1. SLR(1) parsing
2. LR(1) parsing.
Our goal: impose deterministic choices on this non-deterministic LR parsing algorithm

c := first symbol of input w$

while(true )

\alpha := the stack

if A \rightarrow \beta \cdot c \gamma \in \delta_G(q_0, \alpha )
then shift c onto the stack

\quad c := next input toke n;

if A \rightarrow \beta \cdot \in \delta_G(q_0, \alpha )
then reduce : pop \beta off the stack
and then push A onto the stack;

if S \rightarrow \beta \cdot \in \delta_G(q_0, \alpha )
then accept and exit if no more input;
if none of the above then ERROR
The easy part: NFA $\rightarrow$ DFA

In general, add new production $S' \rightarrow S$, where $S$ is the original start symbol. For the simple term grammar $G_2$, add production

$$E' \rightarrow E$$

which produces the NFA start state

$$q_0 = E' \rightarrow \bullet E$$

The DFA start state is then

$$\varepsilon - \text{closure} \left( \{ E' \rightarrow \bullet E \} \right) =$$

<table>
<thead>
<tr>
<th>Production</th>
<th>DFA Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E' \rightarrow \bullet E$</td>
<td>$E$</td>
</tr>
<tr>
<td>$E \rightarrow \bullet E + T$</td>
<td>$E$</td>
</tr>
<tr>
<td>$E \rightarrow \bullet T$</td>
<td>$T$</td>
</tr>
<tr>
<td>$T \rightarrow \bullet T \ast F$</td>
<td>$T$</td>
</tr>
<tr>
<td>$T \rightarrow \bullet F$</td>
<td>$F$</td>
</tr>
<tr>
<td>$F \rightarrow \bullet (E)$</td>
<td>$F$</td>
</tr>
<tr>
<td>$F \rightarrow \bullet \text{id}$</td>
<td>$F$</td>
</tr>
</tbody>
</table>
The DFA transition function $\delta$

For this DFA

$$\delta(I, X) = \varepsilon \text{- closure} \left( \{ A \rightarrow \alpha X \bullet \beta \mid A \rightarrow \alpha \bullet X \beta \in I \} \right)$$

Many books call this $\text{GOTO}(I, X)$.

and repeat the construction of DFA

specialised to LR(0) items (using
function called CLOSURE). I see no reason to do
this since we already know how to build a DFA
from an NFA (see Lexing lecture).
A few DFA transitions for grammar $G_2$

- $E \rightarrow T \cdot$
- $T \rightarrow T \cdot * F$
- $F \rightarrow (\cdot E)$
- $E \rightarrow \cdot E + T$
- $F \rightarrow \cdot (E)$
- $E \rightarrow \cdot T$
- $T \rightarrow \cdot T * F$
- $T \rightarrow \cdot F$
- $F \rightarrow \cdot id$
- $T \rightarrow F \cdot$
- $F \rightarrow id \cdot$
- $F \rightarrow id$
Full DFA for the stack language of $G_2$

As usual, the ERROR state and transitions to it are not included in the diagram.
How can we avoid shift/reduce conflicts?

Consider $I_2$

$$
\begin{align*}
I_2 & \\
E & \rightarrow T \bullet \\
T & \rightarrow T \bullet * F
\end{align*}
$$

This inspires one approach called SLR(1) (Simple LR(1)):

1) Shift using if $*$ is the next token.
2) Reduce with $E \rightarrow T$ only if next token is in
   \begin{align*}
   \text{FOLLOW}(E) &= \{(, +, $\}.
   \end{align*}
Now we can do a DETERMINISTIC SLR(1) parse of \((x+y)\)

1) When the stack contains \(\alpha\), the parser is in state \(\delta(I_0, \alpha)\). For example,
   \[
   \delta(I_0, \text{E + T}) = I_9
   \]
   \[
   \delta(I_0, \text{(T}^* \text{)}) = I_7
   \]
   \[
   \delta(I_0, \text{E} \ast \text{T}) = \text{ERROR}
   \]

2) When the current state is \(I\), the next token is \(c\), and \(A \rightarrow \beta \bullet c \gamma \in I\), then shift \(t\) onto stack.

3) When the current state is \(I\), the next token is \(c\), \(A \rightarrow \beta \bullet \in I\), and \(c \in \text{FOLLOW}(A)\), then reduce with production \(A \rightarrow \beta\)
Replay parsing of \((x+y)\) using SLR(1) actions
\((FW(X)\) abbreviates FOLLOW\((X)\))

<table>
<thead>
<tr>
<th>stack, input</th>
<th>State action</th>
<th>reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>$, (x + y)$</td>
<td>I_0 shift</td>
<td>$F \rightarrow \bullet(E) \in I_0$</td>
</tr>
<tr>
<td>$(, x + y)$</td>
<td>I_4 shift</td>
<td>$F \rightarrow \bullet id \in I_4$</td>
</tr>
<tr>
<td>$(x, + y)$</td>
<td>I_5 reduce</td>
<td>&quot;$+&quot; \in FW(F)$</td>
</tr>
<tr>
<td>$(F, + y)$</td>
<td>I_3 reduce</td>
<td>&quot;$+&quot; \in FW(T)$</td>
</tr>
<tr>
<td>$(T, + y)$</td>
<td>I_2 reduce</td>
<td>&quot;$+&quot; \in FW(E)$</td>
</tr>
<tr>
<td>$(E, + y)$</td>
<td>I_8 shift</td>
<td>$E \rightarrow E \bullet + T \in I_8$</td>
</tr>
<tr>
<td>$(E+, y)$</td>
<td>I_6 shift</td>
<td>$F \rightarrow \bullet id \in I_6$</td>
</tr>
<tr>
<td>stack, input</td>
<td>State</td>
<td>action</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------</td>
<td>-----------------</td>
</tr>
<tr>
<td>$(E + y, )$</td>
<td>I₅</td>
<td>reduce $F \rightarrow id$</td>
</tr>
<tr>
<td>$(E + F, )$</td>
<td>I₃</td>
<td>reduce $T \rightarrow F$</td>
</tr>
<tr>
<td>$(E + T, )$</td>
<td>I₉</td>
<td>reduce $E \rightarrow E + T$</td>
</tr>
<tr>
<td>$(E, )$</td>
<td>I₈</td>
<td>shift</td>
</tr>
<tr>
<td>$(E), $</td>
<td>I₁₁</td>
<td>reduce $F \rightarrow (E)$</td>
</tr>
<tr>
<td>$F, $</td>
<td>I₃</td>
<td>reduce $T \rightarrow F$</td>
</tr>
<tr>
<td>$T, $</td>
<td>I₂</td>
<td>reduce $F \rightarrow E$</td>
</tr>
<tr>
<td>$E, $</td>
<td>I₁</td>
<td>reduce $E' \rightarrow E$</td>
</tr>
<tr>
<td>$E', $</td>
<td></td>
<td>accept!</td>
</tr>
</tbody>
</table>
Better idea: Replace the stack contents with state numbers!

(  
(id  
(F  
(T  
(E  
(E +  

0  
04  
045  
043  
042  
048  
0486  

(E + id  
(E + F  
(E + T  
(E  
(F  
(T  
(E  

04865  
04863  
04869  
048  
04 11  
03  
02  
01)
LR parsing with DFA states on the stack

\[ a := \text{first symbol of input } w\\ \]
\[
\text{while(true )}
\]
\[
s := \text{state at top of stack}
\]
\[
\text{if } \text{ACTION}[s, a] = \text{shift } t
\]
\[
\text{then push } t \text{ on stack}
\]
\[
a := \text{next input toke } n
\]
\[
\text{else if } \text{ACTION}[s, a] = \text{reduce } A \rightarrow \beta
\]
\[
\text{then pop } |\beta| \text{ states off the stack}
\]
\[
t := \text{state at top of stack}
\]
\[
\text{push } \text{GOTO}[t, A] \text{ onto the stack}
\]
\[
\text{else if } \text{ACTION}[s, a] = \text{accept}
\]
\[
\text{then accept and exit}
\]
\[
\text{else ERROR}
\]
ACTION and GOTO for SLR(1)

If \([A \rightarrow \alpha \bullet a\beta] \in I_i\) and \(\delta(I_i, a) = I_j\) then \(\text{ACTION}[i, a] = \text{shift} \ j\)

If \([A \rightarrow \alpha \bullet] \in I_i\) and \(A \neq S'\)
then for all \(a \in \text{FOLLOW}(A)\),
\[\text{ACTION}[i, a] = \text{reduce} A \rightarrow \alpha\]

If \([S' \rightarrow S \bullet] \in I_i\) then \(\text{ACTION}[i, $] = \text{accept}\)

If \(\delta(I_i, A) = I_j\) then \(\text{GOTO}[i, A] = j\)

(Now do you see why I prefer to use \(\delta\) rather than \(\text{GOTO}()\)?)

Note: there may still be shift/reduce or reduce/reduce conflicts!


<table>
<thead>
<tr>
<th>State</th>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>id</td>
<td>s5</td>
</tr>
<tr>
<td>0</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>s5</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>r6</td>
<td>r6</td>
</tr>
<tr>
<td>10</td>
<td>s6</td>
<td>s11</td>
</tr>
<tr>
<td>11</td>
<td>s5</td>
<td>s5</td>
</tr>
</tbody>
</table>
## Example parse

<table>
<thead>
<tr>
<th></th>
<th>STACK</th>
<th>SYMBOLS</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>id * id + id $</td>
<td>shift</td>
</tr>
<tr>
<td>2</td>
<td>0 5</td>
<td>id</td>
<td>* id + id $</td>
<td>reduce by $F \to id$</td>
</tr>
<tr>
<td>3</td>
<td>0 3</td>
<td>$F$</td>
<td>* id + id $</td>
<td>reduce by $T \to F$</td>
</tr>
<tr>
<td>4</td>
<td>0 2</td>
<td>$T$</td>
<td>* id + id $</td>
<td>shift</td>
</tr>
<tr>
<td>5</td>
<td>0 2 7</td>
<td>$T*$</td>
<td>id + id $</td>
<td>shift</td>
</tr>
<tr>
<td>6</td>
<td>0 2 7 5</td>
<td>$T*id$</td>
<td>+ id $</td>
<td>reduce by $F \to id$</td>
</tr>
<tr>
<td>7</td>
<td>0 2 7 10</td>
<td>$T*F$</td>
<td>+ id $</td>
<td>reduce by $T \to T*F$</td>
</tr>
<tr>
<td>8</td>
<td>0 2</td>
<td>$T$</td>
<td>+ id $</td>
<td>reduce by $E \to T$</td>
</tr>
<tr>
<td>9</td>
<td>0 1</td>
<td>$E$</td>
<td>+ id $</td>
<td>shift</td>
</tr>
<tr>
<td>10</td>
<td>0 1 6</td>
<td>$E+$</td>
<td>id $</td>
<td>shift</td>
</tr>
<tr>
<td>11</td>
<td>0 1 6 5</td>
<td>$E+id$</td>
<td>$</td>
<td>reduce by $F \to id$</td>
</tr>
<tr>
<td>12</td>
<td>0 1 6 3</td>
<td>$E+F$</td>
<td>$</td>
<td>reduce by $T \to F$</td>
</tr>
<tr>
<td>13</td>
<td>0 1 6 9</td>
<td>$E+T$</td>
<td>$</td>
<td>reduce by $E \to E+T$</td>
</tr>
<tr>
<td>14</td>
<td>0 1</td>
<td>$E$</td>
<td>$</td>
<td>accept</td>
</tr>
</tbody>
</table>
Beyond SLR(1)?

\[ G_3 = (N_3, T_3, P_3, S') \]

\[ N_3 = \{ S', S, L, R \} \]

\[ T_3 = \{ *, =, \text{id} \} \]

\[ P_3 : S' \rightarrow S$ \]

\[ S \rightarrow L = R \mid R \]

\[ L \rightarrow *R \mid \text{id} \]

\[ R \rightarrow L \]
LR(0) DFA for grammar $G_3$

In state 4 there is a shift/reduce conflict between $S \rightarrow L \bullet = R$ and $R \rightarrow L \bullet$
SLR(1) cannot resolve this conflict.

\[ [S \to L\bullet = R] \in I_4 \text{ so } \delta(I_4, "=") = I_6 \]
and so \( \text{ACTION}[4,"="] = \text{shift } 6 \)

However, \([R \to L\bullet] \in I_4\)
and "=" \(\in FOLLOW(R) = \{"=",\$\}\),
so \( \text{ACTION}[4,"="] = \text{reduce } R \to L \)
Beyond SLR(1)?  LR(1)!

Problems: with SLR(1) there may be shift - reduce or reduce - reduce conflicts when ACTION and GOTO are not uniquely defined.

Either fix the grammar or use a more powerful technique.

LR(1) parsing starts with items of the form

\[ A \rightarrow \alpha \bullet \beta, a \]

where a is an explicit look - ahead token.
Define an NFA with $LR(1)$ items as states

For each $b \in \text{FIRST}(\beta a)$:

$$A \rightarrow \alpha \cdot B\beta, a \xrightarrow{\epsilon} B \rightarrow \cdot \gamma, b$$
LR(1) DFA for grammar $G_3$

No ambiguity. Reduce $R \rightarrow L$ only if next token is $. Otherwise shift if next token is $=$.
ACTION and GOTO for LR(1)

If \([A \rightarrow \alpha \cdot a\beta, a] \in I_i \) and \(\delta(I_i, a) = I_j\) then \(\text{ACTION}[i, a] = \text{shift } j\)

If \([A \rightarrow \alpha \cdot, b] \in I_i \) and \(A \neq S'\), then
\[
\text{ACTION}[i, b] = \text{reduce } A \rightarrow \alpha
\]

If \([S' \rightarrow S \cdot, \$] \in I_i \) then \(\text{ACTION}[i, \$] = \text{accept}\)

If \(\delta(I_i, A) = I_j\) then \(\text{GOTO}[i, A] = j\)
SLR(1) vs LR(1)

SLR(1):
If \([A \rightarrow \alpha \bullet] \in I_i \) and \(A \neq S'\)
then for all \(a \in \text{FOLLOW}(A)\),
\[
\text{ACTION}[i,a] = \text{reduce } A \rightarrow \alpha
\]

LR(1):
If \([A \rightarrow \alpha \bullet, b] \in I_i \) and \(A \neq S'\), then
\[
\text{ACTION}[i,b] = \text{reduce } A \rightarrow \alpha
\]

Note that the look-ahead symbol \(b\) is used ONLY for reductions, not for shifts.
1. LR(1) is more powerful than SLR(1)
2. The DFA associated with a LR(1) parser may have a very large number of states
3. This inspired an optimisation (collapsing states) resulting in the class of LALR papers normally implemented as YACC. These parsers have fewer states but can produce very strange error messages.
4. Ocaml’s Menhir is based on LR(1) and claims to overcome many YACC problems.
5. We will not cover LALR parsing.