# Compiler Construction

## Lecture 5: Foundations of LR parsing

Jeremy Yallop
jeremy.yallop@cl.cam.ac.uk

Lent 2026

# Derivations

# Recap: example grammars



**$G_1$**

$E \to E + E \mid E * E \mid (E) \mid \mathsf{id}$

eliminate ambiguity

**$G_2$**

$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \mathsf{id}$

eliminate left recursion

**$G_3$**

$E \to T\,E'$
$E' \to +\,T\,E' \mid \epsilon$
$T \to F\,T'$
$T' \to *\,F\,T' \mid \epsilon$
$F \to (\,E\,) \mid \mathsf{id}$

add $S$

add $S$, \$

add $S$, \$

**$G_1'$**

$S \to E$
$E \to E + E \mid E * E \mid (E) \mid \mathsf{id}$

eliminate ambiguity

**$G_2'$**

$S \to E\,\$$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \mathsf{id}$

eliminate left recursion

**$G_3'$**

$S \to E\,\$$
$E \to T\,E'$
$E' \to +\,T\,E' \mid \epsilon$
$T \to F\,T'$
$T' \to *\,F\,T' \mid \epsilon$
$F \to (\,E\,) \mid \mathsf{id}$

**Today's running example**

# Leftmost vs rightmost derivations

**Leftmost** derivation step:

$$wA\alpha \Rightarrow_{lm} w\beta\alpha$$

(basis of top-down (L**L**) parsing)

**Rightmost** derivation step:

$$\alpha Aw \Rightarrow_{rm} \alpha\beta w$$

(basis of bottom-up (L**R**) parsing)

where

$$w \in T*$$
$$\alpha, \beta \in (N \cup T)^*$$
$$A \to \beta \in P$$

# *Bottom-up* parsers perform the derivation in reverse



Rightmost derivation

$$
\begin{aligned}
S &\Rightarrow_{rm} \mathbf{E} \\
&\Rightarrow_{rm} \mathbf{T} \\
&\Rightarrow_{rm} \mathbf{F} \\
&\Rightarrow_{rm} (\mathbf{E}) \\
&\Rightarrow_{rm} (E + \mathbf{T}) \\
&\Rightarrow_{rm} (E + \mathbf{F}) \\
&\Rightarrow_{rm} (\mathbf{E} + y) \\
&\Rightarrow_{rm} (\mathbf{T} + y) \\
&\Rightarrow_{rm} (\mathbf{F} + y) \\
&\Rightarrow_{rm} (x + y)
\end{aligned}
$$

flip!

**Reversed** rightmost derivation

$$
\begin{aligned}
(x + y) &\Leftarrow \\
(\mathbf{F} + y) &\Leftarrow \\
(\mathbf{T} + y) &\Leftarrow \\
(\mathbf{E} + y) &\Leftarrow \\
(E + \mathbf{F}) &\Leftarrow \\
(E + \mathbf{T}) &\Leftarrow \\
(\mathbf{E}) &\Leftarrow \\
\mathbf{F} &\Leftarrow \\
\mathbf{T} &\Leftarrow \\
\mathbf{E} &\Leftarrow S
\end{aligned}
$$

parsing direction

# Backwards derivation $\rightsquigarrow$ stack machine execution



$G_2'$

$S \to E\,\$$ $\quad$ $T \to T * F \mid F$

$E \to E + T \mid T$ $\quad$ $F \to (E) \mid \mathrm{id}$

$$
\begin{aligned}
(x+y) &\Leftarrow \\
(F+y) &\Leftarrow \\
(T+y) &\Leftarrow \\
(E+y) &\Leftarrow \\
(E+F) &\Leftarrow \\
(E+T) &\Leftarrow \\
(E) &\Leftarrow \\
F &\Leftarrow \\
T &\Leftarrow \\
E &\Leftarrow S
\end{aligned}
$$

View reversed derivation as a stack machine $\longrightarrow$

| stack | input |
|---|---|

# Backwards derivation $\rightsquigarrow$ stack machine execution



$G_2'$

$S \to E\,\$$ $\quad$ $T \to T * F \mid F$

$E \to E + T \mid T$ $\quad$ $F \to (E) \mid \mathrm{id}$

$$
\begin{aligned}
(x+y) &\Leftarrow \\
(F+y) &\Leftarrow \\
(T+y) &\Leftarrow \\
(E+y) &\Leftarrow \\
(E+F) &\Leftarrow \\
(E+T) &\Leftarrow \\
(E) &\Leftarrow \\
F &\Leftarrow \\
T &\Leftarrow \\
E &\Leftarrow S
\end{aligned}
$$

View reversed derivation as a stack machine $\longrightarrow$

| stack | input |
|---|---|
| $\$$ | $(x+y)\$$ |

# Backwards derivation $\rightsquigarrow$ stack machine execution



$G_2'$

| | |
|---|---|
| $S \to E\,\$$ | $T \to T * F \mid F$ |
| $E \to E + T \mid T$ | $F \to (E) \mid \mathrm{id}$ |

$$
\begin{aligned}
(x+y) &\Leftarrow \\
(F+y) &\Leftarrow \\
(T+y) &\Leftarrow \\
(E+y) &\Leftarrow \\
(E+F) &\Leftarrow \\
(E+T) &\Leftarrow \\
(E) &\Leftarrow \\
F &\Leftarrow \\
T &\Leftarrow \\
E &\Leftarrow S
\end{aligned}
$$

View reversed derivation as a stack machine $\longrightarrow$

| stack | input |
|---|---|
| $\$$ | $(x+y)\$$ |
| $\$(F$ | $+y)\$$ |

# Backwards derivation $\rightsquigarrow$ stack machine execution



$G_2'$

| | |
|---|---|
| $S \to E\,\$$ | $T \to T * F \mid F$ |
| $E \to E + T \mid T$ | $F \to (E) \mid \mathrm{id}$ |

$$
\begin{array}{rcl}
(x+y) & \Leftarrow & \\
(F+y) & \Leftarrow & \\
(T+y) & \Leftarrow & \\
(E+y) & \Leftarrow & \\
(E+F) & \Leftarrow & \\
(E+T) & \Leftarrow & \\
(E) & \Leftarrow & \\
F & \Leftarrow & \\
T & \Leftarrow & \\
E & \Leftarrow & S
\end{array}
$$

View reversed derivation as a stack machine $\longrightarrow$

| stack | input |
|---|---|
| $\$$ | $(x+y)\$$ |
| $\$(F$ | $+y)\$$ |
| $\$(T$ | $+y)\$$ |

# Backwards derivation $\rightsquigarrow$ stack machine execution



$G_2'$

$S \to E\,\$$ $\quad$ $T \to T * F \mid F$

$E \to E + T \mid T$ $\quad$ $F \to (E) \mid \mathrm{id}$

View reversed derivation as a stack machine

$$
\begin{aligned}
(x+y) &\Leftarrow \\
(F+y) &\Leftarrow \\
(T+y) &\Leftarrow \\
(E+y) &\Leftarrow \\
(E+F) &\Leftarrow \\
(E+T) &\Leftarrow \\
(E) &\Leftarrow \\
F &\Leftarrow \\
T &\Leftarrow \\
E &\Leftarrow S
\end{aligned}
$$

$\longrightarrow$

| stack | input |
|---|---|
| $\$$ | $(x+y)\$$ |
| $\$(F$ | $+y)\$$ |
| $\$(T$ | $+y)\$$ |
| $\$(E$ | $+y)\$$ |

# Backwards derivation $\rightsquigarrow$ stack machine execution



$G_2'$

$S \to E\,\$$ $\quad$ $T \to T * F \mid F$

$E \to E + T \mid T$ $\quad$ $F \to (E) \mid \mathrm{id}$

View reversed derivation as a stack machine

$$
\begin{aligned}
(x+y) &\Leftarrow \\
(F+y) &\Leftarrow \\
(T+y) &\Leftarrow \\
(E+y) &\Leftarrow \\
(E+F) &\Leftarrow \\
(E+T) &\Leftarrow \\
(E) &\Leftarrow \\
F &\Leftarrow \\
T &\Leftarrow \\
E &\Leftarrow S
\end{aligned}
$$

$\longrightarrow$

| stack | input |
|---|---|
| $\$$ | $(x+y)\$$ |
| $\$(F$ | $+y)\$$ |
| $\$(T$ | $+y)\$$ |
| $\$(E$ | $+y)\$$ |
| $\$(E+F$ | $)\$$ |

# Backwards derivation $\rightsquigarrow$ stack machine execution



$G_2'$

$S \to E\,\$$ $\quad$ $T \to T * F \mid F$

$E \to E + T \mid T$ $\quad$ $F \to (E) \mid \mathrm{id}$

$$
\begin{aligned}
(x+y) &\Leftarrow \\
(F+y) &\Leftarrow \\
(T+y) &\Leftarrow \\
(E+y) &\Leftarrow \\
(E+F) &\Leftarrow \\
(E+T) &\Leftarrow \\
(E) &\Leftarrow \\
F &\Leftarrow \\
T &\Leftarrow \\
E &\Leftarrow S
\end{aligned}
$$

View reversed derivation as a stack machine $\longrightarrow$

| stack | input |
|---|---|
| $\$$ | $(x+y)\$$ |
| $\$(F$ | $+y)\$$ |
| $\$(T$ | $+y)\$$ |
| $\$(E$ | $+y)\$$ |
| $\$(E+F$ | $)\$$ |
| $\$(E+T$ | $)\$$ |

# Backwards derivation $\rightsquigarrow$ stack machine execution



$G_2'$

$S \to E\,\$$ $\quad$ $T \to T * F \mid F$

$E \to E + T \mid T$ $\quad$ $F \to (E) \mid \mathrm{id}$

$$
\begin{aligned}
(x+y) &\Leftarrow \\
(F+y) &\Leftarrow \\
(T+y) &\Leftarrow \\
(E+y) &\Leftarrow \\
(E+F) &\Leftarrow \\
(E+T) &\Leftarrow \\
(E) &\Leftarrow \\
F &\Leftarrow \\
T &\Leftarrow \\
E &\Leftarrow S
\end{aligned}
$$

View reversed derivation as a stack machine $\longrightarrow$

| stack | input |
|---|---|
| $\$$ | $(x+y)\$$ |
| $\$(F$ | $+y)\$$ |
| $\$(T$ | $+y)\$$ |
| $\$(E$ | $+y)\$$ |
| $\$(E+F$ | $)\$$ |
| $\$(E+T$ | $)\$$ |
| $\$(E)$ | $\$$ |

# Backwards derivation $\rightsquigarrow$ stack machine execution



$G_2'$

$S \to E\,\$$ $\quad$ $T \to T * F \mid F$

$E \to E + T \mid T$ $\quad$ $F \to (E) \mid \mathrm{id}$

$$
\begin{aligned}
(x+y) &\Leftarrow \\
(F+y) &\Leftarrow \\
(T+y) &\Leftarrow \\
(E+y) &\Leftarrow \\
(E+F) &\Leftarrow \\
(E+T) &\Leftarrow \\
(E) &\Leftarrow \\
F &\Leftarrow \\
T &\Leftarrow \\
E &\Leftarrow S
\end{aligned}
$$

View reversed derivation as a stack machine $\longrightarrow$

| stack | input |
|---|---|
| $\$$ | $(x+y)\$$ |
| $\$(F$ | $+y)\$$ |
| $\$(T$ | $+y)\$$ |
| $\$(E$ | $+y)\$$ |
| $\$(E+F$ | $)\$$ |
| $\$(E+T$ | $)\$$ |
| $\$(E)$ | $\$$ |
| $\$F$ | $\$$ |

# Backwards derivation $\rightsquigarrow$ stack machine execution



$G_2'$

$S \to E\,\$$ $\quad$ $T \to T * F \mid F$

$E \to E + T \mid T$ $\quad$ $F \to (E) \mid \mathrm{id}$

$$
\begin{aligned}
(x+y) &\Leftarrow \\
(F+y) &\Leftarrow \\
(T+y) &\Leftarrow \\
(E+y) &\Leftarrow \\
(E+F) &\Leftarrow \\
(E+T) &\Leftarrow \\
(E) &\Leftarrow \\
F &\Leftarrow \\
T &\Leftarrow \\
E &\Leftarrow S
\end{aligned}
$$

View reversed derivation as a stack machine $\longrightarrow$

| stack | input |
|---|---|
| $\$$ | $(x+y)\$$ |
| $\$(F$ | $+y)\$$ |
| $\$(T$ | $+y)\$$ |
| $\$(E$ | $+y)\$$ |
| $\$(E+F$ | $)\$$ |
| $\$(E+T$ | $)\$$ |
| $\$(E)$ | $\$$ |
| $\$F$ | $\$$ |
| $\$T$ | $\$$ |

# Backwards derivation $\rightsquigarrow$ stack machine execution



$G_2'$

| | |
|---|---|
| $S \to E\,\$$ | $T \to T * F \mid F$ |
| $E \to E + T \mid T$ | $F \to (E) \mid \mathrm{id}$ |

$$
\begin{aligned}
(x+y) &\Leftarrow \\
(F+y) &\Leftarrow \\
(T+y) &\Leftarrow \\
(E+y) &\Leftarrow \\
(E+F) &\Leftarrow \\
(E+T) &\Leftarrow \\
(E) &\Leftarrow \\
F &\Leftarrow \\
T &\Leftarrow \\
E &\Leftarrow S
\end{aligned}
$$

View reversed derivation as a stack machine $\longrightarrow$

| stack | input |
|---|---|
| $\$$ | $(x+y)\$$ |
| $\$(F$ | $+y)\$$ |
| $\$(T$ | $+y)\$$ |
| $\$(E$ | $+y)\$$ |
| $\$(E+F$ | $)\$$ |
| $\$(E+T$ | $)\$$ |
| $\$(E)$ | $\$$ |
| $\$F$ | $\$$ |
| $\$T$ | $\$$ |
| $\$E$ | $\$$ |

# Backwards derivation $\rightsquigarrow$ stack machine execution



$G_2'$

$S \to E\,\$$ $\quad$ $T \to T * F \mid F$

$E \to E + T \mid T$ $\quad$ $F \to (E) \mid \mathrm{id}$

$$
\begin{aligned}
(x+y) &\Leftarrow \\
(F+y) &\Leftarrow \\
(T+y) &\Leftarrow \\
(E+y) &\Leftarrow \\
(E+F) &\Leftarrow \\
(E+T) &\Leftarrow \\
(E) &\Leftarrow \\
F &\Leftarrow \\
T &\Leftarrow \\
E &\Leftarrow S
\end{aligned}
$$

View reversed derivation as a stack machine $\longrightarrow$

| stack | input |
|---|---|
| $\$$ | $(x+y)\$$ |
| $\$(F$ | $+y)\$$ |
| $\$(T$ | $+y)\$$ |
| $\$(E$ | $+y)\$$ |
| $\$(E+F$ | $)\$$ |
| $\$(E+T$ | $)\$$ |
| $\$(E)$ | $\$$ |
| $\$F$ | $\$$ |
| $\$T$ | $\$$ |
| $\$E$ | $\$$ |
| $\$S$ | $\$$ |

# Formalisation

# LR parser configurations

An **LR parser configuration** has the form

$$\$\alpha, w\$$$

stack $\alpha$

remaining input $w$

The configuration is **valid** when there exists a rightmost derivation of the form

$$S \Rightarrow_{rm}^{*} \alpha\ w$$

(NB: stacks now grow *rightwards*.)

# LR parser reductions

There may be **lots of possible steps** from each configuration.

Suppose:

$$\alpha A w \Rightarrow_{rm} \alpha \beta B z w$$

One possible step between configurations replaces $\beta B z$ with $A$ on top of the stack:

$$\$\alpha\beta Bz, w\$ \xrightarrow[A \to \beta Bz]{\text{reduce}} \$\alpha A, w\$$$

This action is called a **reduction** using production $A \to \beta Bz$.

# Reductions are not sufficient

Suppose we have the derivation:

$$\begin{aligned} & \alpha A w \\ \Rightarrow_{rm} \quad & \alpha\beta B z w \quad (\text{using } A \to \beta B z) \\ \Rightarrow_{rm} \quad & \alpha\beta\gamma z w \quad (\text{using } B \to \gamma) \end{aligned}$$

The reverse simulation gets stuck:

$$\begin{aligned} & \$\alpha\beta\gamma, zw\$ \\ \xrightarrow[B\to\gamma]{\text{reduce}} \quad & \$\alpha\beta B, zw\$ \\ \xrightarrow[???]{} \quad & ??? \end{aligned}$$

We have $\beta B$ on top of the stack, but we want $\beta B z$ on top of the stack.

# We also need *shift*

A **shift** action shifts a terminal onto the stack.

$$
\begin{aligned}
& \alpha A w \\
\Rightarrow_{rm} \quad & \alpha\beta B z w \quad (\text{using } A \to \beta B z) \\
\Rightarrow_{rm} \quad & \alpha\beta\gamma z w \quad (\text{using } B \to \gamma)
\end{aligned}
$$

$$
\begin{aligned}
& \$\alpha\beta\gamma, zw\$ \\
\xrightarrow[B\to\gamma]{\text{reduce}} \quad & \$\alpha\beta B, zw\$ \\
\xrightarrow[z]{\textbf{shift}} \quad & \$\alpha\beta Bz, w\$ \\
\xrightarrow[A\to\beta Bz]{\text{reduce}} \quad & \$\alpha A, w\$
\end{aligned}
$$

Q: *How do we know when to stop shifting?*
(e.g. here we don't want to shift w)

# Another replay

Derivation

$$
\begin{array}{lll}
 & \alpha B w A z & \\
\Rightarrow_{rm} & \alpha B w y z & (\text{using } A \to y) \\
\Rightarrow_{rm} & \alpha \gamma w y z & (\text{using } B \to \gamma)
\end{array}
$$

Our parser's possible actions:

$$
\begin{array}{ll}
 & \$\alpha\gamma, wyz\$ \\
\xrightarrow[B\to\gamma]{\text{reduce}} & \$\alpha B, wyz\$ \\
\xrightarrow[w]{\text{shift}} & \$\alpha Bw, yz\$ \\
\xrightarrow[y]{\text{shift}} & \$\alpha Bwy, z\$ \\
\xrightarrow[A\to y]{\text{reduce}} & \$\alpha BwA, z\$
\end{array}
$$

Again: *how do we know when to reduce and when to stop shifting?*

# Shift & reduce

# Shift and reduce are sufficient

It appears that if we have a derivation

$$S \Rightarrow_{rm}^{*} w$$

we can always "replay" it in reverse using shift/reduce actions:

$$\$, w\$ \rightarrow^{*} \$S, \$$$

i.e. **shift and reduce are sufficient**.

However, we have used the desired derivation to guide the "replay".
When parsing there is no derivation available in advance.
So **our parser is non-deterministic**: it must *guess* what the future holds.

# Replay parsing of $(x + y)$ using shift/reduce actions



$S \to E\,\$$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \text{id}$

top of stack $X$ next input token $a$

| stack | input | action$[X, a]$ |
|---|---|---|

# Replay parsing of $(x + y)$ using shift/reduce actions



$S \to E\,\$$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \text{id}$

top of stack $X$ — next input token $a$

| stack | input | action$[X, a]$ |
|---|---|---|
| $ | $(x + y)$$ | shift ( |

# Replay parsing of $(x + y)$ using shift/reduce actions



$S \to E\,\$$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \text{id}$

top of stack $X$ — next input token $a$

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$$ | $(x + y)\$$ | shift ( |
| $\$($ | $x + y)\$$ | shift $x$ |

# Replay parsing of $(x + y)$ using shift/reduce actions



$S \to E\,\$$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \text{id}$

top of stack $X$ — next input token $a$

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$$ | $(x + y)\$$ | shift ( |
| $\$($ | $x + y)\$$ | shift $x$ |
| $\$(x$ | $+y)\$$ | reduce $F \to id$ |

# Replay parsing of $(x + y)$ using shift/reduce actions



$S \to E\,\$$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \text{id}$

top of stack $X$ — next input token $a$

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$$ | $(x + y)\$$ | shift ( |
| $\$($ | $x + y)\$$ | shift $x$ |
| $\$(x$ | $+y)\$$ | reduce $F \to id$ |
| $\$(F$ | $+y)\$$ | reduce $T \to F$ |

# Replay parsing of $(x + y)$ using shift/reduce actions



$S \to E\,\$$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \text{id}$

top of stack $X$ — next input token $a$

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$$ | $(x + y)\$$ | shift ( |
| $\$($ | $x + y)\$$ | shift $x$ |
| $\$(x$ | $+y)\$$ | reduce $F \to id$ |
| $\$(F$ | $+y)\$$ | reduce $T \to F$ |
| $\$(T$ | $+y)\$$ | reduce $E \to T$ |

# Replay parsing of $(x + y)$ using shift/reduce actions



$S \to E\,\$$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \text{id}$

top of stack $X$ — next input token $a$

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$$ | $(x + y)\$$ | shift ( |
| $\$($ | $x + y)\$$ | shift $x$ |
| $\$(x$ | $+y)\$$ | reduce $F \to id$ |
| $\$(F$ | $+y)\$$ | reduce $T \to F$ |
| $\$(T$ | $+y)\$$ | reduce $E \to T$ |
| $\$(E$ | $+y)\$$ | shift $+$ |

# Replay parsing of $(x + y)$ using shift/reduce actions



$S \to E\,\$$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \text{id}$

top of stack $X$ — next input token $a$

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$$ | $(x + y)\$$ | shift $($ |
| $\$($ | $x + y)\$$ | shift $x$ |
| $\$(x$ | $+y)\$$ | reduce $F \to id$ |
| $\$(F$ | $+y)\$$ | reduce $T \to F$ |
| $\$(T$ | $+y)\$$ | reduce $E \to T$ |
| $\$(E$ | $+y)\$$ | shift $+$ |
| $\$(E+$ | $y)\$$ | shift $y$ |

# Replay parsing of $(x + y)$ using shift/reduce actions



$S \to E\,\$$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \text{id}$

top of stack $X$ — next input token $a$

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$$ | $(x + y)\$$ | shift $($ |
| $\$($ | $x + y)\$$ | shift $x$ |
| $\$(x$ | $+y)\$$ | reduce $F \to id$ |
| $\$(F$ | $+y)\$$ | reduce $T \to F$ |
| $\$(T$ | $+y)\$$ | reduce $E \to T$ |
| $\$(E$ | $+y)\$$ | shift $+$ |
| $\$(E+$ | $y)\$$ | shift $y$ |
| $\$(E + y$ | $)\$$ | reduce $F \to id$ |

# Replay parsing of $(x + y)$ using shift/reduce actions

$S \to E\,\$$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \text{id}$

top of stack $X$ — next input token $a$

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$$ | $(x+y)\$$ | shift $($ |
| $\$($ | $x+y)\$$ | shift $x$ |
| $\$(x$ | $+y)\$$ | reduce $F \to id$ |
| $\$(F$ | $+y)\$$ | reduce $T \to F$ |
| $\$(T$ | $+y)\$$ | reduce $E \to T$ |
| $\$(E$ | $+y)\$$ | shift $+$ |
| $\$(E+$ | $y)\$$ | shift $y$ |
| $\$(E+y$ | $)\$$ | reduce $F \to id$ |

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$(E+F$ | $)\$$ | reduce $T \to F$ |

# Replay parsing of $(x + y)$ using shift/reduce actions

$S \to E\,\$$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \text{id}$

top of stack $X$ — next input token $a$

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$$ | $(x+y)\$$ | shift $($ |
| $\$($ | $x+y)\$$ | shift $x$ |
| $\$(x$ | $+y)\$$ | reduce $F \to id$ |
| $\$(F$ | $+y)\$$ | reduce $T \to F$ |
| $\$(T$ | $+y)\$$ | reduce $E \to T$ |
| $\$(E$ | $+y)\$$ | shift $+$ |
| $\$(E+$ | $y)\$$ | shift $y$ |
| $\$(E+y$ | $)\$$ | reduce $F \to id$ |

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$(E+F$ | $)\$$ | reduce $T \to F$ |
| $\$(E+T$ | $)\$$ | reduce $E \to E+T$ |

# Replay parsing of $(x + y)$ using shift/reduce actions



$S \to E\,\$$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \text{id}$

top of stack $X$ — next input token $a$

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$$ | $(x+y)\$$ | shift $($ |
| $\$($ | $x+y)\$$ | shift $x$ |
| $\$(x$ | $+y)\$$ | reduce $F \to id$ |
| $\$(F$ | $+y)\$$ | reduce $T \to F$ |
| $\$(T$ | $+y)\$$ | reduce $E \to T$ |
| $\$(E$ | $+y)\$$ | shift $+$ |
| $\$(E+$ | $y)\$$ | shift $y$ |
| $\$(E+y$ | $)\$$ | reduce $F \to id$ |

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$(E+F$ | $)\$$ | reduce $T \to F$ |
| $\$(E+T$ | $)\$$ | reduce $E \to E+T$ |
| $\$(E$ | $)\$$ | shift $)$ |

# Replay parsing of $(x + y)$ using shift/reduce actions



$S \to E\,\$$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \text{id}$

top of stack $X$ — next input token $a$

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$$ | $(x+y)\$$ | shift $($ |
| $\$($ | $x+y)\$$ | shift $x$ |
| $\$(x$ | $+y)\$$ | reduce $F \to id$ |
| $\$(F$ | $+y)\$$ | reduce $T \to F$ |
| $\$(T$ | $+y)\$$ | reduce $E \to T$ |
| $\$(E$ | $+y)\$$ | shift $+$ |
| $\$(E+$ | $y)\$$ | shift $y$ |
| $\$(E+y$ | $)\$$ | reduce $F \to id$ |

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$(E+F$ | $)\$$ | reduce $T \to F$ |
| $\$(E+T$ | $)\$$ | reduce $E \to E+T$ |
| $\$(E$ | $)\$$ | shift $)$ |
| $\$(E)$ | $\$$ | reduce $F \to (E)$ |

# Replay parsing of $(x + y)$ using shift/reduce actions



$S \to E\,\$$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \text{id}$

top of stack $X$ — next input token $a$

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$$ | $(x+y)\$$ | shift ( |
| $\$($ | $x+y)\$$ | shift $x$ |
| $\$(x$ | $+y)\$$ | reduce $F \to id$ |
| $\$(F$ | $+y)\$$ | reduce $T \to F$ |
| $\$(T$ | $+y)\$$ | reduce $E \to T$ |
| $\$(E$ | $+y)\$$ | shift $+$ |
| $\$(E+$ | $y)\$$ | shift $y$ |
| $\$(E+y$ | $)\$$ | reduce $F \to id$ |

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$(E+F$ | $)\$$ | reduce $T \to F$ |
| $\$(E+T$ | $)\$$ | reduce $E \to E+T$ |
| $\$(E$ | $)\$$ | shift ) |
| $\$(E)$ | $\$$ | reduce $F \to (E)$ |
| $\$F$ | $\$$ | reduce $T \to F$ |

# Replay parsing of $(x + y)$ using shift/reduce actions



$S \to E\,\$$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \text{id}$

top of stack $X$ — next input token $a$

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$$ | $(x+y)\$$ | shift $($ |
| $\$($ | $x+y)\$$ | shift $x$ |
| $\$(x$ | $+y)\$$ | reduce $F \to id$ |
| $\$(F$ | $+y)\$$ | reduce $T \to F$ |
| $\$(T$ | $+y)\$$ | reduce $E \to T$ |
| $\$(E$ | $+y)\$$ | shift $+$ |
| $\$(E+$ | $y)\$$ | shift $y$ |
| $\$(E+y$ | $)\$$ | reduce $F \to id$ |

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$(E+F$ | $)\$$ | reduce $T \to F$ |
| $\$(E+T$ | $)\$$ | reduce $E \to E+T$ |
| $\$(E$ | $)\$$ | shift $)$ |
| $\$(E)$ | $\$$ | reduce $F \to (E)$ |
| $\$F$ | $\$$ | reduce $T \to F$ |
| $\$T$ | $\$$ | reduce $E \to T$ |

# Replay parsing of $(x + y)$ using shift/reduce actions



$S \to E\,\$$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \text{id}$

top of stack $X$ — next input token $a$

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$$ | $(x+y)\$$ | shift ( |
| $\$($ | $x+y)\$$ | shift $x$ |
| $\$(x$ | $+y)\$$ | reduce $F \to id$ |
| $\$(F$ | $+y)\$$ | reduce $T \to F$ |
| $\$(T$ | $+y)\$$ | reduce $E \to T$ |
| $\$(E$ | $+y)\$$ | shift $+$ |
| $\$(E+$ | $y)\$$ | shift $y$ |
| $\$(E+y$ | $)\$$ | reduce $F \to id$ |

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$(E+F$ | $)\$$ | reduce $T \to F$ |
| $\$(E+T$ | $)\$$ | reduce $E \to E+T$ |
| $\$(E$ | $)\$$ | shift ) |
| $\$(E)$ | $\$$ | reduce $F \to (E)$ |
| $\$F$ | $\$$ | reduce $T \to F$ |
| $\$T$ | $\$$ | reduce $E \to T$ |
| $\$E$ | $\$$ | reduce $S \to E$ |

# Replay parsing of $(x + y)$ using shift/reduce actions



$S \to E\,\$$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \text{id}$

top of stack $X$ — next input token $a$

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$$ | $(x+y)\$$ | shift ( |
| $\$($ | $x+y)\$$ | shift $x$ |
| $\$(x$ | $+y)\$$ | reduce $F \to id$ |
| $\$(F$ | $+y)\$$ | reduce $T \to F$ |
| $\$(T$ | $+y)\$$ | reduce $E \to T$ |
| $\$(E$ | $+y)\$$ | shift $+$ |
| $\$(E+$ | $y)\$$ | shift $y$ |
| $\$(E+y$ | $)\$$ | reduce $F \to id$ |

| stack | input | action$[X, a]$ |
|---|---|---|
| $\$(E+F$ | $)\$$ | reduce $T \to F$ |
| $\$(E+T$ | $)\$$ | reduce $E \to E+T$ |
| $\$(E$ | $)\$$ | shift ) |
| $\$(E)$ | $\$$ | reduce $F \to (E)$ |
| $\$F$ | $\$$ | reduce $T \to F$ |
| $\$T$ | $\$$ | reduce $E \to T$ |
| $\$E$ | $\$$ | reduce $S \to E$ |
| $\$S$ | $\$$ | accept! |

# Aside: shift and reduce construct trees



$S \to E\,\$$
$E \to E + T$
$E \to T$
$T \to T * F$
$T \to F$
$F \to (E)$
$F \to \mathrm{id}$

# Aside: shift and reduce construct trees



shift (

$S \to E\,\$$
$E \to E + T$
$E \to T$
$T \to T * F$
$T \to F$
$F \to (E)$
$F \to \mathrm{id}$

(

# Aside: shift and reduce construct trees



shift (
shift $x$

$S \to E\,\$$
$E \to E + T$
$E \to T$
$T \to T * F$
$T \to F$
$F \to (E)$
$F \to \mathrm{id}$

( x

# Aside: shift and reduce construct trees



shift (
shift $x$
reduce $F \to id$

$S \to E\,\$$
$E \to E + T$
$E \to T$
$T \to T * F$
$T \to F$
$F \to (E)$
$F \to \text{id}$

# Aside: shift and reduce construct trees



shift (
shift $x$
reduce $F \to id$
reduce $T \to F$

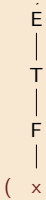
$S \to E\,\$$
$E \to E + T$
$E \to T$
$T \to T * F$
$T \to F$
$F \to (E)$
$F \to \mathrm{id}$

# Aside: shift and reduce construct trees



shift (
shift $x$
reduce $F \to id$
reduce $T \to F$
reduce $E \to T$

$S \to E\,\$$
$E \to E + T$
$E \to T$
$T \to T * F$
$T \to F$
$F \to (E)$
$F \to \text{id}$



# Aside: shift and reduce construct trees



shift (
shift $x$
reduce $F \to id$
reduce $T \to F$
reduce $E \to T$
shift $+$

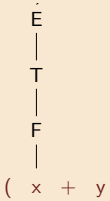
$S \to E\,\$$
$E \to E + T$
$E \to T$
$T \to T * F$
$T \to F$
$F \to (E)$
$F \to \mathrm{id}$

# Aside: shift and reduce construct trees



shift (
shift $x$
reduce $F \to id$
reduce $T \to F$
reduce $E \to T$
shift $+$
shift $y$

$S \to E\,\$$
$E \to E + T$
$E \to T$
$T \to T * F$
$T \to F$
$F \to (E)$
$F \to \mathrm{id}$



# Aside: shift and reduce construct trees

shift (
shift $x$
reduce $F \to id$
reduce $T \to F$
reduce $E \to T$
shift $+$
shift $y$
reduce $F \to id$

$S \to E\,\$$
$E \to E + T$
$E \to T$
$T \to T * F$
$T \to F$
$F \to (E)$
$F \to \mathrm{id}$

# Aside: shift and reduce construct trees



shift (
shift $x$
reduce $F \to id$
reduce $T \to F$
reduce $E \to T$
shift $+$
shift $y$
reduce $F \to id$
reduce $T \to F$

$S \to E\,\$$
$E \to E + T$
$E \to T$
$T \to T * F$
$T \to F$
$F \to (E)$
$F \to \mathrm{id}$

# Aside: shift and reduce construct trees

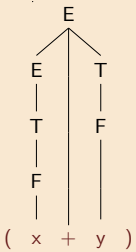


shift (
shift $x$
reduce $F \to id$
reduce $T \to F$
reduce $E \to T$
shift $+$
shift $y$
reduce $F \to id$
reduce $T \to F$
reduce $E \to E + T$

$S \to E\,\$$
$E \to E + T$
$E \to T$
$T \to T * F$
$T \to F$
$F \to (E)$
$F \to \mathrm{id}$

# Aside: shift and reduce construct trees

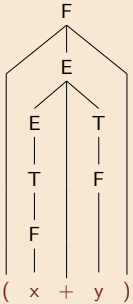
shift (
shift $x$
reduce $F \to id$
reduce $T \to F$
reduce $E \to T$
shift $+$
shift $y$
reduce $F \to id$
reduce $T \to F$
reduce $E \to E + T$
shift )




$S \to E\,\$$
$E \to E + T$
$E \to T$
$T \to T * F$
$T \to F$
$F \to (E)$
$F \to \mathrm{id}$

# Aside: shift and reduce construct trees

shift (
shift $x$
reduce $F \to id$
reduce $T \to F$
reduce $E \to T$
shift $+$
shift $y$
reduce $F \to id$
reduce $T \to F$
reduce $E \to E + T$
shift )
reduce $F \to (E)$

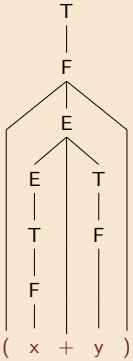



$S \to E\,\$$
$E \to E + T$
$E \to T$
$T \to T * F$
$T \to F$
$F \to (E)$
$F \to \mathrm{id}$

# Aside: shift and reduce construct trees



shift (
shift $x$
reduce $F \to id$
reduce $T \to F$
reduce $E \to T$
shift $+$
shift $y$
reduce $F \to id$
reduce $T \to F$
reduce $E \to E + T$
shift )
reduce $F \to (E)$
reduce $T \to F$

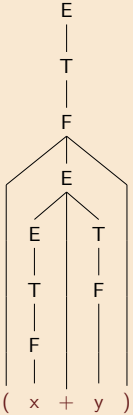



$S \to E\,\$$
$E \to E + T$
$E \to T$
$T \to T * F$
$T \to F$
$F \to (E)$
$F \to \mathrm{id}$

# Aside: shift and reduce construct trees



shift (
shift $x$
reduce $F \to id$
reduce $T \to F$
reduce $E \to T$
shift $+$
shift $y$
reduce $F \to id$
reduce $T \to F$
reduce $E \to E + T$
shift )
reduce $F \to (E)$
reduce $T \to F$
reduce $E \to T$

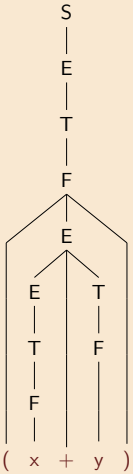



$S \to E\,\$$
$E \to E + T$
$E \to T$
$T \to T * F$
$T \to F$
$F \to (E)$
$F \to \mathrm{id}$

# Aside: shift and reduce construct trees



shift (
shift $x$
reduce $F \to id$
reduce $T \to F$
reduce $E \to T$
shift $+$
shift $y$
reduce $F \to id$
reduce $T \to F$
reduce $E \to E + T$
shift )
reduce $F \to (E)$
reduce $T \to F$
reduce $E \to T$
reduce $S \to E$




$S \to E\,\$$
$E \to E + T$
$E \to T$
$T \to T * F$
$T \to F$
$F \to (E)$
$F \to \mathrm{id}$

# How do we decide when to shift or reduce?



Suppose $A \to \beta\gamma$ is a production. In the configuration

$$\$\alpha\beta\gamma, x\$$$

we *might* want to reduce with $A \to \beta\gamma$.

However, if we have

$$\$\alpha\beta, x\$$$

we *might* want to continue parsing,
hoping to eventually have $\beta\gamma$ on top of the stack,
so that we can then reduce to $A$.

# Items

# LR(0) items

**LR(0) items** record how much of a production's RHS is already parsed.

For every grammar production

$$A \to \beta\gamma \qquad (\beta, \gamma \in (N \cup T)^*)$$

there is an LR(0) item

$$A \to \beta \bullet \gamma$$

$A \to \beta \bullet \gamma$ means:
- we've parsed input $x$ derivable from $\beta$
- we *might* next see input derivable from $\gamma$

# LR(0) items for $G_2'$

| | | | |
|---|---|---|---|
| $S \to \bullet E$ | $E \to \bullet E + T$ | $T \to \bullet T * F$ | $F \to \bullet ( E )$ |
| $S \to E \bullet$ | $E \to E \bullet + T$ | $T \to T \bullet * F$ | $F \to ( \bullet E )$ |
| | $E \to E + \bullet T$ | $T \to T * \bullet F$ | $F \to ( E \bullet )$ |
| | $E \to E + T \bullet$ | $T \to T * F \bullet$ | $F \to ( E ) \bullet$ |
| | $E \to \bullet T$ | $T \to \bullet F$ | $F \to \bullet \mathrm{id}$ |
| | $E \to T \bullet$ | $T \to F \bullet$ | $F \to \mathrm{id} \bullet$ |

**Definition**: item $A \to \beta \bullet \gamma$ is **valid for** $\phi\beta$ if there exists a derivation

$$\begin{aligned} & S \\ \Rightarrow_{rm}^{*} \quad & \phi A w \\ \Rightarrow_{rm} \quad & \phi\beta\gamma w \end{aligned}$$

If

$$A \to \beta \bullet \gamma \text{ is valid for } \phi\beta$$

then

parser can use $A \to \beta \bullet \gamma$ as a guide in configuration $\$\phi\beta, w\$$

Suppose parser is in config $\$\phi\beta, cz\$$ and $A \to \beta \bullet c\gamma$ is valid for $\phi\beta$.
Then we *might* shift $c$ onto the stack:

$$\$\phi\beta, cz\$ \xrightarrow{\text{shift } c} \$\phi\beta c, z\$$$

Suppose parser is in config $\$\phi\beta, z\$$ and $A \to \beta \bullet$ is valid for $\phi\beta$.
Then we *might* perform a reduction

$$\$\phi\beta, z\$ \xrightarrow[A \to \beta]{\text{reduce}} \$\phi A, z\$$$

# Using items as parsing guides (continued)

Suppose parser is in valid config $\$\phi\beta, w\$$ (so $S \Rightarrow^*_{rm} \phi\beta w$).

Suppose $A \to \beta \bullet \gamma$ is valid for $\phi\beta$.

Then $\gamma$ *might* capture the future of our parse (the past of the derivation).

That is, it *might* be that

$$\begin{array}{lll} & S & \\ \Rightarrow^*_{rm} & \phi A x & \\ \Rightarrow_{rm} & \phi\beta\gamma x & \\ \Rightarrow^*_{rm} & \phi\beta y x & = \quad \phi\beta w \end{array}$$

If so, our parser *might* proceed like so:

$$\begin{array}{lll} & \$\phi\beta, yx\$ & = \quad \$\phi\beta, w\$ \\ \to^* & \$\phi\beta\gamma, x\$ & \\ \xrightarrow{\text{reduce}} & \$\phi A, x\$ & \end{array}$$

i.e. our parser could guess that $\gamma$ will derive a prefix of the remaining input w.

# Key idea

# The key idea in LR parsing



**Idea**: Augment shift/reduce parser so that in every configuration $\$\alpha, w\$$ it can derive the set of items valid for $\alpha$.

At each step parser can (non-deterministically) select an item to use as a guide.

# NFA with LR(0) items as states

$$A \to \beta \bullet c\gamma \xrightarrow{c} A \to \beta c \bullet \gamma$$

$$A \to \beta \bullet B\gamma \xrightarrow{B} A \to \beta B \bullet \gamma$$

$$A \to \beta \bullet B\gamma \xrightarrow{\epsilon} B \to \bullet \alpha_i$$

Initial state is item constructed from unique starting production, e.g.:

$$q0 \quad = \quad S \to \bullet E$$

Let $\delta_G$ be the transition function of this NFA (and every state be accepting).

**Theorem**:

$$A \to \beta \bullet \gamma \in \delta_G(q_0, \phi\beta)$$

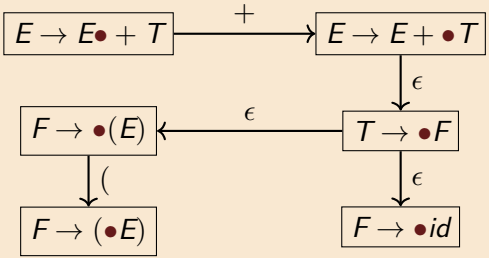
*if and only if*

$$A \to \beta \bullet \gamma \text{ is valid for } \phi\beta.$$

(NB: The set of words $\phi\beta$ is a *regular* language!)

# A few NFA transitions for grammar $G_2$





# A non-deterministic LR parsing algorithm

$c :=$ NextToken()

while true:

  $\alpha :=$ the stack

  if $A \to \beta \bullet c\gamma \in \delta_G(q_0, \alpha)$
    then SHIFT $c$; $c :=$ NextToken()

  if $A \to \beta \bullet \in \delta_G(q_0, \alpha)$
    then REDUCE via $A \to \beta$

  if $S \to \beta \bullet \in \delta_G(q_0, \alpha)$
    then ACCEPT (if no more input)

  if none of the above
    then ERROR

**non-deterministic** since
multiple "if" conditions can be true
& multiple items can match any condition

Next time: SLR(1) & LR(1)