

Compiler Construction

Lecture 5: Foundations of LR parsing

Jeremy Yallop

`jeremy.yallop@cl.cam.ac.uk`

Lent 2025

Derivations

Recap: example grammars

Derivations

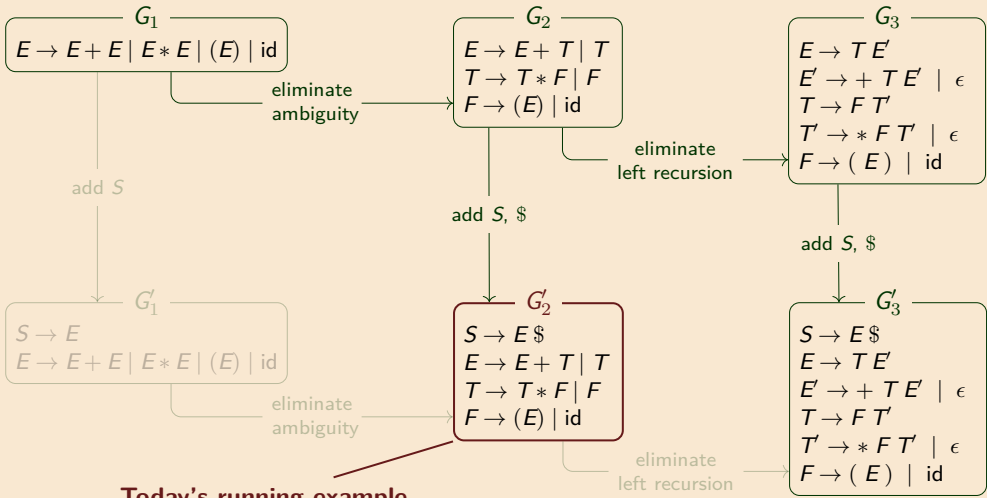


Formalisation

Shift & reduce

Items

Key idea



Leftmost vs rightmost derivations

Derivations



Formalisation

Leftmost derivation step:

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

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

Rightmost derivation step:

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

(basis of bottom-up (**LR**) parsing)

Shift & reduce

where

$$w \in T^*$$

$$\alpha, \beta \in (N \cup T)^*$$

$$A \rightarrow \beta \in P$$

Items

Key idea

Bottom-up parsers perform the derivation in reverse

Derivations

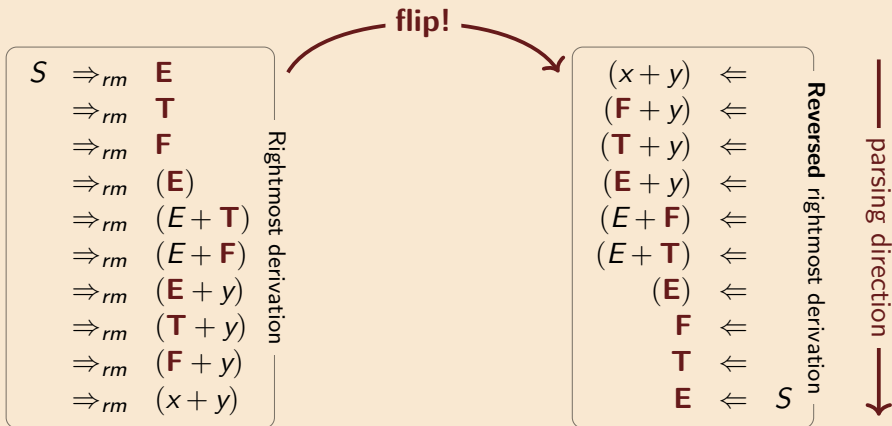


Formalisation

Shift & reduce

Items

Key idea



Backwards derivation $\rightsquigarrow$ stack machine execution

Derivations



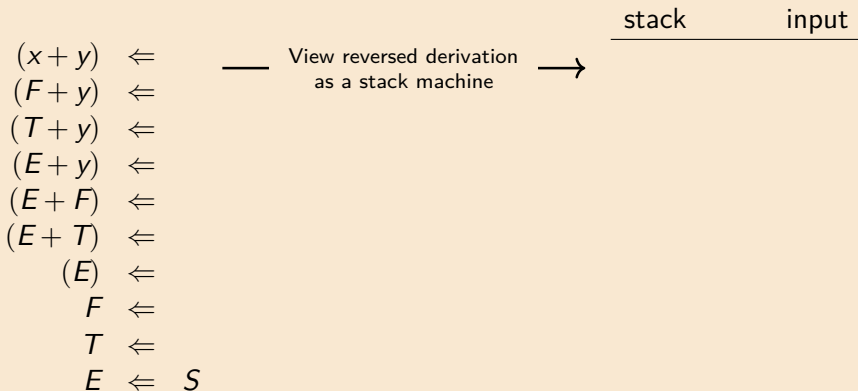
Formalisation

Shift & reduce

Items

Key idea

$$\begin{array}{l} S \rightarrow E \$ \\ E \rightarrow E + T \mid T \end{array} \quad G'_2 \quad \begin{array}{l} T \rightarrow T * F \mid F \\ F \rightarrow (E) \mid \text{id} \end{array}$$



Backwards derivation $\rightsquigarrow$ stack machine execution

Derivations



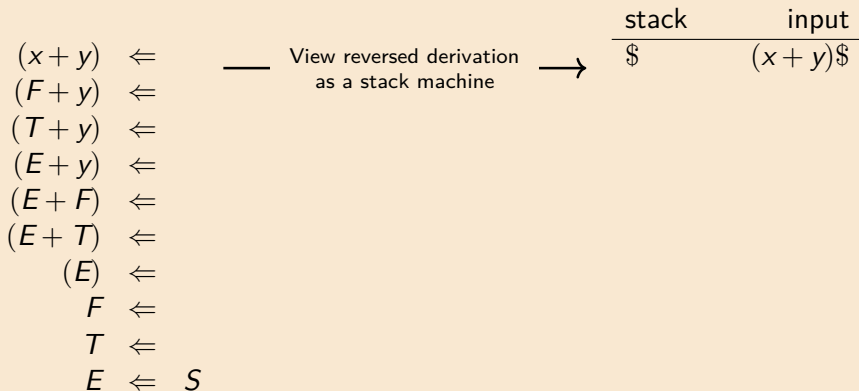
Formalisation

Shift & reduce

Items

Key idea

$$\begin{array}{l} S \rightarrow E \$ \\ E \rightarrow E + T \mid T \end{array} \quad G'_2 \quad \begin{array}{l} T \rightarrow T * F \mid F \\ F \rightarrow (E) \mid \text{id} \end{array}$$



Backwards derivation $\rightsquigarrow$ stack machine execution

Derivations



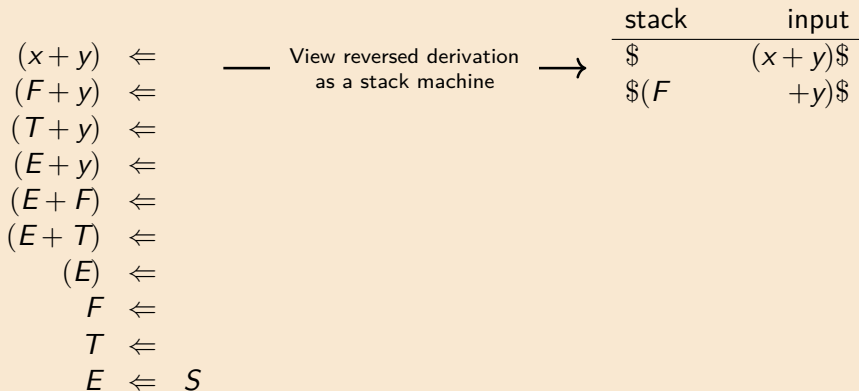
Formalisation

Shift & reduce

Items

Key idea

$$G'_2 \quad \begin{array}{l} S \rightarrow E \$ \\ E \rightarrow E + T \mid T \end{array} \quad \begin{array}{l} T \rightarrow T * F \mid F \\ F \rightarrow (E) \mid \text{id} \end{array}$$



Backwards derivation $\rightsquigarrow$ stack machine execution

Derivations



Formalisation

Shift & reduce

Items

Key idea

$$\frac{S \rightarrow E\$ \quad E \rightarrow E+T \mid T}{G'_2} \quad \frac{T \rightarrow T*F \mid F \quad F \rightarrow (E) \mid \text{id}}{G'_2}$$

$(x + y) \Leftarrow$ $(F + y) \Leftarrow$ $(T + y) \Leftarrow$ $(E + y) \Leftarrow$ $(E + F) \Leftarrow$ $(E + T) \Leftarrow$ $(E) \Leftarrow$ $F \Leftarrow$ $T \Leftarrow$ $E \Leftarrow S$	$\xrightarrow{\text{View reversed derivation as a stack machine}}$	<table border="0"> <tr> <th style="text-align: left;">stack</th> <th style="text-align: left;">input</th> </tr> <tr> <td>\$</td> <td>$(x + y)\$</td> </tr> <tr> <td>$\$(F$</td> <td>$+y)\$</td> </tr> <tr> <td>$\$(T$</td> <td>$+y)\$</td> </tr> </table>	stack	input	\$	$(x + y)$$	$$(F$	$+y)$$	$$(T$	$+y)$$
stack	input									
\$	$(x + y)$$									
$$(F$	$+y)$$									
$$(T$	$+y)$$									

Backwards derivation $\rightsquigarrow$ stack machine execution

Derivations



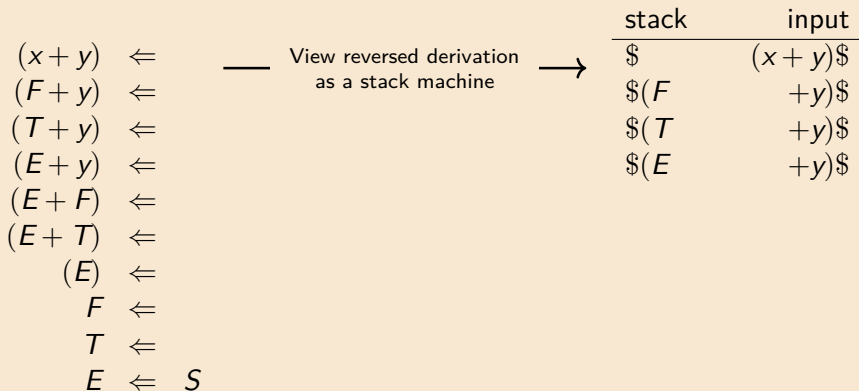
Formalisation

Shift &
reduce

Items

Key idea

$$\begin{array}{l} S \rightarrow E \$ \\ E \rightarrow E + T \mid T \end{array} \quad G_2 \quad \begin{array}{l} T \rightarrow T * F \mid F \\ F \rightarrow (E) \mid \text{id} \end{array}$$



Backwards derivation $\rightsquigarrow$ stack machine execution

Derivations



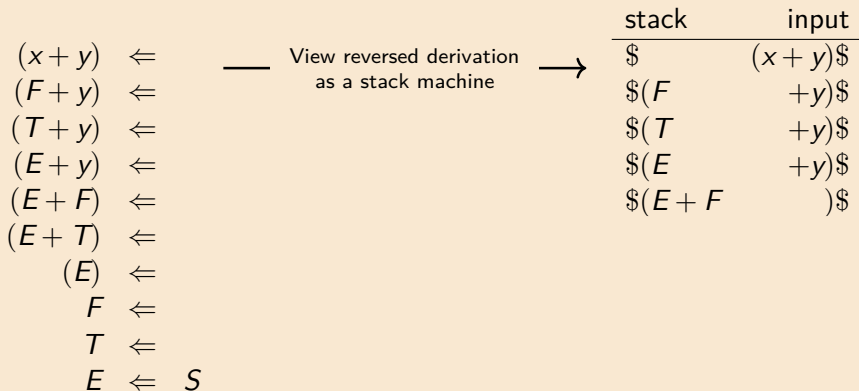
Formalisation

Shift & reduce

Items

Key idea

$$\begin{array}{l} S \rightarrow E \$ \\ E \rightarrow E + T \mid T \end{array} \quad G_2 \quad \begin{array}{l} T \rightarrow T * F \mid F \\ F \rightarrow (E) \mid \text{id} \end{array}$$



Backwards derivation $\rightsquigarrow$ stack machine execution

Derivations



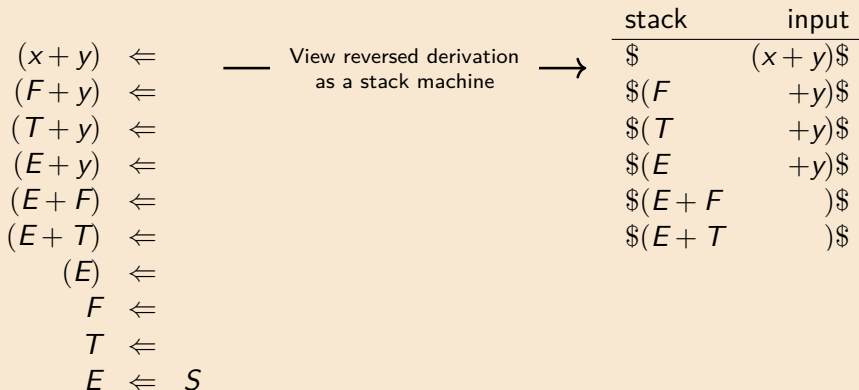
Formalisation

Shift &
reduce

Items

Key idea

$$\begin{array}{l} S \rightarrow E \$ \\ E \rightarrow E + T \mid T \end{array} \quad G_2 \quad \begin{array}{l} T \rightarrow T * F \mid F \\ F \rightarrow (E) \mid \text{id} \end{array}$$



Backwards derivation $\rightsquigarrow$ stack machine execution

Derivations



Formalisation

Shift &
reduce

Items

Key idea

$$G_2 \quad \begin{array}{l} S \rightarrow E \$ \\ E \rightarrow E + T \mid T \end{array} \quad \begin{array}{l} T \rightarrow T * F \mid F \\ F \rightarrow (E) \mid \text{id} \end{array}$$

			stack	input
		View reversed derivation as a stack machine	\$	$(x + y) \$$
$(x + y) \Leftarrow$			$\$(F$	$+y) \$$
$(F + y) \Leftarrow$			$\$(T$	$+y) \$$
$(T + y) \Leftarrow$			$\$(E$	$+y) \$$
$(E + y) \Leftarrow$			$\$(E + F$	$) \$$
$(E + F) \Leftarrow$			$\$(E + T$	$) \$$
$(E + T) \Leftarrow$			$\$(E)$	$\$$
$(E) \Leftarrow$				
$F \Leftarrow$				
$T \Leftarrow$				
$E \Leftarrow S$				

Backwards derivation $\rightsquigarrow$ stack machine execution

Derivations



Formalisation

Shift & reduce

Items

Key idea

$$G_2 \quad \begin{array}{l} S \rightarrow E \$ \\ E \rightarrow E + T \mid T \end{array} \quad \begin{array}{l} T \rightarrow T * F \mid F \\ F \rightarrow (E) \mid \text{id} \end{array}$$

			stack	input
$(x + y) \Leftarrow$	$\longrightarrow$	View reversed derivation as a stack machine	$\longrightarrow$	$\begin{array}{c} \$ \\ (x + y) \$ \end{array}$
$(F + y) \Leftarrow$			$\begin{array}{c} \$ (F \\ \$ (T \end{array}$	$\begin{array}{c} + y) \$ \\ + y) \$ \end{array}$
$(T + y) \Leftarrow$			$\begin{array}{c} \$ (E \\ \$ (E + F \end{array}$	$\begin{array}{c} + y) \$ \\) \$ \end{array}$
$(E + y) \Leftarrow$			$\begin{array}{c} \$ (E + T \\ \$ (E \end{array}$	$\begin{array}{c}) \$ \\ \$ \end{array}$
$(E + F) \Leftarrow$			$\begin{array}{c} \$ F \end{array}$	$\begin{array}{c} \$ \end{array}$
$(E + T) \Leftarrow$				
$(E) \Leftarrow$				
$F \Leftarrow$				
$T \Leftarrow$				
$E \Leftarrow S$				

Backwards derivation $\rightsquigarrow$ stack machine execution

Derivations



Formalisation

Shift &
reduce

Items

Key idea

$$G_2$$

$$\begin{array}{l} S \rightarrow E \$ \\ E \rightarrow E + T \mid T \end{array} \quad \begin{array}{l} T \rightarrow T * F \mid F \\ F \rightarrow (E) \mid \text{id} \end{array}$$

			stack	input
		View reversed derivation as a stack machine	\$	(x + y)\$
(x + y) $\Leftarrow$			\$(F	+y)\$
(F + y) $\Leftarrow$			\$(T	+y)\$
(T + y) $\Leftarrow$			\$(E	+y)\$
(E + y) $\Leftarrow$			\$(E + F	)\$
(E + F) $\Leftarrow$			\$(E + T	)\$
(E + T) $\Leftarrow$			\$(E)	\$
(E) $\Leftarrow$			\$F	\$
F $\Leftarrow$			\$T	\$
T $\Leftarrow$				
E $\Leftarrow$ S				

Backwards derivation $\rightsquigarrow$ stack machine execution

Derivations



Formalisation

Shift &
reduce

Items

Key idea

$$G_2 \quad \begin{array}{l} S \rightarrow E \$ \\ E \rightarrow E + T \mid T \end{array} \quad \begin{array}{l} T \rightarrow T * F \mid F \\ F \rightarrow (E) \mid \text{id} \end{array}$$

			stack	input
		View reversed derivation as a stack machine	\$	(x + y)\$
	$(x + y) \Leftarrow$		$\$(F$	$+y)\$$
	$(F + y) \Leftarrow$		$\$(T$	$+y)\$$
	$(T + y) \Leftarrow$		$\$(E$	$+y)\$$
	$(E + y) \Leftarrow$		$\$(E + F$	)\$
	$(E + F) \Leftarrow$		$\$(E + T$	)\$
	$(E + T) \Leftarrow$		$\$(E)$	\$
	$(E) \Leftarrow$		$\$F$	\$
	$F \Leftarrow$		$\$T$	\$
	$T \Leftarrow$		$\$E$	\$
	$E \Leftarrow S$			

Backwards derivation $\rightsquigarrow$ stack machine execution

Derivations



Formalisation

$$G_2 \quad \begin{array}{l} S \rightarrow E \$ \\ E \rightarrow E + T \mid T \end{array} \quad \begin{array}{l} T \rightarrow T * F \mid F \\ F \rightarrow (E) \mid \text{id} \end{array}$$

Shift & reduce

Items

Key idea

			stack	input
$(x + y) \Leftarrow$	View reversed derivation as a stack machine	$\longrightarrow$	\$	$(x + y) \$$
$(F + y) \Leftarrow$			$\$(F$	$+y) \$$
$(T + y) \Leftarrow$			$\$(T$	$+y) \$$
$(E + y) \Leftarrow$			$\$(E$	$+y) \$$
$(E + F) \Leftarrow$			$\$(E + F$	)\$
$(E + T) \Leftarrow$			$\$(E + T$	)\$
$(E) \Leftarrow$			$\$(E)$	\$
$F \Leftarrow$			$\$F$	\$
$T \Leftarrow$			$\$T$	\$
$E \Leftarrow S$			$\$E$	\$
			$\$S$	\$

Formalisation

LR parser configurations

Derivations

Formalisation

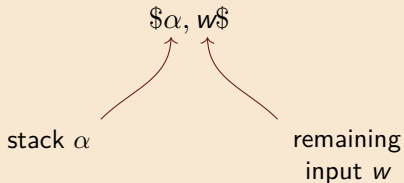


Shift &
reduce

Items

Key idea

An **LR parser configuration** has the form



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

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

(NB: stacks now grow *rightwards*.)

Derivations

Formalisation



Shift &
reduce

Items

Key idea

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:

$$\$ \alpha \beta B z, w \$ \xrightarrow[A \rightarrow \beta B z]{\text{reduce}} \$ \alpha A, w \$$$

This action is called a **reduction** using production $A \rightarrow \beta B z$.

Reductions are not sufficient

Derivations

Formalisation



Shift &
reduce

Items

Key idea

Suppose we have the derivation:

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

The reverse simulation gets stuck:

$$\begin{array}{ccc} & & \$\alpha\beta\gamma, zw\$ \\ & & \xrightarrow[B \rightarrow \gamma]{\text{reduce}} \$\alpha\beta B, zw\$ \\ & & \xrightarrow{???} ??? \end{array}$$

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

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

$$\begin{array}{lcl}
 & \alpha A w & \\
 \Rightarrow_{rm} & \alpha \beta B z w & \text{(using } A \rightarrow \beta B z \text{)} \\
 \Rightarrow_{rm} & \alpha \beta \gamma z w & \text{(using } B \rightarrow \gamma \text{)}
 \end{array}
 \qquad
 \begin{array}{lcl}
 & \$\alpha\beta\gamma, zw\$ & \\
 \xrightarrow[B \rightarrow \gamma]{\text{reduce}} & \$\alpha\beta B, zw\$ & \\
 \xrightarrow[\text{shift}]{z} & \$\alpha\beta B z, w\$ & \\
 \xrightarrow[A \rightarrow \beta B z]{\text{reduce}} & \$\alpha A, w\$ &
 \end{array}$$

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

Derivations

Formalisation



Shift &
reduce

Items

Key idea

Derivation

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

Our parser's possible actions:

$$\begin{aligned} & \$\alpha\gamma, w y z\$ \\ \xrightarrow[B \rightarrow \gamma]{\text{reduce}} & \$\alpha B, w y z\$ \\ \xrightarrow{\text{shift}} & \$\alpha B w, y z\$ \\ \xrightarrow{\text{shift}} & \$\alpha B w y, z\$ \\ \xrightarrow[A \rightarrow y]{\text{reduce}} & \$\alpha B w A, z\$ \end{aligned}$$

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

Shift & reduce

Shift and reduce are sufficient

Derivations

It appears that if we have a derivation

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

Formalisation

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

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

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

Shift &
reduce



Items

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.

Key idea

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

Derivations

Formalisation

Shift &
reduce



Items

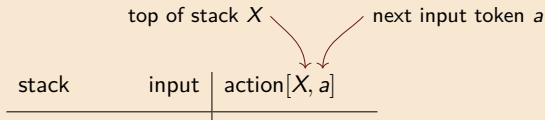
Key idea

$S \rightarrow E \$$

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid \text{id}$



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

Derivations

Formalisation

Shift &
reduce



Items

Key idea

$$\begin{aligned} S &\rightarrow E \$ \\ E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

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

Diagram illustrating the shift/reduce action:

- The stack contains the start symbol $\$$.
- The input is $(x + y) \$$.
- The action is **shift (**, indicating the opening parenthesis is shifted onto the stack.
- Annotations: "top of stack X " points to the stack, and "next input token a " points to the opening parenthesis in the input.

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

Derivations

Formalisation

Shift &
reduce



Items

Key idea

$$\begin{aligned} S &\rightarrow E \$ \\ E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

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

Diagram illustrating the shift/reduce actions for the expression $(x + y)$. The table shows the state of the stack and input at each step. Red arrows point from the text "top of stack X " to the X in the action $[X, a]$ and from "next input token a " to the a in the action $[X, a]$.

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

Derivations

Formalisation

Shift &
reduce



Items

Key idea

$$\begin{aligned} S &\rightarrow E \$ \\ E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

stack	input	action $[X, a]$
\$	$(x + y)\$$	shift (
$\$($	$x + y)\$$	shift x
$\$(x$	$+y)\$$	reduce $F \rightarrow \text{id}$

Diagram labels: "top of stack X " points to the X in the action column. "next input token a " points to the a in the action column.

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

Derivations

Formalisation

Shift &
reduce



Items

Key idea

$$\begin{aligned} S &\rightarrow E \$ \\ E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

stack	input	action $[X, a]$
\$	$(x + y)\$$	shift (
$\$($	$x + y)\$$	shift x
$\$(x$	$+y)\$$	reduce $F \rightarrow \text{id}$
$\$(F$	$+y)\$$	reduce $T \rightarrow F$

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

Derivations

Formalisation

Shift &
reduce



Items

Key idea

$$\begin{aligned} S &\rightarrow E \$ \\ E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

stack	input	action[X, a]
\$	(x + y)\$	shift (
\$(	x + y)\$	shift x
\$(x	+y)\$	reduce $F \rightarrow \text{id}$
\$(F	+y)\$	reduce $T \rightarrow F$
\$(T	+y)\$	reduce $E \rightarrow T$

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

Derivations

Formalisation

Shift &
reduce



Items

Key idea

$$\begin{aligned} S &\rightarrow E \$ \\ E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

stack	input	action[X, a]
\$	(x + y)\$	shift (
\$(	x + y)\$	shift x
\$(x	+y)\$	reduce $F \rightarrow \text{id}$
\$(F	+y)\$	reduce $T \rightarrow F$
\$(T	+y)\$	reduce $E \rightarrow T$
\$(E	+y)\$	shift +

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

Derivations

Formalisation

Shift &
reduce



Items

Key idea

$$\begin{aligned} S &\rightarrow E \$ \\ E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

stack	input	action $[X, a]$
\$	$(x + y)\$$	shift (
$\$($	$x + y)\$$	shift x
$\$(x$	$+y)\$$	reduce $F \rightarrow \text{id}$
$\$(F$	$+y)\$$	reduce $T \rightarrow F$
$\$(T$	$+y)\$$	reduce $E \rightarrow T$
$\$(E$	$+y)\$$	shift +
$\$(E+$	$y)\$$	shift y

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

Derivations

Formalisation

Shift &
reduce



Items

Key idea

$$\begin{aligned} S &\rightarrow E \$ \\ E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

top of stack X next input token a

stack	input	action $[X, a]$
\$	$(x + y)\$$	shift (
$\$($	$x + y)\$$	shift x
$\$(x$	$+y)\$$	reduce $F \rightarrow \text{id}$
$\$(F$	$+y)\$$	reduce $T \rightarrow F$
$\$(T$	$+y)\$$	reduce $E \rightarrow T$
$\$(E$	$+y)\$$	shift +
$\$(E+$	$y)\$$	shift y
$\$(E + y$	$)\$$	reduce $F \rightarrow \text{id}$

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

Derivations

Formalisation

Shift &
reduce



Items

Key idea

$S \rightarrow E \$$
 $E \rightarrow E + T \mid T$
 $T \rightarrow T * F \mid F$
 $F \rightarrow (E) \mid id$

top of stack X next input token a

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

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

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

Derivations

$$\begin{aligned} S &\rightarrow E \$ \\ E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

Formalisation

top of stack X next input token a

stack	input	action[X, a]
\$	$(x + y)\$$	shift (
$\$($	$x + y)\$$	shift x
$\$(x$	$+y)\$$	reduce $F \rightarrow \text{id}$
$\$(F$	$+y)\$$	reduce $T \rightarrow F$
$\$(T$	$+y)\$$	reduce $E \rightarrow T$
$\$(E$	$+y)\$$	shift +
$\$(E+$	$y)\$$	shift y
$\$(E + y$	$)\$$	reduce $F \rightarrow \text{id}$

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

Shift &
reduce



Items

Key idea

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

Derivations

$S \rightarrow E \$$
 $E \rightarrow E + T \mid T$
 $T \rightarrow T * F \mid F$
 $F \rightarrow (E) \mid id$

Formalisation

top of stack X next input token a

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

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

Shift &
reduce



Items

Key idea

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

Derivations

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

Formalisation

top of stack X next input token a

stack	input	action[X, a]
\$	$(x + y)\$$	shift (
$\$($	$x + y)\$$	shift x
$\$(x$	$+y)\$$	reduce $F \rightarrow \text{id}$
$\$(F$	$+y)\$$	reduce $T \rightarrow F$
$\$(T$	$+y)\$$	reduce $E \rightarrow T$
$\$(E$	$+y)\$$	shift +
$\$(E+$	$y)\$$	shift y
$\$(E + y$	$)\$$	reduce $F \rightarrow \text{id}$

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

Shift &
reduce



Items

Key idea

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

Derivations

$$\begin{aligned} S &\rightarrow E \$ \\ E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

Formalisation

top of stack X next input token a

stack	input	action $[X, a]$	stack	input	action $[X, a]$
\$	$(x + y)\$$	shift (	$\$(E + F$	$)\$$	reduce $T \rightarrow F$
$\$($	$x + y)\$$	shift x	$\$(E + T$	$)\$$	reduce $E \rightarrow E + T$
$\$(x$	$+y)\$$	reduce $F \rightarrow \text{id}$	$\$(E$	$)\$$	shift)
$\$(F$	$+y)\$$	reduce $T \rightarrow F$	$\$(E)$	$\$$	reduce $F \rightarrow (E)$
$\$(T$	$+y)\$$	reduce $E \rightarrow T$	$\$F$	$\$$	reduce $T \rightarrow F$
$\$(E$	$+y)\$$	shift +			
$\$(E+$	$y)\$$	shift y			
$\$(E + y$	$)\$$	reduce $F \rightarrow \text{id}$			

Shift &
reduce



Items

Key idea

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

Derivations

$S \rightarrow E \$$
 $E \rightarrow E + T \mid T$
 $T \rightarrow T * F \mid F$
 $F \rightarrow (E) \mid id$

Formalisation

top of stack X next input token a

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

Shift &
reduce



Items

Key idea

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

Derivations

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

Formalisation

top of stack X next input token a

stack	input	action $[X, a]$	stack	input	action $[X, a]$
\$	$(x + y)\$$	shift (	$\$(E + F$	$)\$$	reduce $T \rightarrow F$
$\$($	$x + y)\$$	shift x	$\$(E + T$	$)\$$	reduce $E \rightarrow E + T$
$\$(x$	$+y)\$$	reduce $F \rightarrow \text{id}$	$\$(E$	$)\$$	shift)
$\$(F$	$+y)\$$	reduce $T \rightarrow F$	$\$(E)$	$\$$	reduce $F \rightarrow (E)$
$\$(T$	$+y)\$$	reduce $E \rightarrow T$	$\$F$	$\$$	reduce $T \rightarrow F$
$\$(E$	$+y)\$$	shift +	$\$T$	$\$$	reduce $E \rightarrow T$
$\$(E+$	$y)\$$	shift y	$\$E$	$\$$	reduce $S \rightarrow E$
$\$(E + y$	$)\$$	reduce $F \rightarrow \text{id}$			

Shift &
reduce



Items

Key idea

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

Derivations

$$\begin{aligned} S &\rightarrow E \$ \\ E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

Formalisation

top of stack X next input token a

stack	input	action $[X, a]$	stack	input	action $[X, a]$
\$	$(x + y)\$$	shift (	$\$(E + F$	$)\$$	reduce $T \rightarrow F$
$\$($	$x + y)\$$	shift x	$\$(E + T$	$)\$$	reduce $E \rightarrow E + T$
$\$(x$	$+y)\$$	reduce $F \rightarrow \text{id}$	$\$(E$	$)\$$	shift)
$\$(F$	$+y)\$$	reduce $T \rightarrow F$	$\$(E)$	$\$$	reduce $F \rightarrow (E)$
$\$(T$	$+y)\$$	reduce $E \rightarrow T$	$\$F$	$\$$	reduce $T \rightarrow F$
$\$(E$	$+y)\$$	shift +	$\$T$	$\$$	reduce $E \rightarrow T$
$\$(E+$	$y)\$$	shift y	$\$E$	$\$$	reduce $S \rightarrow E$
$\$(E + y$	$)\$$	reduce $F \rightarrow \text{id}$	$\$S$	$\$$	accept!

Shift &
reduce



Items

Key idea

Aside: shift and reduce construct trees

Derivations

Formalisation

Shift &
reduce



Items

Key idea

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

Aside: shift and reduce construct trees

Derivations

shift (

Formalisation

Shift &
reduce



Items

Key idea

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

(

Aside: shift and reduce construct trees

Derivations

shift (
shift x

Formalisation

Shift &
reduce



Items

Key idea

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

(x

Aside: shift and reduce construct trees

Derivations

shift (
shift x
reduce $F \rightarrow id$

Formalisation

Shift &
reduce



Items

Key idea

$S \rightarrow E \$$
 $E \rightarrow E + T$
 $E \rightarrow T$
 $T \rightarrow T * F$
 $T \rightarrow F$
 $F \rightarrow (E)$
 $F \rightarrow id$

F
|
(x

Aside: shift and reduce construct trees

Derivations

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

Formalisation

Shift &
reduce



Items

Key idea

$S \rightarrow E \$$
 $E \rightarrow E + T$
 $E \rightarrow T$
 $T \rightarrow T * F$
 $T \rightarrow F$
 $F \rightarrow (E)$
 $F \rightarrow id$

T
|
F
|
(x

Aside: shift and reduce construct trees

Derivations

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

Formalisation

Shift &
reduce



Items

Key idea

$S \rightarrow E \$$
 $E \rightarrow E + T$
 $E \rightarrow T$
 $T \rightarrow T * F$
 $T \rightarrow F$
 $F \rightarrow (E)$
 $F \rightarrow id$

$\dot{E}$
|
T
|
F
|
(x

Aside: shift and reduce construct trees

Derivations

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

Formalisation

Shift &
reduce



Items

Key idea

$S \rightarrow E \$$
 $E \rightarrow E + T$
 $E \rightarrow T$
 $T \rightarrow T * F$
 $T \rightarrow F$
 $F \rightarrow (E)$
 $F \rightarrow id$

$\dot{E}$
|
 T
|
 F
|
(x +

Aside: shift and reduce construct trees

Derivations

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

Formalisation

Shift &
reduce



Items

Key idea

$S \rightarrow E \$$
 $E \rightarrow E + T$
 $E \rightarrow T$
 $T \rightarrow T * F$
 $T \rightarrow F$
 $F \rightarrow (E)$
 $F \rightarrow id$

$\dot{E}$
|
T
|
F
|
(x + y

Aside: shift and reduce construct trees

Derivations

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

Formalisation

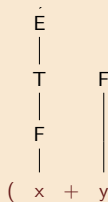
Shift &
reduce



Items

Key idea

$S \rightarrow E \$$
 $E \rightarrow E + T$
 $E \rightarrow T$
 $T \rightarrow T * F$
 $T \rightarrow F$
 $F \rightarrow (E)$
 $F \rightarrow id$



Aside: shift and reduce construct trees

Derivations

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

Formalisation

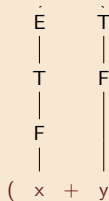
Shift &
reduce



Items

Key idea

$S \rightarrow E \$$
 $E \rightarrow E + T$
 $E \rightarrow T$
 $T \rightarrow T * F$
 $T \rightarrow F$
 $F \rightarrow (E)$
 $F \rightarrow id$



Aside: shift and reduce construct trees

Derivations

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

Formalisation

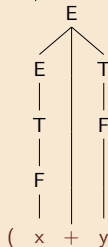
Shift &
reduce



Items

Key idea

$S \rightarrow E \$$
 $E \rightarrow E + T$
 $E \rightarrow T$
 $T \rightarrow T * F$
 $T \rightarrow F$
 $F \rightarrow (E)$
 $F \rightarrow id$



Aside: shift and reduce construct trees

Derivations

Formalisation

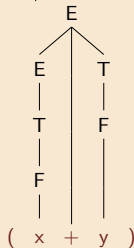
Shift &
reduce



Items

Key idea

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



$S \rightarrow E \$$
 $E \rightarrow E + T$
 $E \rightarrow T$
 $T \rightarrow T * F$
 $T \rightarrow F$
 $F \rightarrow (E)$
 $F \rightarrow id$

Aside: shift and reduce construct trees

Derivations

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

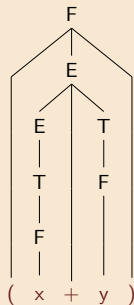
Formalisation

Shift &
reduce



Items

Key idea



$S \rightarrow E \$$
 $E \rightarrow E + T$
 $E \rightarrow T$
 $T \rightarrow T * F$
 $T \rightarrow F$
 $F \rightarrow (E)$
 $F \rightarrow id$

Aside: shift and reduce construct trees

Derivations

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

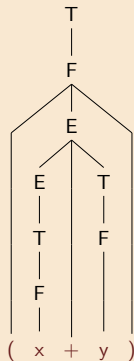
Formalisation

Shift &
reduce



Items

Key idea



$S \rightarrow E \$$
 $E \rightarrow E + T$
 $E \rightarrow T$
 $T \rightarrow T * F$
 $T \rightarrow F$
 $F \rightarrow (E)$
 $F \rightarrow id$

Aside: shift and reduce construct trees

Derivations

Formalisation

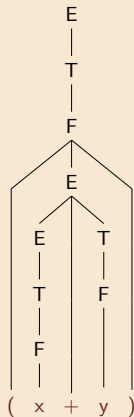
Shift &
reduce



Items

Key idea

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



$S \rightarrow E \$$
 $E \rightarrow E + T$
 $E \rightarrow T$
 $T \rightarrow T * F$
 $T \rightarrow F$
 $F \rightarrow (E)$
 $F \rightarrow id$

Aside: shift and reduce construct trees

Derivations

Formalisation

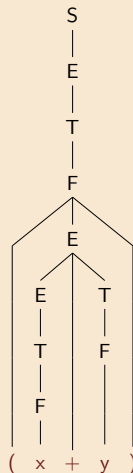
Shift &
reduce



Items

Key idea

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



$S \rightarrow E \$$
 $E \rightarrow E + T$
 $E \rightarrow T$
 $T \rightarrow T * F$
 $T \rightarrow F$
 $F \rightarrow (E)$
 $F \rightarrow id$

How do we decide when to shift or reduce?

Derivations

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

Formalisation

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

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

Shift &
reduce

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

Key idea

Items

Derivations

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

Formalisation

For every grammar production

$$A \rightarrow \beta\gamma \quad (\beta, \gamma \in (N \cup T)^*)$$

there is an LR(0) item

$$A \rightarrow \beta \bullet \gamma$$

Shift &
reduce

Items



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

Key idea

LR(0) items for G_2'

Derivations

Formalisation

Shift &
reduce

Items



Key idea

$$S \rightarrow \bullet E$$

$$S \rightarrow E \bullet$$

$$E \rightarrow \bullet E + T$$

$$E \rightarrow E \bullet + T$$

$$E \rightarrow E + \bullet T$$

$$E \rightarrow E + T \bullet$$

$$E \rightarrow \bullet T$$

$$E \rightarrow T \bullet$$

$$T \rightarrow \bullet T * F$$

$$T \rightarrow T \bullet * F$$

$$T \rightarrow T * \bullet F$$

$$T \rightarrow T * F \bullet$$

$$T \rightarrow \bullet F$$

$$T \rightarrow F \bullet$$

$$F \rightarrow \bullet (E)$$

$$F \rightarrow (\bullet E)$$

$$F \rightarrow (E \bullet)$$

$$F \rightarrow (E) \bullet$$

$$F \rightarrow \bullet \text{id}$$

$$F \rightarrow \text{id} \bullet$$

Derivations

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

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

Formalisation

Shift &
reduce

If

$A \rightarrow \beta \bullet \gamma$ is valid for $\phi\beta$

then

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

Items



Key idea

Using items as parsing guides

Derivations

Formalisation

Shift &
reduce

Items



Key idea

Suppose parser is in config $\$ \phi \beta, cz \$$ and $A \rightarrow \beta \bullet c \gamma$ 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 $A \rightarrow \beta \bullet$ is valid for $\phi \beta$.
Then we *might* perform a reduction

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

Using items as parsing guides (continued)

Derivations

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

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

Formalisation

Then γ *might* capture the future of our parse (the past of the derivation).

That is, it *might* be that

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

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

$$\begin{aligned} \$ \phi \beta, y x \$ &= \$ \phi \beta, w \$ \\ &\xrightarrow{*} \$ \phi \beta \gamma, x \$ \\ \xrightarrow{\text{reduce}} & \$ \phi A, x \$ \end{aligned}$$

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

Shift &
reduce

Items



Key idea

Key idea

The key idea in LR parsing

Derivations

Formalisation

Shift &
reduce

Items

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

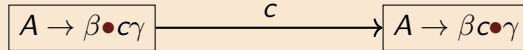
At each step parser can (non-deterministically) select an item to use as a guide.

Key idea

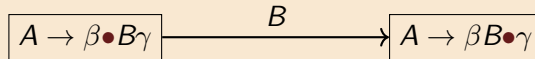


NFA with LR(0) items as states

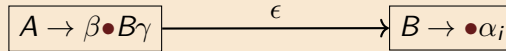
Derivations



Formalisation



Shift &
reduce



Items

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

$$q_0 = S \rightarrow \bullet E$$

Let δ_G be the transition function of this NFA (and every state be accepting).

Key idea



Main LR parsing theorem

Derivations

Formalisation

Shift &
reduce

Items

Theorem:

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

if and only if

$A \rightarrow \beta \bullet \gamma$ is valid for $\phi\beta$.

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

Key idea



A few NFA transitions for grammar G_2

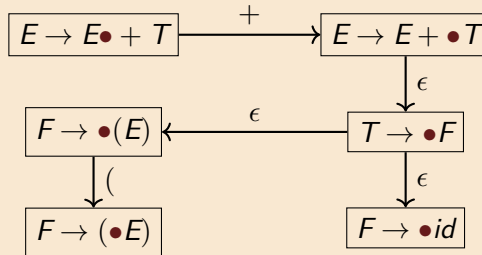
Derivations

Formalisation

Shift &
reduce

Items

Key idea



A non-deterministic LR parsing algorithm

Derivations

$c := \text{NextToken}()$

while true:

$\alpha := \text{the stack}$

if $A \rightarrow \beta \bullet c \gamma \in \delta_G(q_0, \alpha)$
then **SHIFT** c ; $c := \text{NextToken}()$

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

if $S \rightarrow \beta \bullet \in \delta_G(q_0, \alpha)$
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

Key idea



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