LALR Analysis
Motivation
• As explained before, in LR(1) parsers there are many more states than in the previous procedures, LR(0) and SLR(1).
  • This is because there are states which contain the same configurations, but with different look-ahead symbols.
• A possible simplification of LR(1) parsers are LALR(1) parsers (Look-Ahead Left-to-Right parsers)
  • They have the same number of states as LR(0) and SLR(1) parsers.
  • For illustration, a language like Pascal will have a few hundreds of states if constructed as SLR(1), but it will have thousands if built as LR(1).

• We are going to build LALR(1) parsers from LR(1) parsers

LALR Analysis
LALR parsers: introductory example
• We have seen an LR(1) example grammar, which was not SLR(1):
  (1) $S \rightarrow A$
  (2) $S \rightarrow xb$
  (3) $A \rightarrow aAb$
  (4) $A \rightarrow B$
  (5) $B \rightarrow x$

• this grammar generated the following language:
  \[
  \{xb\} \cup \{a^nxb^n \mid n \geq 0\}
  \]

LALR Analysis
LALR parsers: introductory example
• Remember the augmented grammar:

  \[
  \begin{align*}
  (0) & S' \rightarrow S$
  (1) & S \rightarrow A$
  (2) & S \rightarrow xb$
  (3) & A \rightarrow aAb$
  (4) & A \rightarrow B$
  (5) & B \rightarrow x
  \end{align*}
  \]

LALR Analysis
LALR parsers: introductory example
• SLR(1) Deterministic finite automata with the transitions

\[
\begin{array}{c}
\begin{tikzpicture}
\node (S) at (0,0) {$S'$};
\node (A) at (1,0) {$A$};
\node (B) at (2,0) {$B$};
\node (S1) at (1,1) {$S \rightarrow A$};
\node (S2) at (2,1) {$S \rightarrow B$};
\node (S3) at (1,2) {$A \rightarrow aAb$};
\node (S4) at (2,2) {$A \rightarrow B$};
\node (S5) at (1,3) {$B \rightarrow x$};
\node (S6) at (2,3) {$B \rightarrow xb$};
\node (S7) at (3,1) {$A \rightarrow aAb$};
\node (S8) at (4,1) {$A \rightarrow B$};
\node (S9) at (3,2) {$B \rightarrow x$};
\node (S10) at (4,2) {$B \rightarrow x$};
\node (S11) at (3,3) {$B \rightarrow xb$};
\node (S12) at (4,3) {$B \rightarrow xb$};
\draw[->] (S) -- (S1);
\draw[->] (S1) -- (S2);
\draw[->] (S2) -- (S3);
\draw[->] (S3) -- (S4);
\draw[->] (S4) -- (S5);
\draw[->] (S5) -- (S6);
\draw[->] (S6) -- (S7);
\draw[->] (S7) -- (S8);
\draw[->] (S8) -- (S9);
\draw[->] (S9) -- (S10);
\draw[->] (S10) -- (S11);
\draw[->] (S11) -- (S12);
\end{tikzpicture}
\end{array}
\]
LALR Analysis

**LALR parsers: introductory example**

- This was the analysis table for the SLR(1) grammar, with the conflict

<table>
<thead>
<tr>
<th>X_0</th>
<th>X_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>s3</td>
</tr>
<tr>
<td>1</td>
<td>s5</td>
</tr>
<tr>
<td>2</td>
<td>r1</td>
</tr>
<tr>
<td>3</td>
<td>r4</td>
</tr>
<tr>
<td>4</td>
<td>r5</td>
</tr>
<tr>
<td>5</td>
<td>r2</td>
</tr>
<tr>
<td>6</td>
<td>r3</td>
</tr>
</tbody>
</table>

- Deterministic Finite Automata with transitions for an LR(1) parser

- As can be seen,
  - The states in an LR(1) parser are the same as in an LR(0) parser, but some of them appear several times with different lookahead symbols in the configurations.
  - The idea is to simplify the analyser, by merging all the states with the same configurations but different lookahead symbols.
  - The resulting analyser will contain the same states than the SLR(1) parser.
  - Having less states, the resulting analyser will be less powerful than the original LR(1) parser, and there is a higher probability of having collisions (as happened in SLR(1) parsers)
  - On the other hand, there will be less collisions in LALR(1) parsers than in SLR(1) parsers, so there is a gain in this process.

- In the example,
  - s_5 and s_7 contain the same configurations, but with different look-ahead symbols. We can merge them in the following state:
    
    \[ s_{57} = \{ \]
    
    - A \rightarrow a \cdot Ab\$
    - A \rightarrow aAb\$
    - A \rightarrow B\$
    - B \rightarrow x\$

  - s_3 and s_8 can also be merged into the following state:
    
    \[ s_{38} = \{ \]
    
    - A \rightarrow Ab\$
    - B \rightarrow x\$

  - s_6 and s_11 can also be merged into the following state:
    
    \[ s_{611} = \{ \]
    
    - A \rightarrow Ab\$
    - B \rightarrow x\$

  - s_10 and s_12 can also be merged into the following state:
    
    \[ s_{1012} = \{ \]
    
    - A \rightarrow Ab\$

In summary, LALR(1) parsers can simplify LR(1) analyzers by merging states with the same configurations but different lookahead symbols, reducing the risk of collisions while still maintaining a significant level of power compared to SLR(1) parsers.
LALR Analysis

LALR(1) parsers: introductory example

- Deterministic Finite Automata with transitions for an LALR parser

```
S' → S {$}  S → A {$}  A → a A {$}  A → a B {$}  A → B {$}  B → x {$}
S → x b {$}  B → x {$}
S → S' {$}  S → A {$}  A → a A {$}  A → a B {$}  A → B {$}  B → x {$}
S → x b {$}  A → a A {$}  A → a B {$}  A → B {$}  B → x {$}
```

LALR Analysis

LALR(1)

- Shifts in the table:
  - It is the same as in LR(0)
  - They can be obtained by following the transitions in the table.
  - If the automata can go from $s_i$ to $s_j$ by means of symbol $X$, then we shall add the following action:
    $$\text{Syntactic_table}[i,X] = \begin{cases} s_j & s_i \in \Sigma_i \\ j & s_i \notin \Sigma_i \end{cases}$$

- Reductions in the table:
  - In the cells for the states which contain reduction configurations, of the form $\lambda \rightarrow \gamma \ast (\sigma_1, \ldots, \sigma_n)$ we have to add the reduction of the rule $\lambda \rightarrow \gamma$ only in the columns for their look-ahead non-terminal symbols, i.e., $\{\sigma_1, \ldots, \sigma_n\}$.
  - Therefore, this step is the same as in LR(1), once the diagram for LALR has been built using the previous procedure.

LALR Analysis

LALR(1): parsing examples

- Analysis table
  The following is an example of analysis with two strings:

  aaxbb
  ax

<table>
<thead>
<tr>
<th>$\Sigma_i$</th>
<th>$\Sigma_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>a</td>
</tr>
<tr>
<td>0</td>
<td>s57</td>
</tr>
<tr>
<td>1</td>
<td>acc</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>r4</td>
</tr>
<tr>
<td>4</td>
<td>s13</td>
</tr>
<tr>
<td>57</td>
<td>s57</td>
</tr>
<tr>
<td>611</td>
<td>s1012</td>
</tr>
<tr>
<td>9</td>
<td>r5</td>
</tr>
<tr>
<td>1012</td>
<td>r3</td>
</tr>
<tr>
<td>13</td>
<td>r2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action</th>
<th>Go-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>acc</td>
<td></td>
</tr>
<tr>
<td>r1</td>
<td></td>
</tr>
<tr>
<td>r4</td>
<td></td>
</tr>
<tr>
<td>s13</td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td></td>
</tr>
<tr>
<td>s1012</td>
<td></td>
</tr>
<tr>
<td>r3</td>
<td></td>
</tr>
<tr>
<td>r2</td>
<td></td>
</tr>
</tbody>
</table>
LALR Analysis

\{xb, a^nxb^n | n \geq 0\}

- \( S' \rightarrow S \$
- \( S \rightarrow A \)
- \( S \rightarrow xb \)
- \( A \rightarrow aAb \)
- \( A \rightarrow B \)
- \( B \rightarrow x \)

Action | Go-to
--- | ---
0 | s57
1 | s4
2 | acc
38 | r4
4 | s13
57 | s57
611 | s1012
9 | r5
1012 | r3
13 | r2

LALR Analysis

\{xb, a^nxb^n | n \geq 0\}

- \( S' \rightarrow S \$
- \( S \rightarrow A \)
- \( S \rightarrow xb \)
- \( A \rightarrow aAb \)
- \( A \rightarrow B \)
- \( B \rightarrow x \)

Action | Go-to
--- | ---
57 | a
0 | s57
1 | s4
2 | acc
38 | r4
4 | s13
57 | s57
611 | s1012
9 | r5
1012 | r3
13 | r2

LALR Analysis

\{xb, a^nxb^n | n \geq 0\}

- \( S' \rightarrow S \$
- \( S \rightarrow A \)
- \( S \rightarrow xb \)
- \( A \rightarrow aAb \)
- \( A \rightarrow B \)
- \( B \rightarrow x \)

Action | Go-to
--- | ---
9 | x
57 | s57
a
0 | s57
1 | s4
2 | acc
38 | r4
4 | s13
57 | s57
611 | s1012
9 | r5
1012 | r3
13 | r2
LALR Analysis

\[ \{xb, a^nxb^n \mid n \geq 0\} \]

<table>
<thead>
<tr>
<th>( \Sigma_r )</th>
<th>( \Sigma_a )</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>a</td>
</tr>
<tr>
<td>0</td>
<td>s57</td>
</tr>
<tr>
<td>1</td>
<td>acc</td>
</tr>
<tr>
<td>2</td>
<td>r1</td>
</tr>
<tr>
<td>38</td>
<td>r4</td>
</tr>
<tr>
<td>4</td>
<td>s13</td>
</tr>
<tr>
<td>57</td>
<td>s57</td>
</tr>
<tr>
<td>611</td>
<td>s1012</td>
</tr>
<tr>
<td>9</td>
<td>r5</td>
</tr>
<tr>
<td>1012</td>
<td>r3</td>
</tr>
<tr>
<td>13</td>
<td>r2</td>
</tr>
</tbody>
</table>

Action | Go-to

(0) \( S' \rightarrow SS \)
(1) \( S \rightarrow A \)
(2) \( S \rightarrow xb \)
(3) \( A \rightarrow aAb \)
(4) \( A \rightarrow B \)
(5) \( B \rightarrow x \)
### LALR Analysis

#### Grammar

\{xb, a^nxb^n \mid n \geq 0\}

#### LR(0) Items

<table>
<thead>
<tr>
<th>( \Sigma_T )</th>
<th>( \Sigma_N )</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>a</td>
</tr>
<tr>
<td>0</td>
<td>s57</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>611</td>
<td></td>
</tr>
</tbody>
</table>

#### LR(1) States

<table>
<thead>
<tr>
<th>( \Sigma_T )</th>
<th>( \Sigma_N )</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>a</td>
</tr>
<tr>
<td>0</td>
<td>s57</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>611</td>
<td></td>
</tr>
</tbody>
</table>

#### LR(1) Actions

- (0) \( S' \rightarrow SS \)
- (1) \( S \rightarrow A \)
- (2) \( S \rightarrow xb \)
- (3) \( A \rightarrow aAb \)
- (4) \( A \rightarrow B \)
- (5) \( B \rightarrow x \)

#### LR(1) Go-to

- \( E \rightarrow S \)
- \( a \rightarrow S \)
- \( b \rightarrow S \)
- \( x \rightarrow S \)
- \( $ \rightarrow S \)

#### LR(1) Action

- 1
- 2
- 3
- 4
- 5
### LALR Analysis

#### Grammar

\[
\{xb, a^nxb^n | n \geq 0\}
\]

#### Production Rules

1. \( S' \rightarrow S \)
2. \( S \rightarrow A \)
3. \( S \rightarrow xb \)
4. \( A \rightarrow aAb \)
5. \( A \rightarrow B \)
6. \( B \rightarrow x \)
7. \( B \rightarrow x \) where \( \{xb, anxbn | n \geq 0\} \)

#### LR(0) Tables

**Start State:** \( S' \)

#### Action Table

<table>
<thead>
<tr>
<th>( \Sigma_r )</th>
<th>( \Sigma_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>a</td>
</tr>
<tr>
<td>0</td>
<td>s57</td>
</tr>
<tr>
<td>1</td>
<td>acc</td>
</tr>
<tr>
<td>2</td>
<td>r1</td>
</tr>
<tr>
<td>38</td>
<td>r4</td>
</tr>
<tr>
<td>4</td>
<td>s13</td>
</tr>
<tr>
<td>57</td>
<td>s57</td>
</tr>
</tbody>
</table>

#### Go-to Table

<table>
<thead>
<tr>
<th></th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>r1</td>
</tr>
<tr>
<td>2</td>
<td>r4</td>
</tr>
<tr>
<td>3</td>
<td>s13</td>
</tr>
<tr>
<td>4</td>
<td>r5</td>
</tr>
<tr>
<td>5</td>
<td>s1012</td>
</tr>
<tr>
<td>6</td>
<td>r3</td>
</tr>
<tr>
<td>7</td>
<td>r2</td>
</tr>
<tr>
<td>8</td>
<td>acc</td>
</tr>
</tbody>
</table>

#### LR(0) Analysis

- **Start State:** \( S' \)
- **Actions:**
  - 0: Go-to \( S' \)
  - 1: Acc
  - 2: \( r1 \)
  - 3: \( r4 \)
  - 4: \( s13 \)
  - 5: \( r5 \)
  - 6: \( s1012 \)
  - 7: \( r3 \)
  - 8: Acc
- **Go-to States:**
  - \( S' \rightarrow S \)
  - \( S \rightarrow A \)
  - \( S \rightarrow xb \)
  - \( A \rightarrow aAb \)
  - \( A \rightarrow B \)
  - \( B \rightarrow x \)
  - \( B \rightarrow x \)
LALR Analysis

Evaluation

- Power:
  - LALR(1) is less powerful than LR(1), but more so than SLR(1).
  - However, most structures found in programming languages are LALR(1), so they can be parsed with this procedure.

- Efficiency:
  - There are less states in an LALR(1) parser than in an LR(1) parser.