars.
- LALR tables are considerably smaller than canonical LR(1) tables.
  - SLR and LALR tables always have the same number of states.
  - Such number is typically several hundred states.
  - A canonical LR(1) table typically has several thousand states!

Lookahead LR Parsing

- Lookahead LR parsers (LALR parsers) are often used in practice.
- Most common syntactic constructs in programming languages are representable by LALR grammars.
- LALR tables are considerably smaller than canonical LR(1) tables.
  - SLR and LALR tables always have the same number of states.
  - Such number is typically several hundred states.
  - A canonical LR(1) table typically has several thousand states!

Lookahead LR Parsing

- Lookahead LR parsers (LALR parsers) are often used in practice.
- Most common syntactic constructs in programming languages are representable by LALR grammars.
- LALR tables are considerably smaller than canonical LR(1) tables.
  - SLR and LALR tables always have the same number of states.
  - Such number is typically several hundred states.
  - A canonical LR(1) table typically has several thousand states!

The Core

Definition

The core of an LR(1) state q is the set

$$core(q) = \{c | [c, l] \in q\}$$

Two LR(1) states q_1 and q_2 are core-equivalent if $core(q_1) = core(q_2)$.

Example (Automaton of G_8)

The following states are core-equivalent.

- I_4 and I_7.
- I_8 and I_9.
- I_3 and I_6.

How are they different?

The Core

Definition

The core of an LR(1) state q is the set

$$core(q) = \{c | [c, l] \in q\}$$

Two LR(1) states q_1 and q_2 are core-equivalent if $core(q_1) = core(q_2)$.

Example (Automaton of G_8)

The following states are core-equivalent.

- I_4 and I_7.
- I_8 and I_9.
- I_3 and I_6.

How are they different?

Core-Equivalence and the Automaton

Observation

Let M_G^1 be the LR(1) DFA for a CFG G, and let q_1, q_2 be states of M_G^1 and $X \in V \cup \Sigma$. If q_1 and q_2 are core-equivalent, then $\delta(q_1, X)$ and $\delta(q_2, X)$ are core-equivalent.

- The number of states of the automaton may be reduced considerably if we use core-equivalence-classes as states.
- In particular, if $\{q_1, \ldots, q_n\}$ is an equivalence class of core-equivalence, then we may replace the states $q_1, \ldots, q_n$ by the single state $Q = q_1 \cup \cdots \cup q_n$.
- Transitions are adjusted appropriately.

Core-Equivalence and the Automaton

Observation

Let M_G^1 be the LR(1) DFA for a CFG G, and let q_1, q_2 be states of M_G^1 and $X \in V \cup \Sigma$. If q_1 and q_2 are core-equivalent, then $\delta(q_1, X)$ and $\delta(q_2, X)$ are core-equivalent.

- The number of states of the automaton may be reduced considerably if we use core-equivalence-classes as states.
- In particular, if $\{q_1, \ldots, q_n\}$ is an equivalence class of core-equivalence, then we may replace the states $q_1, \ldots, q_n$ by the single state $Q = q_1 \cup \cdots \cup q_n$.
- Transitions are adjusted appropriately.

Core-Equivalence and the Automaton

Observation

Let M_G^1 be the LR(1) DFA for a CFG G, and let q_1, q_2 be states of M_G^1 and $X \in V \cup \Sigma$. If q_1 and q_2 are core-equivalent, then $\delta(q_1, X)$ and $\delta(q_2, X)$ are core-equivalent.

- The number of states of the automaton may be reduced considerably if we use core-equivalence-classes as states.
- In particular, if $\{q_1, \ldots, q_n\}$ is an equivalence class of core-equivalence, then we may replace the states $q_1, \ldots, q_n$ by the single state $Q = q_1 \cup \cdots \cup q_n$.
- Transitions are adjusted appropriately.

Core-Equivalence and the Automaton

Observation

Let M_G^1 be the LR(1) DFA for a CFG G, and let q_1, q_2 be states of M_G^1 and $X \in V \cup \Sigma$. If q_1 and q_2 are core-equivalent, then $\delta(q_1, X)$ and $\delta(q_2, X)$ are core-equivalent.

- The number of states of the automaton may be reduced considerably if we use core-equivalence-classes as states.
- In particular, if $\{q_1, \ldots, q_n\}$ is an equivalence class of core-equivalence, then we may replace the states $q_1, \ldots, q_n$ by the single state $Q = q_1 \cup \cdots \cup q_n$.
- Transitions are adjusted appropriately.

The LALR Automaton

Definition

Let $\langle \mathcal{Q}(G), V \cup \Sigma, \delta, q_0, F \rangle$ be an LR(1) automaton. The LALR automaton $M_G^A = \langle \mathcal{Q}'(G), V \cup \Sigma, \delta', q_0', F' \rangle$ is such that

- $\mathcal{Q}'(G) = \{ \bigcup_{q_i \in [q]} q_i | [q]$ is an equivalence class of core-equivalence $\}$
- q_0 and q_0' are core-equivalent.
- $q' \in F'$ if and only if there is some $q \in F$ such that q and q' are core-equivalent.
- $\delta'(q', X)$ is core-equivalent to $\delta(q, X)$, where q and q' are core-equivalent.

The LALR Automaton

Definition

Let $\langle \mathcal{Q}(G), V \cup \Sigma, \delta, q_0, F \rangle$ be an LR(1) automaton. The LALR automaton $M_G^A = \langle \mathcal{Q}'(G), V \cup \Sigma, \delta', q_0', F' \rangle$ is such that

- $\mathcal{Q}'(G) = \{ \bigcup_{q_i \in [q]} q_i | [q]$ is an equivalence class of core-equivalence $\}$
- q_0 and q_0' are core-equivalent.
- $q' \in F'$ if and only if there is some $q \in F$ such that q and q' are core-equivalent.
- $\delta'(q', X)$ is core-equivalent to $\delta(q, X)$, where q and q' are core-equivalent.

The LALR Automaton

Definition

Let $\langle \mathcal{Q}(G), V \cup \Sigma, \delta, q_0, F \rangle$ be an LR(1) automaton. The LALR automaton $M_G^A = \langle \mathcal{Q}'(G), V \cup \Sigma, \delta', q_0', F' \rangle$ is such that

- $\mathcal{Q}'(G) = \{ \bigcup_{q_i \in [q]} q_i | [q]$ is an equivalence class of core-equivalence $\}$
- q_0 and q_0' are core-equivalent.
- $q' \in F'$ if and only if there is some $q \in F$ such that q and q' are core-equivalent.
- $\delta'(q', X)$ is core-equivalent to $\delta(q, X)$, where q and q' are core-equivalent.

The LALR Automaton

Definition

Let $\langle \mathcal{Q}(G), V \cup \Sigma, \delta, q_0, F \rangle$ be an LR(1) automaton. The LALR automaton $M_G^A = \langle \mathcal{Q}'(G), V \cup \Sigma, \delta', q_0', F' \rangle$ is such that

- $\mathcal{Q}'(G) = \{ \bigcup_{q_i \in [q]} q_i | [q]$ is an equivalence class of core-equivalence $\}$
- q_0 and q_0' are core-equivalent.
- $q' \in F'$ if and only if there is some $q \in F$ such that q and q' are core-equivalent.
- $\delta'(q', X)$ is core-equivalent to $\delta(q, X)$, where q and q' are core-equivalent.

Grammar G_8

Example (LALR automaton)

Draw the state diagram of the LALR automaton for G_8.

The LALR Parsing Table

- The LALR parsing table is constructed in exactly the same way the canonical LR(1) table is constructed, but with M_G^A rather than M_G^1.
- The size of the LALR table is the same as the size of the SLR table.
- If the table has no conflicts, the grammar is said to be an LALR grammar.

Grammar G_8

Example (LALR parsing table)

STATE	ACTION			GOTO	
	c	d	$\$$	S	C
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

© Aho et al. (2007)

Grammar G_8

Example (Trace)

Trace the operation of both the canonical LR(1) and the LALR parsers on input `ccd`. What do you notice?

LALR Conflicts (I)

Observation

If an LR(1) table does not have shift/reduce conflicts, then the corresponding LALR table does not have shift/reduce conflicts.

Proof. A shift/reduce conflict occurs if an LALR state has an item $[A \rightarrow \alpha., a]$, calling for reducing by $A \rightarrow \alpha$, and an item $[B \rightarrow \beta.a\gamma, b]$ calling for a shift. But, then, there must be some LR(1) state with items $[A \rightarrow \alpha., a]$ and $[B \rightarrow \beta.a\gamma, c]$. This state would also have a shift/reduce conflict, contradicting our assumption. $\square$

LALR Conflicts (I)

Observation

If an LR(1) table does not have shift/reduce conflicts, then the corresponding LALR table does not have shift/reduce conflicts.

Proof. A shift/reduce conflict occurs if an LALR state has an item $[A \rightarrow \alpha., a]$, calling for reducing by $A \rightarrow \alpha$, and an item $[B \rightarrow \beta.a\gamma, b]$ calling for a shift. But, then, there must be some LR(1) state with items $[A \rightarrow \alpha., a]$ and $[B \rightarrow \beta.a\gamma, c]$. This state would also have a shift/reduce conflict, contradicting our assumption. $\square$

LALR Conflicts (II)

Observation

If an LR(1) table does not have reduce/reduce conflicts, the corresponding LALR table may have reduce/reduce conflicts.

Proof. Consider the grammar

$$\begin{array}{lcl} S & \longrightarrow & \mathtt{a}A\mathtt{d} \mid \mathtt{a}B\mathtt{e} \mid \mathtt{b}B\mathtt{d} \mid \mathtt{b}A\mathtt{e} \\ A & \longrightarrow & \mathtt{c} \\ B & \longrightarrow & \mathtt{c} \end{array}$$

- This grammar generates the strings $\mathtt{acd}, \mathtt{ace}, \mathtt{bcd}, \mathtt{bce}$.
- The LR(1) state $\{[A \rightarrow c., d], [B \rightarrow c., e]\}$ is reachable from the start state after reading $\mathtt{ac}$.
- Similarly, the LR(1) state $\{[A \rightarrow c., e], [B \rightarrow c., d]\}$ is reachable from the start state after reading $\mathtt{bc}$.
- The LALR automaton will include the state $\{[A \rightarrow c., d/e], [B \rightarrow c., d/e]\}$, which results in a reduce/reduce conflict. $\square$

LALR Conflicts (II)

Observation

If an LR(1) table does not have reduce/reduce conflicts, the corresponding LALR table may have reduce/reduce conflicts.

Proof. Consider the grammar

$$\begin{array}{lcl} S & \longrightarrow & \texttt{a}A\texttt{d} \mid \texttt{a}B\texttt{e} \mid \texttt{b}B\texttt{d} \mid \texttt{b}A\texttt{e} \\ A & \longrightarrow & \texttt{c} \\ B & \longrightarrow & \texttt{c} \end{array}$$

- This grammar generates the strings `acd`, `ace`, `bcd`, `bce`.
- The LR(1) state $\{[A \to c., d], [B \to c., e]\}$ is reachable from the start state after reading `ac`.
- Similarly, the LR(1) state $\{[A \to c., e], [B \to c., d]\}$ is reachable from the start state after reading `bc`.
- The LALR automaton will include the state $\{[A \to c., d/e], [B \to c., d/e]\}$, which results in a reduce/reduce conflict. □

LALR Conflicts (II)

Observation

If an LR(1) table does not have reduce/reduce conflicts, the corresponding LALR table may have reduce/reduce conflicts.

Proof. Consider the grammar

$$\begin{array}{lcl} S & \longrightarrow & \mathtt{a}A\mathtt{d} \mid \mathtt{a}B\mathtt{e} \mid \mathtt{b}B\mathtt{d} \mid \mathtt{b}A\mathtt{e} \\ A & \longrightarrow & \mathtt{c} \\ B & \longrightarrow & \mathtt{c} \end{array}$$

- This grammar generates the strings `acd`, `ace`, `bcd`, `bce`.
- The LR(1) state $\{[A \rightarrow c., d], [B \rightarrow c., e]\}$ is reachable from the start state after reading `ac`.
- Similarly, the LR(1) state $\{[A \rightarrow c., e], [B \rightarrow c., d]\}$ is reachable from the start state after reading `bc`.
- The LALR automaton will include the state $\{[A \rightarrow c., d/e], [B \rightarrow c., d/e]\}$, which results in a reduce/reduce conflict. □

LALR Conflicts (II)

Observation

If an LR(1) table does not have reduce/reduce conflicts, the corresponding LALR table may have reduce/reduce conflicts.

Proof. Consider the grammar

$$\begin{array}{lcl} S & \longrightarrow & \mathtt{a}A\mathtt{d} \mid \mathtt{a}B\mathtt{e} \mid \mathtt{b}B\mathtt{d} \mid \mathtt{b}A\mathtt{e} \\ A & \longrightarrow & \mathtt{c} \\ B & \longrightarrow & \mathtt{c} \end{array}$$

- This grammar generates the strings `acd`, `ace`, `bcd`, `bce`.
- The LR(1) state $\{[A \rightarrow c., d], [B \rightarrow c., e]\}$ is reachable from the start state after reading `ac`.
- Similarly, the LR(1) state $\{[A \rightarrow c., e], [B \rightarrow c., d]\}$ is reachable from the start state after reading `bc`.
- The LALR automaton will include the state $\{[A \rightarrow c., d/e], [B \rightarrow c., d/e]\}$, which results in a reduce/reduce conflict. □

LALR Conflicts (II)

Observation

If an LR(1) table does not have reduce/reduce conflicts, the corresponding LALR table may have reduce/reduce conflicts.

Proof. Consider the grammar

$$\begin{array}{lcl} S & \longrightarrow & \mathtt{a}A\mathtt{d} \mid \mathtt{a}B\mathtt{e} \mid \mathtt{b}B\mathtt{d} \mid \mathtt{b}A\mathtt{e} \\ A & \longrightarrow & \mathtt{c} \\ B & \longrightarrow & \mathtt{c} \end{array}$$

- This grammar generates the strings `acd`, `ace`, `bcd`, `bce`.
- The LR(1) state $\{[A \rightarrow c., d], [B \rightarrow c., e]\}$ is reachable from the start state after reading `ac`.
- Similarly, the LR(1) state $\{[A \rightarrow c., e], [B \rightarrow c., d]\}$ is reachable from the start state after reading `bc`.
- The LALR automaton will include the state $\{[A \rightarrow c., d/e], [B \rightarrow c., d/e]\}$, which results in a reduce/reduce conflict. □