Lexical Analysis & Parsing

15-411/15-611 Compiler Design

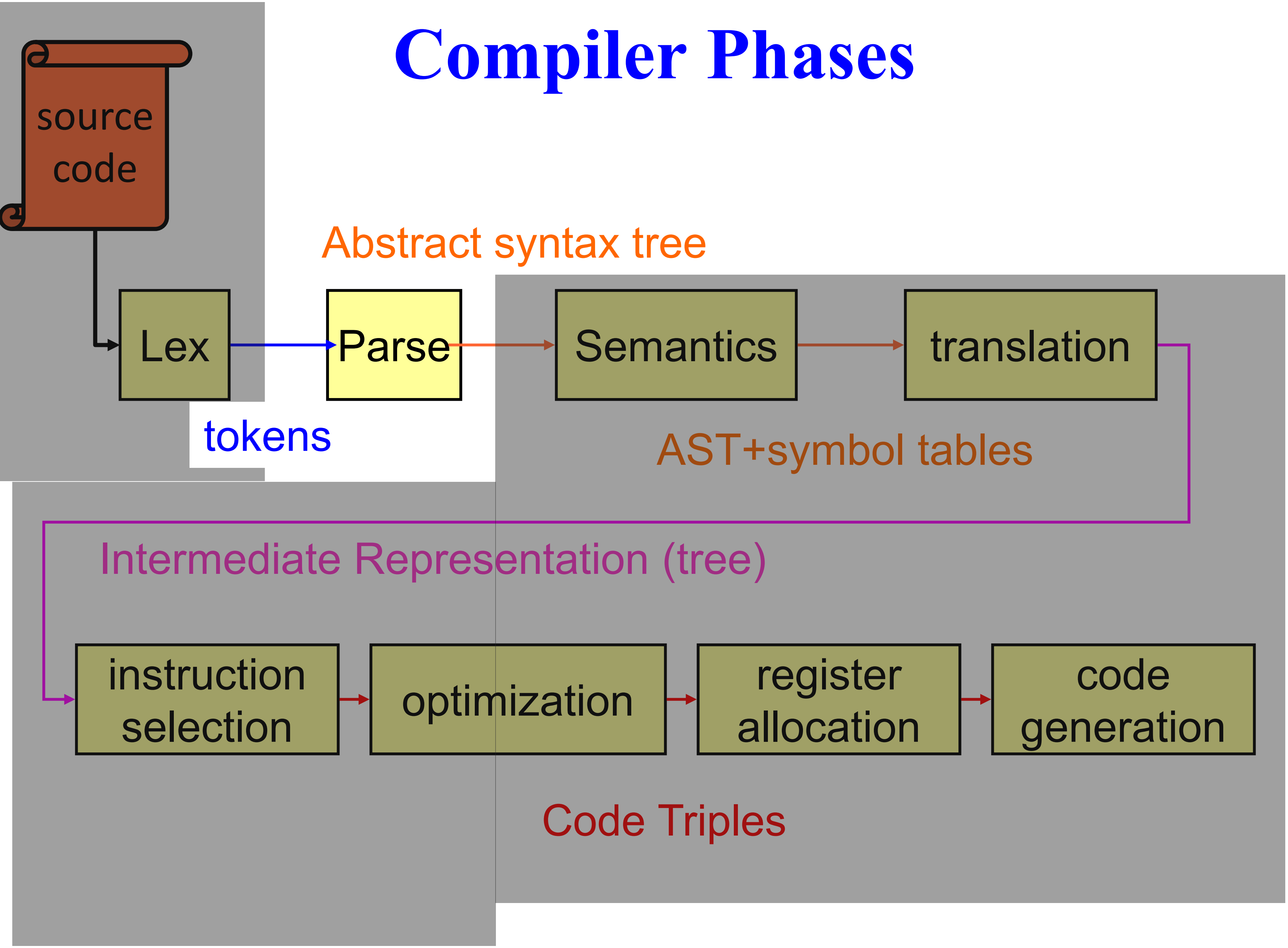
Seth Copen Goldstein

October 1, 2020

Today – part 2

- Languages and Grammars
- Context Free Grammars
- Derivations & Parse Trees
- Ambiguity
- Top-down parsers
- FIRST, FOLLOW, and NULLABLE
- Bottom-up parsers

Compiler Phases



FIRST sets

- We use next k characters in input stream to guide the selection of the proper production.
- Given: $A := \alpha \mid \beta$ we want next input character to decide between α and β.
- FIRST(α) = set of terminals that can begin any string derived from α.
- IOW: $\mathbf{a} \in$ FIRST(α) iff $\alpha \Rightarrow^* \mathbf{a}\gamma$ for some γ

- FIRST(α) $\cap$ FIRST(β) = $\varnothing$ $\rightarrow$ no backtracking needed

Computing FIRST(α)

- Given X := A B C, FIRST(X) = FIRST(A B C)
- Can we ignore B or C?
- Consider:

```
A := a
   |
B := b
   | A
C := c
```

Computing FIRST(α)

- Given X := A B C, FIRST(X) = FIRST(A B C)
- Can we ignore B or C?
- Consider:

```
A := a
   |
B := b
   | A
C := c
```

- FIRST(X) must also include FIRST(C)
- IOW:
  - Must keep track of NTs that are nullable
  - For nullable NTs, determine FOLLOWS(NT)

nullable(A)

- nullable(A) is true if A can derive the empty string
- For example:

```
B := X Y b
X := x
   |  Y Y
Y :=
```

In this case, nullable(X) = nullable(Y) = true

nullable(B) = false

FOLLOW(A)

- FOLLOW(A) is the set of terminals that can immediately follow A in a sentential form.
- I.e.,
  a $\in$ FOLLOW(A) iff S $\Rightarrow^*$ αAaβ for some α and β

Top-Down Parsing

- Can be constructed by hand
- LL(k) grammars can be parsed
  – Left-to-right
  – Leftmost-derivation
  – with k symbols lookahead
- Often requires
  – left-factoring
  – Elimination of left-recursion

Bottom-up parsers

- What is the inherent restriction of top-down parsing, e.g., with LL(k) grammars?

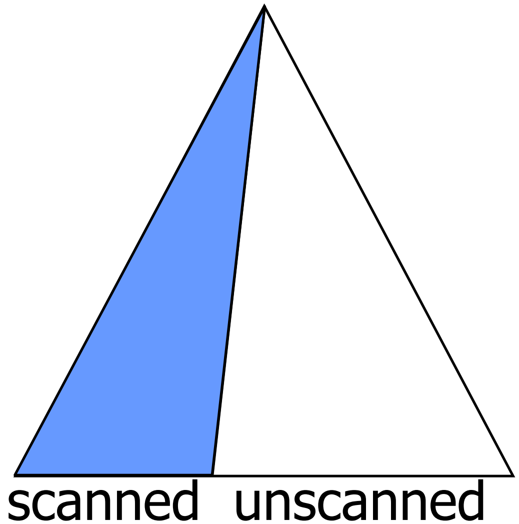
Bottom-up parsers

- What is the inherent restriction of top-down parsing, e.g., with LL(k) grammars?
- Bottom-up parsers use the entire right-hand side of the production
- LR(k):
  - Left-to-right parse,
  - Rightmost derivation (in reverse),
  - k look ahead tokens

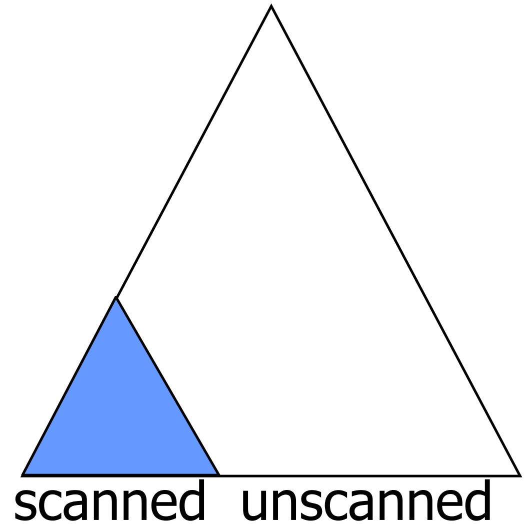
Top-down vs. Bottom-up

LL(k), recursive descent

LR(k), shift-reduce



Top-down



Bottom-up

Example - Top-down

S := X
X := X a
  | b

Is this grammar LL(k)?

How can we make it LL(k)?

S := X
X := b R
R := a R
  |

What about a bottom up parse?

Example - Bottom-up

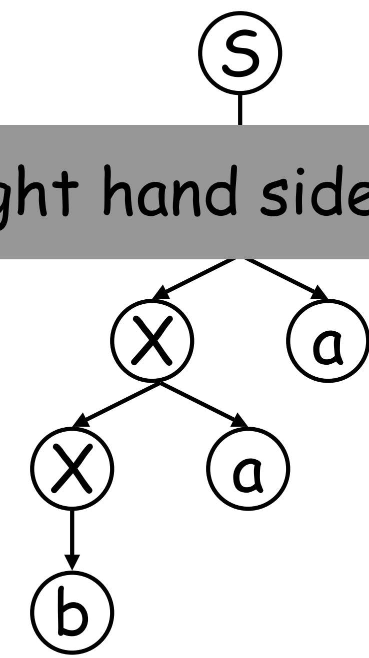
S := X
X := X a
  | b

right-most derivation:

LR parser gets to look at an entire right hand side.

Left-to-Right, Rightmost in reverse

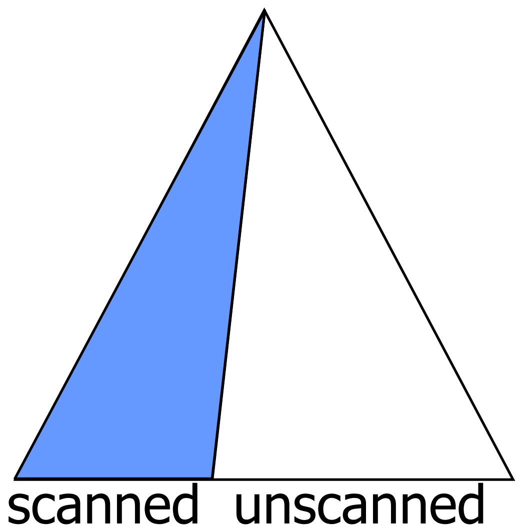
baa
Xaa
Xa
X
S



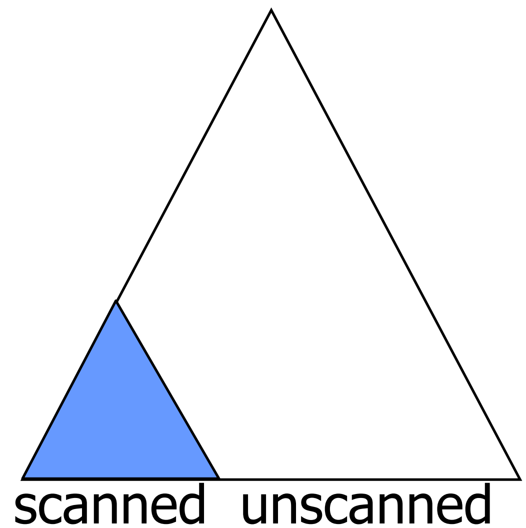
Top-down vs. Bottom-up

LL(k), recursive descent

LR(k), shift-reduce



Top-down



Bottom-up

A Rightmost Derivation

1	S	:= Exp
2	Exp	:= Exp + Term
3	Exp	:= Exp - Term
4	Exp	:= Term
5	Term	:= Term * Factor
6	Term	:= Term / Factor
7	Term	:= Factor
8	Factor	:= **`id`**
9	Factor	:= **`int`**

S

by 1 $\Rightarrow$ Exp

by 2 $\Rightarrow$ Exp + Term

by 5 $\Rightarrow$ Exp + Term * Factor

by 8 $\Rightarrow$ Exp + Term * $\mathbf{id}_x$

by 7 $\Rightarrow$ Exp + Factor * $\mathbf{id}_x$

by 9 $\Rightarrow$ Exp + $\mathbf{int}_3$ * $\mathbf{id}_x$

by 4 $\Rightarrow$ Term + $\mathbf{int}_3$ * $\mathbf{id}_x$

by 7 $\Rightarrow$ Factor + $\mathbf{int}_3$ * $\mathbf{id}_x$

by 9 $\Rightarrow$ $\mathbf{int}_2$ + $\mathbf{int}_3$ * $\mathbf{id}_x$

A Rightmost Derivation In Reverse

int$_2$ + **int**$_3$ * **id**$_x$

Factor + **int**$_3$ * **id**$_x$

Term + **int** * **id**

Lets keep track of where we are in the input.

Exp +

Exp + Factor * **id**$_x$

Exp + Term * **id**$_x$

Exp + Term * Factor

Exp + Term

Exp

S

A Rightmost Derivation In Reverse

$\texttt{int}_2 + \texttt{int}_3 * \texttt{id}_x$

$\text{Factor} + \texttt{int}_3 * \texttt{id}_x$

$\text{Term} + \texttt{int}_3 * \texttt{id}_x$

$\text{Exp} + \texttt{int}_3 * \texttt{id}_x$

$\text{Exp} + \text{Factor} * \texttt{id}_x$

$\text{Exp} + \text{Term} * \texttt{id}_x$

$\text{Exp} + \text{Term} * \text{Factor}$

$\text{Exp} + \text{Term}$

Exp

S

$\texttt{int}_2 \bullet + \texttt{int}_3 * \texttt{id}_x$

$\text{Factor} \bullet + \texttt{int}_3 * \texttt{id}_x$

$\text{Term} \bullet + \texttt{int}_3 * \texttt{id}_x$

$\text{Exp} + \texttt{int}_3 \bullet * \texttt{id}_x$

$\text{Exp} + \text{Factor} \bullet * \texttt{id}_x$

$\text{Exp} + \text{Term} * \texttt{id}_x \bullet$

$\text{Exp} + \text{Term} * \text{Factor} \bullet$

$\text{Exp} + \text{Term} \bullet$

$\text{Exp} \bullet$

$\text{S} \bullet$

A Rightmost Derivation In Reverse

$\texttt{int}_2 + \texttt{int}_3 * \texttt{id}_x$

$\text{Factor} + \texttt{int}_3 * \texttt{id}_x$

$\text{Term} + \texttt{int}_3 * \texttt{id}_x$

$\text{Exp} + \texttt{int}_3 * \texttt{id}_x$

$\text{Exp} + \text{Factor} * \texttt{id}_x$

$\text{Exp} + \text{Term} * \texttt{id}$

$\text{Exp} + \text{Term} *$

$\text{Exp} + \text{Term}$

Exp

S

$\texttt{int}_2 \bullet + \texttt{int}_3 * \texttt{id}_x$

$\text{Factor} \bullet + \texttt{int}_3 * \texttt{id}_x$

$\text{Term} \bullet + \texttt{int}_3 * \texttt{id}_x$

$\text{Exp} + \texttt{int}_3 \bullet * \texttt{id}_x$

$\text{Exp} + \text{Factor} \bullet * \texttt{id}_x$

$\text{Exp} + \text{Term} * \texttt{id}_x \bullet$

$\text{Factor} \bullet$

$\text{Exp} \bullet$

$\text{S} \bullet$

Lets format this differently,
<prefix of sentential form> input

A Rightmost Derivation In Reverse

	`int`$_2$ + `int`$_3$ * `id`$_x$ \$
`int`$_2$	+ `int`$_3$ * `id`$_x$ \$
Factor	+ `int`$_3$ * `id`$_x$ \$
Term	+ `int`$_3$ * `id`$_x$ \$
Exp	+ `int`$_3$ * `id`$_x$ \$
Exp +	`int`$_3$ * `id`$_x$ \$
Exp + `int`$_3$	* `id`$_x$ \$
Exp + Factor	* `id`$_x$ \$
Exp + Term	* `id`$_x$ \$
Exp + Term *	`id`$_x$ \$
Exp + Term * `id`$_x$	\$
Exp + Term * Factor	\$
Exp + Term	\$
Exp	\$
S	\$

A Rightmost Derivation In Reverse

	$\texttt{int}_2 + \texttt{int}_3 * \texttt{id}_x$ \$
$\texttt{int}_2$	$+ \texttt{int}_3 * \texttt{id}_x$ \$
Factor	$+ \texttt{int}_3 * \texttt{id}_x$ \$
Term	$+ \texttt{int}_3 * \texttt{id}_x$ \$
Exp	$+ \texttt{int}_3 * \texttt{id}_x$ \$
Exp +	$\texttt{int}_3 * \texttt{id}_x$ \$
Exp + $\texttt{int}_3$	$* \texttt{id}_x$ \$
Exp + Factor	$* \texttt{id}_x$ \$
Exp + Term	$* \texttt{id}_x$ \$
Exp + Term *	$\texttt{id}_x$ \$
Exp + Term * $\texttt{id}_x$	\$
S	\$

LR-Parser either:

1. shifts a terminal or
2. reduces by a production.

A Rightmost Derivation In Reverse

	$\mathbf{int}_2 + \mathbf{int}_3 * \mathbf{id}_x$ \$	shift 2
$\mathbf{int}_2$	$+ \mathbf{int}_3 * \mathbf{id}_x$ \$	
Factor	$+ \mathbf{int}_3 * \mathbf{id}_x$ \$	
Term	$+ \mathbf{int}_3 * \mathbf{id}_x$ \$	
Exp	$+ \mathbf{int}_3 * \mathbf{id}_x$ \$	
Exp +	$\mathbf{int}_3 * \mathbf{id}_x$ \$	
Exp + $\mathbf{int}_3$	$* \mathbf{id}_x$ \$	
Exp + Factor	$* \mathbf{id}_x$ \$	
Exp + Term	$* \mathbf{id}_x$ \$	
Exp + Term *	$\mathbf{id}_x$ \$	
Exp + Term * $\mathbf{id}_x$	\$	
Exp + Term * Factor	\$	
Exp + Term	\$	
Exp	\$	
S	\$	

A Rightmost Derivation In Reverse

	`int`$_2$ + `int`$_3$ * `id`$_x$ \$	shift 2
`int`$_2$	+ `int`$_3$ * `id`$_x$ \$	reduce by F $\rightarrow$ int
Factor		
Term		
Exp		
Exp +		
Exp + `int`$_3$	* `id`$_x$ \$	
Exp + Factor	* `id`$_x$ \$	
Exp + Term	* `id`$_x$ \$	
Exp + Term *	`id`$_x$ \$	
Exp + Term * `id`$_x$	\$	
Exp + Term * Factor	\$	
Exp + Term	\$	
Exp	\$	
S	\$	

When we reduce by a production: A $\rightarrow$ β, β is on right side of sentential form.

E.g., here β is 'int' and production is F $\rightarrow$ int

A Rightmost Derivation In Reverse

	$\mathbf{int}_2 + \mathbf{int}_3 * \mathbf{id}_x$ \$	shift 2
$\mathbf{int}_2$	$+ \mathbf{int}_3 * \mathbf{id}_x$ \$	reduce by F → int
Factor	$+ \mathbf{int}_3 * \mathbf{id}_x$ \$	reduce by T → F
Term	$+ \mathbf{int}_3 * \mathbf{id}_x$ \$	
Exp	$+ \mathbf{int}_3 * \mathbf{id}_x$ \$	
Exp +	$\mathbf{int}_3 * \mathbf{id}_x$ \$	
Exp + $\mathbf{int}_3$	$* \mathbf{id}_x$ \$	
Exp + Factor	$* \mathbf{id}_x$ \$	
Exp + Term	$* \mathbf{id}_x$ \$	
Exp + Term *	$\mathbf{id}_x$ \$	
Exp + Term * $\mathbf{id}_x$	\$	
Exp + Term * Factor	\$	
Exp + Term	\$	
Exp	\$	
S	\$	

A Rightmost Derivation In Reverse

	int$_2$ + **int**$_3$ * **id**$_x$ $	shift 2
int$_2$	+ **int**$_3$ * **id**$_x$ $	reduce by F → int
Factor	+ **int**$_3$ * **id**$_x$ $	reduce by T → F
Term	+ **int**$_3$ * **id**$_x$ $	reduce by T → E
Exp	+ **int**$_3$ * **id**$_x$ $	
Exp +	**int**$_3$ * **id**$_x$ $	
Exp + **int**$_3$	* **id**$_x$ $	
Exp + Factor	* **id**$_x$ $	
Exp + Term	* **id**$_x$ $	
Exp + Term *	**id**$_x$ $	
Exp + Term * **id**$_x$	$	
Exp + Term * Factor	$	
Exp + Term	$	
Exp	$	
S	$	

A Rightmost Derivation In Reverse

	`int`$_2$ + `int`$_3$ * `id`$_x$ \$	shift 2
`int`$_2$	+ `int`$_3$ * `id`$_x$ \$	reduce by F $\rightarrow$ int
Factor	+ `int`$_3$ * `id`$_x$ \$	reduce by T $\rightarrow$ F
Term	+ `int`$_3$ * `id`$_x$ \$	reduce by T $\rightarrow$ E
Exp	+ `int`$_3$ * `id`$_x$ \$	shift +
Exp +	`int`$_3$ * `id`$_x$ \$	
Exp + `int`$_3$	* `id`$_x$ \$	
Exp + Factor	* `id`$_x$ \$	
Exp + Term	* `id`$_x$ \$	
Exp + Term *	`id`$_x$ \$	
Exp + Term * `id`$_x$	\$	
Exp + Term * Factor	\$	
Exp + Term	\$	
Exp	\$	
S	\$	

A Rightmost Derivation In Reverse

	$\texttt{int}_2 + \texttt{int}_3 * \texttt{id}_x$ \$	shift 2
$\texttt{int}_2$	$+ \texttt{int}_3 * \texttt{id}_x$ \$	reduce by $F \to int$
Factor	$+ \texttt{int}_3 * \texttt{id}_x$ \$	reduce by $T \to F$
Term	$+ \texttt{int}_3 * \texttt{id}_x$ \$	reduce by $T \to E$
Exp	$+ \texttt{int}_3 * \texttt{id}_x$ \$	shift +
Exp +	$\texttt{int}_3 * \texttt{id}_x$ \$	shift 3
Exp + $\texttt{int}_3$	$* \texttt{id}_x$ \$	
Exp + Factor	$* \texttt{id}_x$ \$	
Exp + Term	$* \texttt{id}_x$ \$	
Exp + Term *	$\texttt{id}_x$ \$	
Exp + Term * $\texttt{id}_x$	\$	
Exp + Term * Factor	\$	
Exp + Term	\$	
Exp	\$	
S	\$	

A Rightmost Derivation In Reverse

	int$_2$ + **int**$_3$ * **id**$_x$ \$	shift 2
int$_2$	+ **int**$_3$ * **id**$_x$ \$	reduce by F $\rightarrow$ int
Factor	+ **int**$_3$ * **id**$_x$ \$	reduce by T $\rightarrow$ F
Term	+ **int**$_3$ * **id**$_x$ \$	reduce by T $\rightarrow$ E
Exp	+ **int**$_3$ * **id**$_x$ \$	shift +
Exp +	**int**$_3$ * **id**$_x$ \$	shift 3
Exp + **int**$_3$	* **id**$_x$ \$	reduce by F $\rightarrow$ int
Exp + Factor	* **id**$_x$ \$	
Exp + Term	* **id**$_x$ \$	
Exp + Term *	**id**$_x$ \$	
Exp + Term * **id**$_x$	\$	
Exp + Term * Factor	\$	
Exp + Term	\$	
Exp	\$	
S	\$	

A Rightmost Derivation In Reverse

	$\mathtt{int}_2 + \mathtt{int}_3 * \mathtt{id}_x$ \$	shift 2
$\mathtt{int}_2$	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$	reduce by F $\rightarrow$ int
Factor	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$	reduce by T $\rightarrow$ F
Term	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$	reduce by T $\rightarrow$ E
Exp	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$	shift +
Exp +	$\mathtt{int}_3 * \mathtt{id}_x$ \$	shift 3
Exp + $\mathtt{int}_3$	$* \mathtt{id}_x$ \$	reduce by F $\rightarrow$ int
Exp + Factor	$* \mathtt{id}_x$ \$	reduce by F $\rightarrow$ T
Exp + Term	$* \mathtt{id}_x$ \$	
Exp + Term *	$\mathtt{id}_x$ \$	
Exp + Term * $\mathtt{id}_x$	\$	
Exp + Term * Factor	\$	
Exp + Term	\$	
Exp	\$	
S	\$	

A Rightmost Derivation In Reverse

	$\texttt{int}_2 + \texttt{int}_3 * \texttt{id}_x$ \$	shift 2
$\texttt{int}_2$	$+ \texttt{int}_3 * \texttt{id}_x$ \$	reduce by F $\rightarrow$ int
Factor	$+ \texttt{int}_3 * \texttt{id}_x$ \$	reduce by T $\rightarrow$ F
Term	$+ \texttt{int}_3 * \texttt{id}_x$ \$	reduce by T $\rightarrow$ E
Exp	$+ \texttt{int}_3 * \texttt{id}_x$ \$	shift +
Exp +	$\texttt{int}_3 * \texttt{id}_x$ \$	shift 3
Exp + $\texttt{int}_3$	$* \texttt{id}_x$ \$	reduce by F $\rightarrow$ int
Exp + Factor	$* \texttt{id}_x$ \$	reduce by F $\rightarrow$ T
Exp + Term	$* \texttt{id}_x$ \$	shift *
Exp + Term *	$\texttt{id}_x$ \$	
Exp + Term * $\texttt{id}_x$	\$	
Exp + Term * Factor	\$	
Exp + Term	\$	
Exp	\$	
S	\$	

A Rightmost Derivation In Reverse

	$\texttt{int}_2 + \texttt{int}_3 * \texttt{id}_x$ \$	shift 2
$\texttt{int}_2$	$+ \texttt{int}_3 * \texttt{id}_x$ \$	reduce by F $\rightarrow$ int
Factor	$+ \texttt{int}_3 * \texttt{id}_x$ \$	reduce by T $\rightarrow$ F
Term	$+ \texttt{int}_3 * \texttt{id}_x$ \$	reduce by T $\rightarrow$ E
Exp	$+ \texttt{int}_3 * \texttt{id}_x$ \$	shift +
Exp +	$\texttt{int}_3 * \texttt{id}_x$ \$	shift 3
Exp + $\texttt{int}_3$	$* \texttt{id}_x$ \$	reduce by F $\rightarrow$ int
Exp + Factor	$* \texttt{id}_x$ \$	reduce by F $\rightarrow$ T
Exp + Term	$* \texttt{id}_x$ \$	shift *
Exp + Term *	$\texttt{id}_x$ \$	shift x
Exp + Term * $\texttt{id}_x$	\$	
Exp + Term * Factor	\$	
Exp + Term	\$	
Exp	\$	
S	\$	

A Rightmost Derivation In Reverse

	$\texttt{int}_2 + \texttt{int}_3 * \texttt{id}_x$ \$	shift 2
$\texttt{int}_2$	$+ \texttt{int}_3 * \texttt{id}_x$ \$	reduce by F $\rightarrow$ int
Factor	$+ \texttt{int}_3 * \texttt{id}_x$ \$	reduce by T $\rightarrow$ F
Term	$+ \texttt{int}_3 * \texttt{id}_x$ \$	reduce by T $\rightarrow$ E
Exp	$+ \texttt{int}_3 * \texttt{id}_x$ \$	shift +
Exp +	$\texttt{int}_3 * \texttt{id}_x$ \$	shift 3
Exp + $\texttt{int}_3$	$* \texttt{id}_x$ \$	reduce by F $\rightarrow$ int
Exp + Factor	$* \texttt{id}_x$ \$	reduce by F $\rightarrow$ T
Exp + Term	$* \texttt{id}_x$ \$	shift *
Exp + Term *	$\texttt{id}_x$ \$	shift x
Exp + Term * $\texttt{id}_x$	\$	reduce by F $\rightarrow$ id
Exp + Term * Factor	\$	
Exp + Term	\$	
Exp	\$	
S	\$	

A Rightmost Derivation In Reverse

	`int`$_2$ + `int`$_3$ * `id`$_x$ \$	shift 2
`int`$_2$	+ `int`$_3$ * `id`$_x$ \$	reduce by F $\rightarrow$ int
Factor	+ `int`$_3$ * `id`$_x$ \$	reduce by T $\rightarrow$ F
Term	+ `int`$_3$ * `id`$_x$ \$	reduce by T $\rightarrow$ E
Exp	+ `int`$_3$ * `id`$_x$ \$	shift +
Exp +	`int`$_3$ * `id`$_x$ \$	shift 3
Exp + `int`$_3$	* `id`$_x$ \$	reduce by F $\rightarrow$ int
Exp + Factor	* `id`$_x$ \$	reduce by F $\rightarrow$ T
Exp + Term	* `id`$_x$ \$	shift *
Exp + Term *	`id`$_x$ \$	shift x
Exp + Term * `id`$_x$	\$	reduce by F $\rightarrow$ id
Exp + Term * Factor	\$	reduce by T $\rightarrow$ T * F
Exp + Term	\$	
Exp	\$	
S	\$	

A Rightmost Derivation In Reverse

	$\texttt{int}_2 + \texttt{int}_3 * \texttt{id}_x \ \$$	shift 2
$\texttt{int}_2$	$+ \texttt{int}_3 * \texttt{id}_x \ \$$	reduce by F $\to$ int
Factor	$+ \texttt{int}_3 * \texttt{id}_x \ \$$	reduce by T $\to$ F
Term	$+ \texttt{int}_3 * \texttt{id}_x \ \$$	reduce by T $\to$ E
Exp	$+ \texttt{int}_3 * \texttt{id}_x \ \$$	shift +
Exp +	$\texttt{int}_3 * \texttt{id}_x \ \$$	shift 3
Exp + $\texttt{int}_3$	$* \texttt{id}_x \ \$$	reduce by F $\to$ int
Exp + Factor	$* \texttt{id}_x \ \$$	reduce by F $\to$ T
Exp + Term	$* \texttt{id}_x \ \$$	shift *
Exp + Term *	$\texttt{id}_x \ \$$	shift x
Exp + Term * $\texttt{id}_x$	$\$$	reduce by F $\to$ id
Exp + Term * Factor	$\$$	reduce by T $\to$ T * F
Exp + Term	$\$$	reduce by E $\to$ E + T
Exp	$\$$	
S	$\$$	

A Rightmost Derivation In Reverse

	$\texttt{int}_2 + \texttt{int}_3 * \texttt{id}_x$ \$	shift 2
$\texttt{int}_2$	$+ \texttt{int}_3 * \texttt{id}_x$ \$	reduce by F → int
Factor	$+ \texttt{int}_3 * \texttt{id}_x$ \$	reduce by T → F
Term	$+ \texttt{int}_3 * \texttt{id}_x$ \$	reduce by T → E
Exp	$+ \texttt{int}_3 * \texttt{id}_x$ \$	shift +
Exp +	$\texttt{int}_3 * \texttt{id}_x$ \$	shift 3
Exp + $\texttt{int}_3$	$* \texttt{id}_x$ \$	reduce by F → int
Exp + Factor	$* \texttt{id}_x$ \$	reduce by F → T
Exp + Term	$* \texttt{id}_x$ \$	shift *
Exp + Term *	$\texttt{id}_x$ \$	shift x
Exp + Term * $\texttt{id}_x$	\$	reduce by F → id
Exp + Term * Factor	\$	reduce by T → T * F
Exp + Term	\$	reduce by E → E + T
Exp	\$	reduce by S → E
S	\$	

A Rightmost Derivation In Reverse

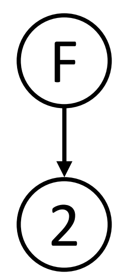
	`int`$_2$ + `int`$_3$ * `id`$_x$ $	shift 2
`int`$_2$	+ `int`$_3$ * `id`$_x$ $	reduce by F → int
Factor	+ `int`$_3$ * `id`$_x$ $	reduce by T → F
Term	+ `int`$_3$ * `id`$_x$ $	reduce by T → E
Exp	+ `int`$_3$ * `id`$_x$ $	shift +
Exp +	`int`$_3$ * `id`$_x$ $	shift 3
Exp + `int`$_3$	* `id`$_x$ $	reduce by F → int
Exp + Factor	* `id`$_x$ $	reduce by F → T
Exp + Term	* `id`$_x$ $	shift *
Exp + Term *	`id`$_x$ $	shift x
Exp + Term * `id`$_x$	$	reduce by F → id
Exp + Term * Factor	$	reduce by T → T * F
Exp + Term	$	reduce by E → E + T
Exp	$	reduce by S → E
S	$	accept!

A Rightmost Derivation In Reverse

	$\texttt{int}_2 + \texttt{int}_3 * \texttt{id}_x$ $	shift 2
$\texttt{int}_2$	$+ \texttt{int}_3 * \texttt{id}_x$ $	
Factor	$+ \texttt{int}_3 * \texttt{id}_x$ $	
Term	$+ \texttt{int}_3 * \texttt{id}_x$ $	
Exp	$+ \texttt{int}_3 * \texttt{id}_x$ $	
Exp +	$\texttt{int}_3 * \texttt{id}_x$ $	
Exp + $\texttt{int}_3$	$* \texttt{id}_x$ $	
Exp + Factor	$* \texttt{id}_x$ $	
Exp + Term	$* \texttt{id}_x$ $	
Exp + Term *	$\texttt{id}_x$ $	
Exp + Term * $\texttt{id}_x$	$	
Exp + Term * Factor	$	
Exp + Term	$	
Exp	$	2
S	$	

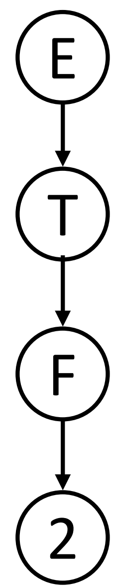
A Rightmost Derivation In Reverse

	int$_2$ + **int**$_3$ * **id**$_x$ \$	shift 2
int$_2$	+ **int**$_3$ * **id**$_x$ \$	reduce by F $\rightarrow$ int
Factor	+ **int**$_3$ * **id**$_x$ \$	
Term	+ **int**$_3$ * **id**$_x$ \$	
Exp	+ **int**$_3$ * **id**$_x$ \$	
Exp +	**int**$_3$ * **id**$_x$ \$	
Exp + **int**$_3$	* **id**$_x$ \$	
Exp + Factor	* **id**$_x$ \$	
Exp + Term	* **id**$_x$ \$	
Exp + Term *	**id**$_x$ \$	
Exp + Term * **id**$_x$	\$	
Exp + Term * Factor	\$	
Exp + Term	\$	
Exp	\$	
S	\$	



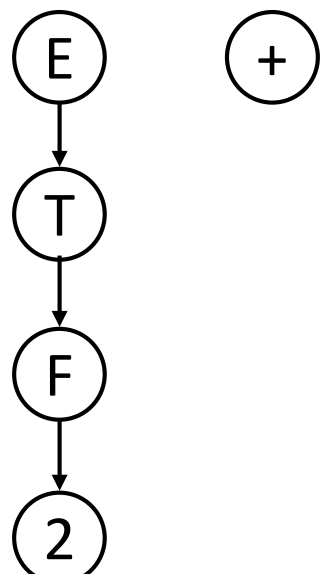
A Rightmost Derivation In Reverse

	$\mathtt{int}_2 + \mathtt{int}_3 * \mathtt{id}_x$ \$	shift 2
$\mathtt{int}_2$	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$	reduce by F $\rightarrow$ int
Factor	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$	reduce by T $\rightarrow$ F
Term	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$	reduce by T $\rightarrow$ E
Exp	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$	
Exp +	$\mathtt{int}_3 * \mathtt{id}_x$ \$	
Exp + $\mathtt{int}_3$	$* \mathtt{id}_x$ \$	
Exp + Factor	$* \mathtt{id}_x$ \$	
Exp + Term	$* \mathtt{id}_x$ \$	
Exp + Term *	$\mathtt{id}_x$ \$	
Exp + Term * $\mathtt{id}_x$	\$	
Exp + Term * Factor	\$	
Exp + Term	\$	
Exp	\$	
S	\$	



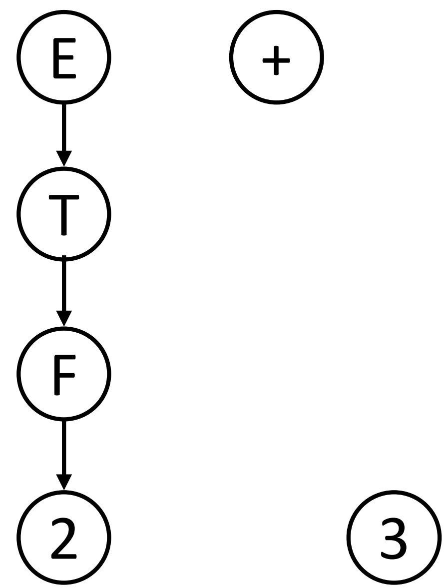
A Rightmost Derivation In Reverse

	$\mathtt{int}_2 + \mathtt{int}_3 * \mathtt{id}_x$ \$	shift 2
$\mathtt{int}_2$	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$	reduce by F → int
Factor	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$	reduce by T → F
Term	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$	reduce by T → E
Exp	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$	shift +
Exp +	$\mathtt{int}_3 * \mathtt{id}_x$ \$	
Exp + $\mathtt{int}_3$	$* \mathtt{id}_x$ \$	
Exp + Factor	$* \mathtt{id}_x$ \$	
Exp + Term	$* \mathtt{id}_x$ \$	
Exp + Term *	$\mathtt{id}_x$ \$	
Exp + Term * $\mathtt{id}_x$	\$	
Exp + Term * Factor	\$	
Exp + Term	\$	
Exp	\$	
S	\$	



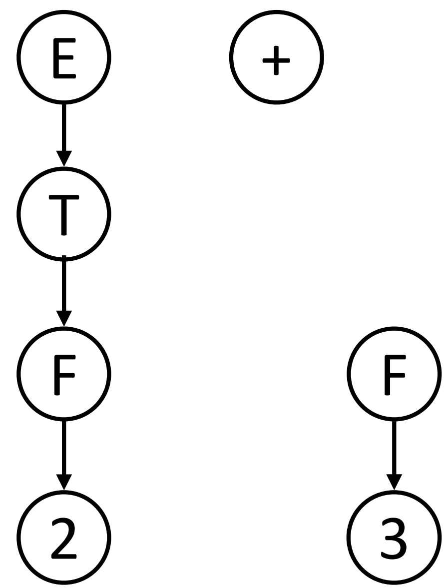
A Rightmost Derivation In Reverse

	int$_2$ + **int**$_3$ * **id**$_x$ \$	shift 2
int$_2$	+ **int**$_3$ * **id**$_x$ \$	reduce by F $\rightarrow$ int
Factor	+ **int**$_3$ * **id**$_x$ \$	reduce by T $\rightarrow$ F
Term	+ **int**$_3$ * **id**$_x$ \$	reduce by T $\rightarrow$ E
Exp	+ **int**$_3$ * **id**$_x$ \$	shift +
Exp +	**int**$_3$ * **id**$_x$ \$	shift 3
Exp + **int**$_3$	* **id**$_x$ \$	
Exp + Factor	* **id**$_x$ \$	
Exp + Term	* **id**$_x$ \$	
Exp + Term *	**id**$_x$ \$	
Exp + Term * **id**$_x$	\$	
Exp + Term * Factor	\$	
Exp + Term	\$	
Exp	\$	
S	\$	



A Rightmost Derivation In Reverse

	$\mathtt{int}_2 + \mathtt{int}_3 * \mathtt{id}_x$ \$	shift 2
$\mathtt{int}_2$	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$	reduce by F $\rightarrow$ int
Factor	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$	reduce by T $\rightarrow$ F
Term	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$	reduce by T $\rightarrow$ E
Exp	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$	shift +
Exp +	$\mathtt{int}_3 * \mathtt{id}_x$ \$	shift 3
Exp + $\mathtt{int}_3$	$* \mathtt{id}_x$ \$	reduce by F $\rightarrow$ int
Exp + Factor	$* \mathtt{id}_x$ \$	
Exp + Term	$* \mathtt{id}_x$ \$	
Exp + Term *	$\mathtt{id}_x$ \$	
Exp + Term * $\mathtt{id}_x$	\$	
Exp + Term * Factor	\$	
Exp + Term	\$	
Exp	\$	
S	\$	



Handles

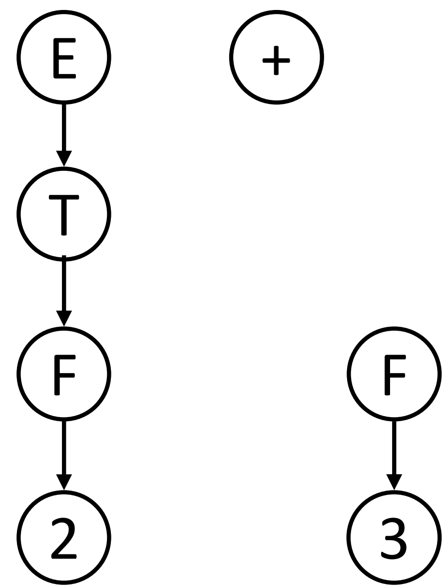
- LR parsing is handle pruning
- LR parsing finds a rightmost derivation (in reverse)
- A handle in γ, a right-hand sentential form, is
  - a position in γ matching β
  - a production $A \rightarrow \beta$

$$S \rightarrow^* \alpha A w \rightarrow \alpha \beta w$$

- if a grammar is unambiguous, then every γ has exactly 1 handle

A Rightmost Derivation In Reverse

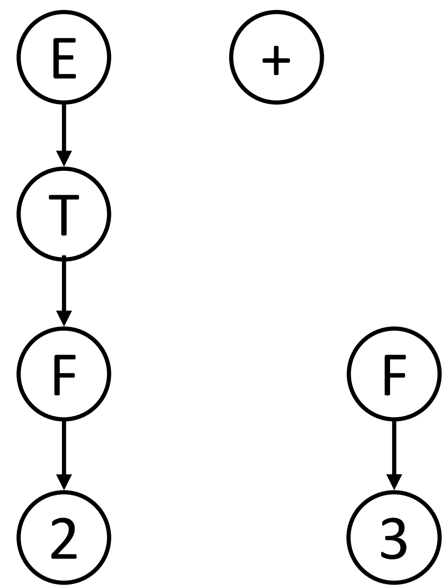
	$\texttt{int}_2 + \texttt{int}_3 * \texttt{id}_x \ \$$	shift 2
$\texttt{int}_2$	$+ \texttt{int}_3 * \texttt{id}_x \ \$$	reduce by F $\rightarrow$ int
Factor	$+ \texttt{int}_3 * \texttt{id}_x \ \$$	reduce by T $\rightarrow$ F
Term	$+ \texttt{int}_3 * \texttt{id}_x \ \$$	reduce by T $\rightarrow$ E
Exp	$+ \texttt{int}_3 * \texttt{id}_x \ \$$	shift +
Exp +	$\texttt{int}_3 * \texttt{id}_x \ \$$	shift 3
Exp + $\texttt{int}_3$	$* \texttt{id}_x \ \$$	reduce by F $\rightarrow$ int
Exp + Factor	$* \texttt{id}_x \ \$$	
Exp + Term	$* \texttt{id}_x \ \$$	
Exp + Term *	$\texttt{id}_x \ \$$	
Exp + Term * $\texttt{id}_x$	$\$$	
Exp + Term * Factor	$\$$	
Exp + Term	$\$$	
Exp	$\$$	
S	$\$$	



A Rightmost Derivation In Reverse

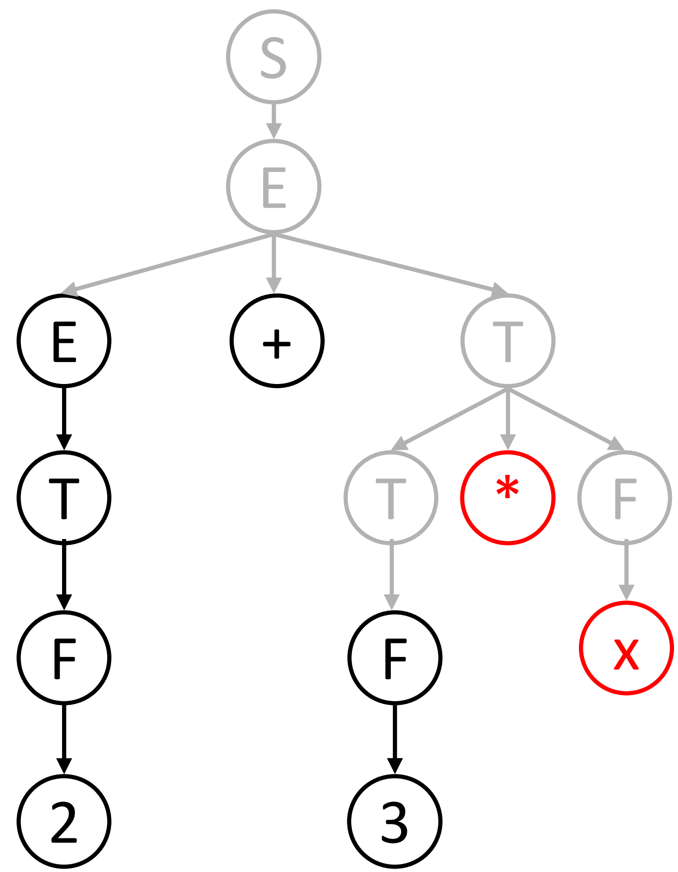
Where is next handle?

	$\mathbf{int}_2 + \mathbf{int}_3 * \mathbf{id}_x$ $	shift 2
$\mathbf{int}_2$	$+ \mathbf{int}_3 * \mathbf{id}_x$ $	reduce by $F \to$ int
Factor	$+ \mathbf{int}_3 * \mathbf{id}_x$ $	reduce by $T \to F$
Term	$+ \mathbf{int}_3 * \mathbf{id}_x$ $	reduce by $T \to E$
Exp	$+ \mathbf{int}_3 * \mathbf{id}_x$ $	shift +
Exp +	$\mathbf{int}_3 * \mathbf{id}_x$ $	shift 3
Exp + $\mathbf{int}_3$	$* \mathbf{id}_x$ $	reduce by $F \to$ int
Exp + Factor	$* \mathbf{id}_x$ $	
Exp + Term	$* \mathbf{id}_x$ $	
Exp + Term *	$\mathbf{id}_x$ $	
Exp + Term * $\mathbf{id}_x$	$	
Exp + Term * Factor	$	
Exp + Term	$	
Exp	$	
S	$	



A Rightmost Derivation In Reverse

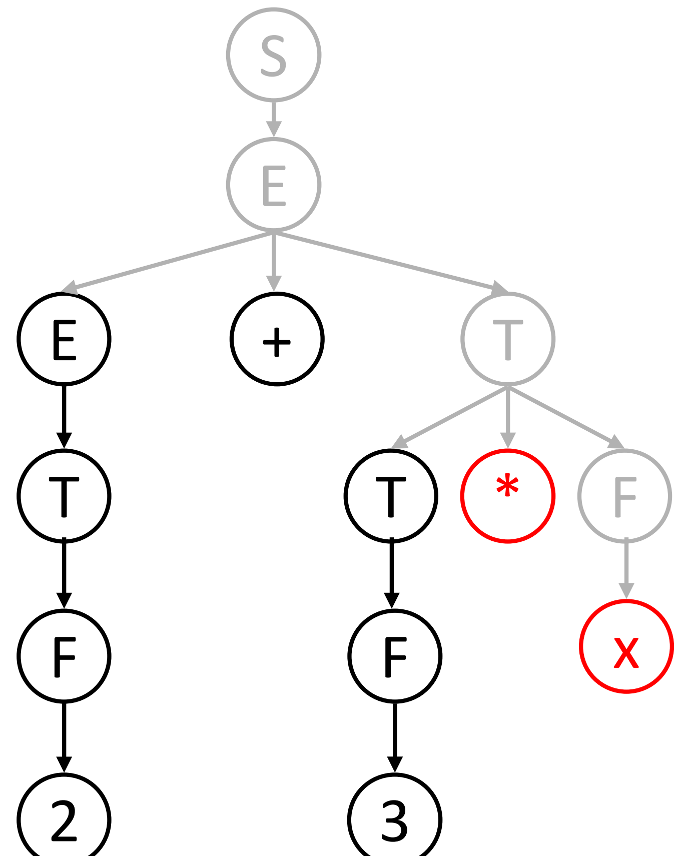
Where is next handle?	$\mathtt{int}_2 + \mathtt{int}_3 * \mathtt{id}_x \ \$$
$\mathtt{int}_2$	$+ \mathtt{int}_3 * \mathtt{id}_x \ \$$
Factor	$+ \mathtt{int}_3 * \mathtt{id}_x \ \$$
Term	$+ \mathtt{int}_3 * \mathtt{id}_x \ \$$
Exp	$+ \mathtt{int}_3 * \mathtt{id}_x \ \$$
Exp +	$\mathtt{int}_3 * \mathtt{id}_x \ \$$
Exp + $\mathtt{int}_3$	$* \mathtt{id}_x \ \$$
Exp + Factor	$* \mathtt{id}_x \ \$$
Exp + Term	$* \mathtt{id}_x \ \$$
Exp + Term *	$\mathtt{id}_x \ \$$
Exp + Term * $\mathtt{id}_x$	$\$$
Exp + Term * Factor	$\$$
Exp + Term	$\$$
Exp	$\$$
S	$\$$



A Rightmost Derivation In Reverse

Where is next handle? E+F*x and T$\rightarrow$ F $_x$ $

int$_2$	+ **int**$_3$ * **id**$_x$ $
Factor	+ **int**$_3$ * **id**$_x$ $
Term	+ **int**$_3$ * **id**$_x$ $
Exp	+ **int**$_3$ * **id**$_x$ $
Exp +	**int**$_3$ * **id**$_x$ $
Exp + **int**$_3$	* **id**$_x$ $
Exp + Factor	* **id**$_x$ $
Exp + Term	* **id**$_x$ $
Exp + Term *	**id**$_x$ $
Exp + Term * **id**$_x$	$
Exp + Term * Factor	$
Exp + Term	$
Exp	$
S	$



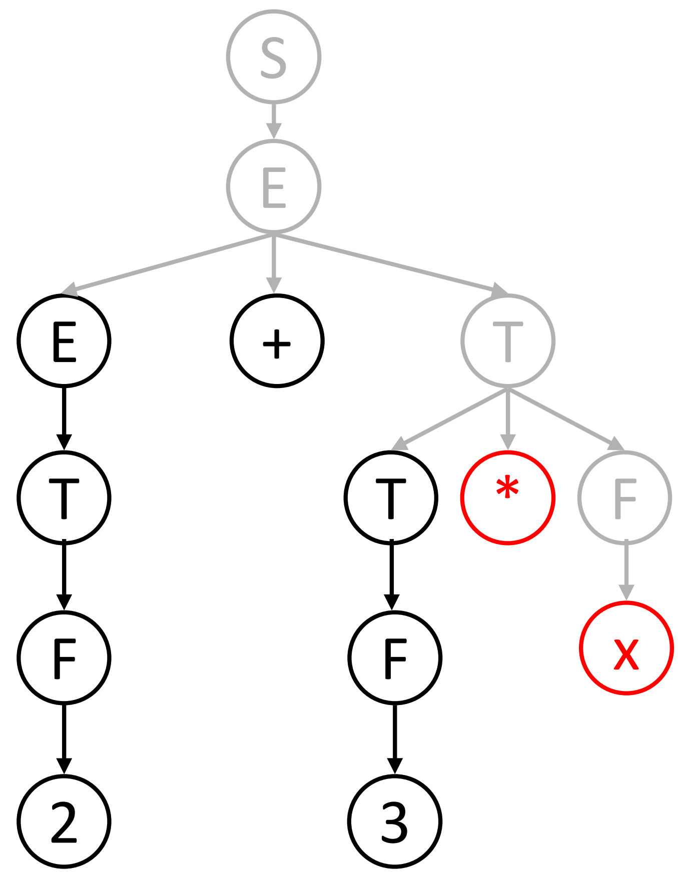
Handle Pruning

- LR parsing consists of
  - shifting til there is a handle on the top of the stack
  - reducing handle
- Key is handle is always on top of stack, i.e., if β is a handle with $A \rightarrow \beta$, then β can be found on top of stack.

A Rightmost Derivation In Reverse

	$\texttt{int}_2 + \texttt{int}_3 * \texttt{id}_x$ \$
$\texttt{int}_2$	$+ \texttt{int}_3 * \texttt{id}_x$ \$
Factor	$+ \texttt{int}_3 * \texttt{id}_x$ \$
Term	$+ \texttt{int}_3 * \texttt{id}_x$ \$
Exp	$+ \texttt{int}_3 * \texttt{id}_x$ \$
Exp +	$\texttt{int}_3 * \texttt{id}_x$ \$
Exp + $\texttt{int}_3$	$* \texttt{id}_x$ \$
Exp + Factor	$* \texttt{id}_x$ \$
Exp + Term	$* \texttt{id}_x$ \$
Exp + Term *	$\texttt{id}_x$ \$
Exp + Term * $\texttt{id}_x$	\$
Exp + Term * Factor	\$
Exp + Term	\$
Exp	\$
S	\$

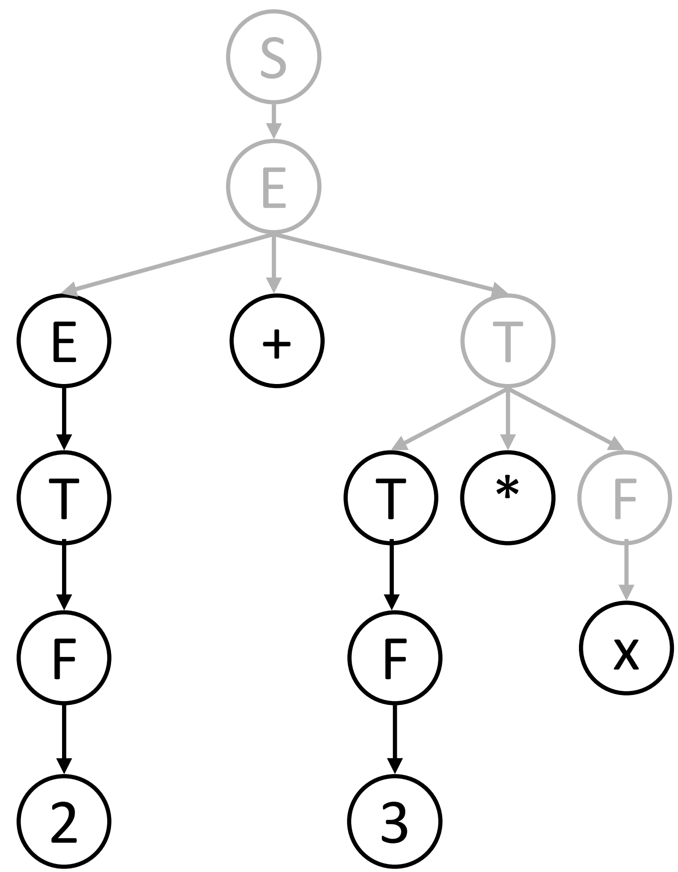
top of stack does not have a handle, so must shift.



A Rightmost Derivation In Reverse

Now, x is a handle.

	$\mathtt{int}_2 + \mathtt{int}_3 * \mathtt{id}_x$ \$
$\mathtt{int}_2$	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$
Factor	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$
Term	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$
Exp	$+ \mathtt{int}_3 * \mathtt{id}_x$ \$
Exp +	$\mathtt{int}_3 * \mathtt{id}_x$ \$
Exp + $\mathtt{int}_3$	$* \mathtt{id}_x$ \$
Exp + Factor	$* \mathtt{id}_x$ \$
Exp + Term	$* \mathtt{id}_x$ \$
Exp + Term *	$\mathtt{id}_x$ \$
Exp + Term * $\mathtt{id}_x$	\$
Exp + Term * Factor	\$
Exp + Term	\$
Exp	\$
S	\$

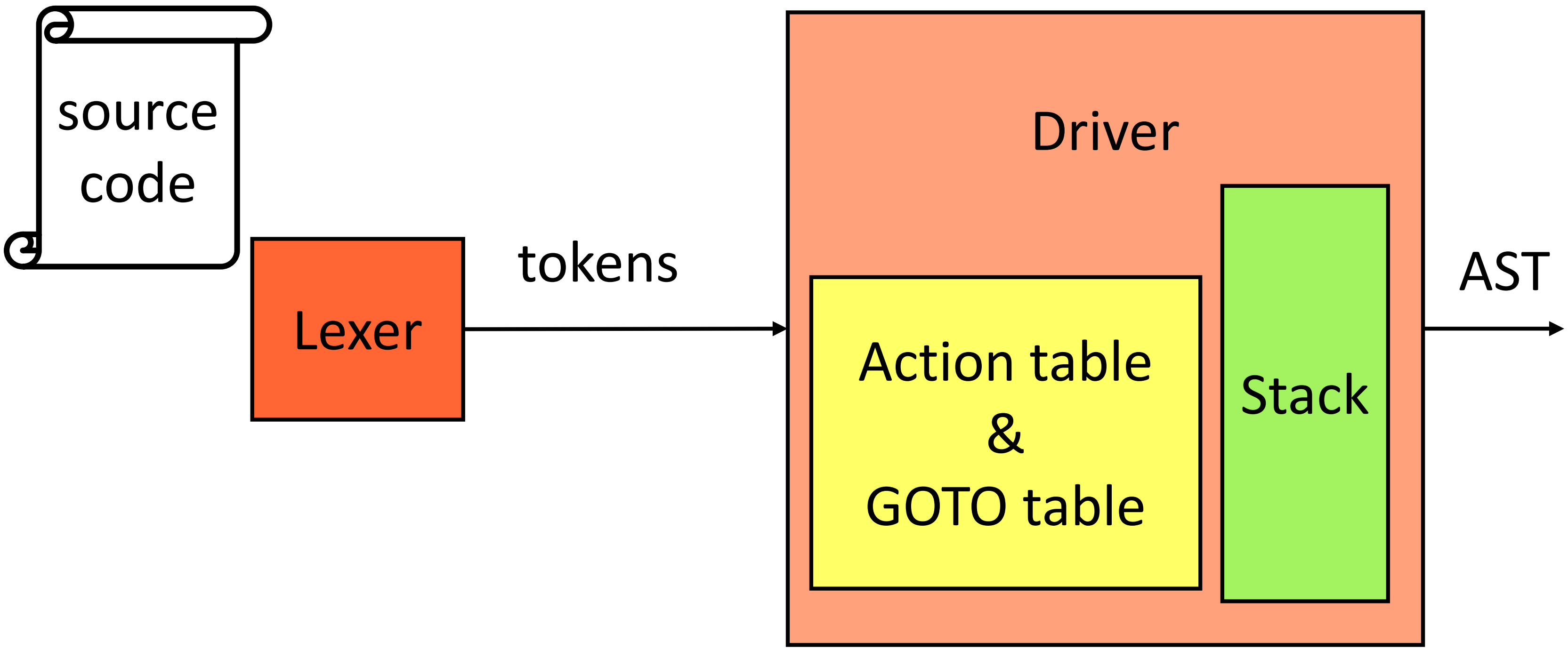


A Shift-Reduce Parser

- Stack holds the viable prefixes.
- input stream holds remaining source
- Four actions:
  - shift: push token from input stream onto stack
  - reduce: right-end of a handle (β of $A \rightarrow \beta$) is at top of stack, pop handle (β), push A
  - accept: success
  - error: syntax error discovered

Key is recognizing handles efficiently

Table-driven LR(k) parsers



Push down automata:
FSM with stack

Parser Loop

Driver

- Same code regardless of grammar
  - only tables change
- (Very) General Algorithm:
  - Based on table contents, top of stack, and current input character either
    - **shift**: pushes onto stack, reads next token
    - **reduce**: manipulate stack to simplify representation of already scanned input
    - **accept**: successfully scanned entire input
    - **error**: input not in language

Stack

Stack

- Represents the scanned input
- Contents?
  - Reduced nonterminals not enough
  - Must store previously seen *states*
    - the context of the current position
  - In fact, nonterminals unnecessary
    - include for readability

$x + y\bullet + z$

T

+

T

Parser Tables

Action table
&
GOTO table

Action table

- given state *s* and **terminal** *a* tells parser loop what action (shift, reduce, accept, reject) to perform

Goto table

- used when performing reduction; given a state *s* and **nonterminal** *X* says what state to transition to

Parser Tables

Action table
&
GOTO table

s*N* push state *N* onto stack
r*R* reduce by rule *R*
g*N* goto state *N*
a accept
error

state	action			goto	
	ident	+	$	E	T
0	s3			g1	g2
1			a		
2		s4	r2		
3		r3	r3		
4	s3			g5	g2
5			r1		

Parser Loop Revisited

Driver

```
while(true)
   s = state on top of stack
   a = current input token
   if(action[s][a] == sN)                    shift
      push N
      read next input token
   else if(action[s][a] == rR)               reduce
      pop rhs of rule R from stack
      X = lhs of rule R
      N = state on top of stack
      push goto[N][X]
   else if(action[s][a] == a)                accept
      return success
   else                                      error
      return failure
```

Example

state	action			goto	
	ident	+	$	E	T
0	**s3**			**g1**	**g2**
1			**a**		
2		**s4**	**r2**		
3		**r3**	**r3**		
4	**s3**			**g5**	**g2**
5			**r1**		

0 $S \rightarrow E\$$

1 $E \rightarrow T + E$

2 $E \rightarrow T$

3 $T \rightarrow \textit{identifier}$

Current input token = **x**

State on top of the stack = **0**

x + y$

(0,S)

Stack

Example

state	action			goto	
	ident	+	$	E	T
0	s3			g1	g2
1			a		
2		s4	r2		
3		r3	r3		
4	s3			g5	g2
5			r1		

0 $S \rightarrow E\$$

1 $E \rightarrow T + E$

2 $E \rightarrow T$

3 $T \rightarrow \textit{identifier}$

Current input token = **+**

State on top of the stack = **3**

(3,x)
(0,S)

x + y$

Example

state	action			goto	
	ident	+	$	E	T
0	s3			g1	g2
1			a		
2		s4	r2		
3		r3	r3		
4	s3			g5	g2
5			r1		

0 $S \rightarrow E\$$

1 $E \rightarrow T + E$

2 $E \rightarrow T$

3 $T \rightarrow \mathit{identifier}$

Current input token = **+**

State on top of the stack = **3**

(3,x)
(0,S)

x + y$

Example

state	action			goto	
	ident	+	$	E	T
0	s3			g1	g2
1			a		
2		s4	r2		
3		r3	r3		
4	s3			g5	g2
5			r1		

0 $S \rightarrow E\$$

1 $E \rightarrow T + E$

2 $E \rightarrow T$

3 $T \rightarrow \mathit{identifier}$

Current input token = **+**

State on top of the stack = **3**

(3,x)

(0,S)

x + y$

Example

state	action			goto	
	ident	+	$	E	T
0	s3			g1	g2
1			a		
2		s4	r2		
3		r3	r3		
4	s3			g5	g2
5			r1		

0 $S \rightarrow E\$$

1 $E \rightarrow T + E$

2 $E \rightarrow T$

3 $T \rightarrow \textit{identifier}$

Current input token = **+**

State on top of the stack = **0**

(3,x)

(0,S)

x + y$

Example

state	action			goto	
	ident	+	$	E	T
0	s3			g1	g2
1			a		
2		s4	r2		
3		r3	r3		
4	s3			g5	g2
5			r1		

0 $S \to E\$$
1 $E \to T + E$
2 $E \to T$
3 $T \to \mathit{identifier}$

Current input token = **+**
State on top of the stack = **2**

(2,T)
(0,S)

x + y$

Example

state	action			goto	
	ident	+	$	E	T
0	s3			g1	g2
1			a		
2		s4	r2		
3		r3	r3		
4	s3			g5	g2
5			r1		

0 $S \rightarrow E\$$

1 $E \rightarrow T + E$

2 $E \rightarrow T$

3 $T \rightarrow \textit{identifier}$

Current input token = **+**

State on top of the stack = **2**

(2,T)
(0,S)

x + y$

Example

state	action			goto	
	ident	+	$	E	T
0	**s3**			**g1**	**g2**
1			**a**		
2		**s4**	**r2**		
3		**r3**	**r3**		
4	**s3**			**g5**	**g2**
5			**r1**		

0 $S \rightarrow E\$$

1 $E \rightarrow T + E$

2 $E \rightarrow T$

3 $T \rightarrow \textit{identifier}$

Current input token = **y**

State on top of the stack = **4**

(4,+)
(2,T)
(0,S)

x + y$

Example

state	action			goto	
	ident	+	$	E	T
0	s3			g1	g2
1			a		
2		s4	r2		
3		r3	r3		
4	s3			g5	g2
5			r1		

0 $S \rightarrow E\$$

1 $E \rightarrow T + E$

2 $E \rightarrow T$

3 $T \rightarrow \textit{identifier}$

Current input token = **y**

State on top of the stack = **4**

(4,+)
(2,T)
(0,S)

x + y$

Example

state	action			goto	
	ident	+	$	E	T
0	s3			g1	g2
1			a		
2		s4	r2		
3		r3	r3		
4	s3			g5	g2
5			r1		

0 $S \rightarrow E\$$

1 $E \rightarrow T + E$

2 $E \rightarrow T$

3 $T \rightarrow$ *identifier*

Current input token = **$**

State on top of the stack = **3**

(3,y)
(4,+)
(2,T)
(0,S)

x + y$

Example

state	action			goto	
	ident	+	$	E	T
0	s3			g1	g2
1			a		
2		s4	r2		
3		r3	r3		
4	s3			g5	g2
5			r1		

0 $S \rightarrow E\$$

1 $E \rightarrow T + E$

2 $E \rightarrow T$

3 $T \rightarrow \textit{identifier}$

Current input token = **$**

State on top of the stack = **3**

(?,T)

(4,+)

(2,T)

(0,S)

x + y$

Example

state	action			goto	
	ident	+	$	E	T
0	s3			g1	g2
1			a		
2		s4	r2		
3		r3	r3		
4	s3			g5	g2
5			r1		

0 $S \rightarrow E\$$

1 $E \rightarrow T + E$

2 $E \rightarrow T$

3 $T \rightarrow \textit{identifier}$

Current input token = **$**

State on top of the stack = **2**

(2,T)
(4,+)
(2,T)
(0,S)

x + y$

Example

state	action			goto	
	ident	+	$	E	T
0	s3			g1	g2
1			a		
2		s4	r2		
3		r3	r3		
4	s3			g5	g2
5			r1		

0 $S \rightarrow E\$$

1 $E \rightarrow T + E$

2 $E \rightarrow T$

3 $T \rightarrow \textit{identifier}$

Current input token = **$**

State on top of the stack = **2**

(2,T)
(4,+)
(2,T)
(0,S)

x + y$

Example

state	action			goto	
	ident	+	$	E	T
0	s3			g1	g2
1			a		
2		s4	r2		
3		r3	r3		
4	s3			g5	g2
5			r1		

0 $S \rightarrow E\$$
1 $E \rightarrow T + E$
2 $E \rightarrow T$
3 $T \rightarrow identifier$

Current input token = **$**
State on top of the stack = **2**

(?,E)

(4,+)
(2,T)
(0,S)

x + y$

Example

state	action			goto	
	ident	+	$	E	T
0	s3			g1	g2
1			a		
2		s4	r2		
3		r3	r3		
4	s3			g5	g2
5			r1		

0 $S \rightarrow E\$$

1 $E \rightarrow T + E$

2 $E \rightarrow T$

3 $T \rightarrow \textit{identifier}$

Current input token = **$**

State on top of the stack = **5**

(5,E)
(4,+)
(2,T)
(0,S)

x + y$

Example

state	action			goto	
	ident	+	$	E	T
0	s3			g1	g2
1			a		
2		s4	r2		
3		r3	r3		
4	s3			g5	g2
5			r1		

0 $S \rightarrow E\$$

1 $E \rightarrow T + E$

2 $E \rightarrow T$

3 $T \rightarrow \textit{identifier}$

Current input token = **$**

State on top of the stack = **5**

(5,E)
(4,+)
(2,T)
(0,S)

x + y$

Example

	action			goto	
state	*ident*	+	$	E	T
0	s3			g1	g2
1			a		
2		s4	r2		
3		r3	r3		
4	s3			g5	g2
5			r1		

0 $S \to E\$$
1 $E \to T + E$
2 $E \to T$
3 $T \to \textit{identifier}$

Current input token = **$**
State on top of the stack = **5**

(5,E)
(4,+)
(2,T)
(0,S)

x + y$

Example

state	action			goto	
	ident	+	$	E	T
0	s3			g1	g2
1			a		
2		s4	r2		
3		r3	r3		
4	s3			g5	g2
5			r1		

0 $S \to E\$$

1 $E \to T + E$

2 $E \to T$

3 $T \to \textit{identifier}$

Current input token = **$**

State on top of the stack = **1**

(1,E)

(0,S)

x + y$

Example

state	action			goto	
	ident	+	$	E	T
0	s3			g1	g2
1			a		
2		s4	r2		
3		r3	r3		
4	s3			g5	g2
5			r1		

Accept!

0 $S \rightarrow E\$$

1 $E \rightarrow T + E$

2 $E \rightarrow T$

3 $T \rightarrow \mathit{identifier}$

Current input token = **$**

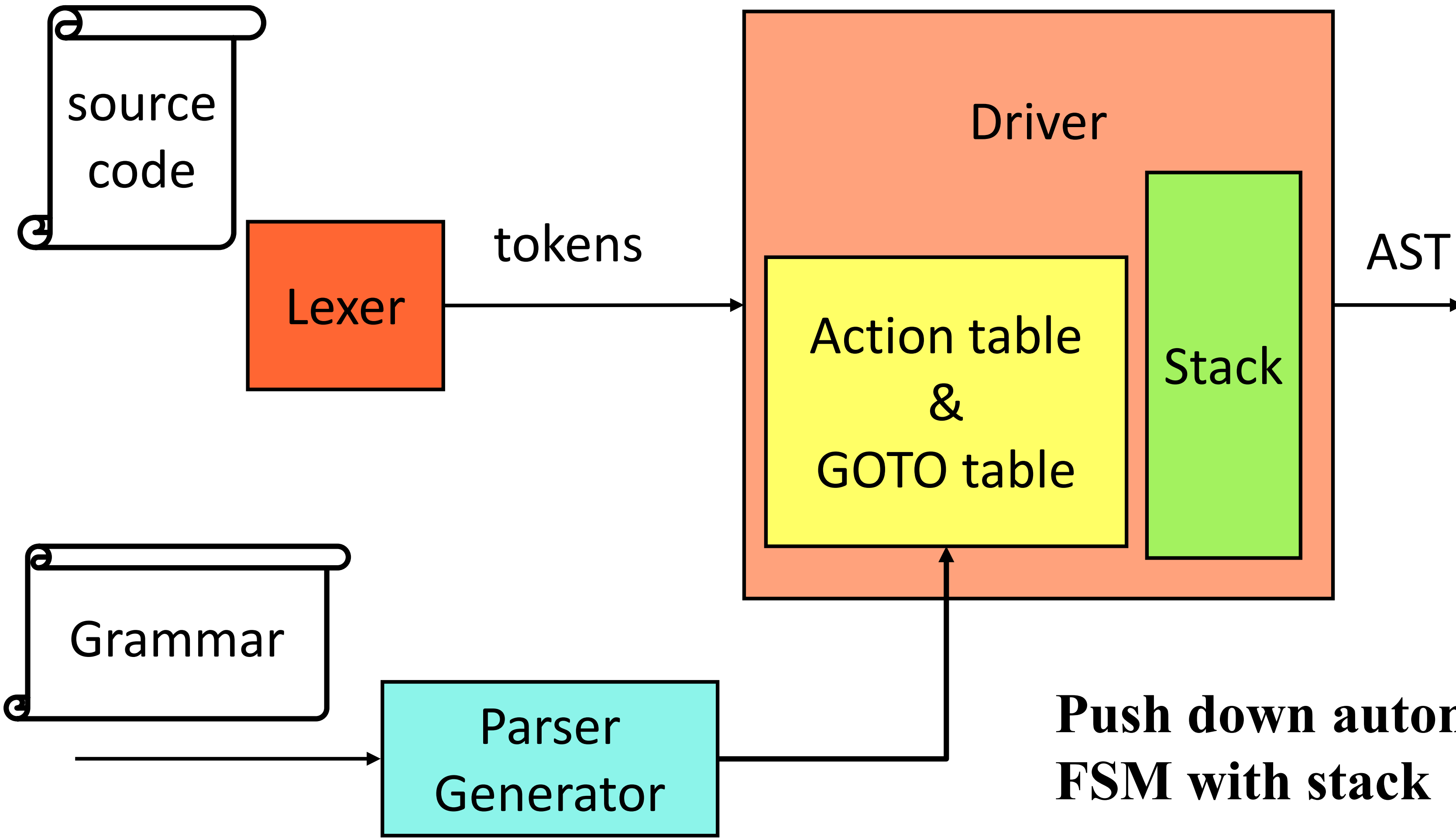
State on top of the stack = **1**

(1,E)

(0,S)

x + y$

Table-driven LR(k) parsers



Push down automata: FSM with stack

The parser generator

Parser Generator

- Finds handles
- Creates the action and GOTO tables.
- Creates the states
  - Each state indicates how much of a handle we have seen
  - each state is a set of *items*

Items

- Items are used to identify handles.
- LR(k) items have the form:

  [production-with-dot, lookahead]

- For example, $A \to a\ X\ b$ has 4 LR(0) items
  - $[A \to \bullet\ a\ X\ b]$
  - $[A \to a \bullet X\ b]$
  - $[A \to a\ X \bullet b]$
  - $[A \to a\ X\ b \bullet]$

The $\bullet$ indicates how much of the handle we have recognized.

What LR(0) Items Mean

- $[X \to \bullet\ \alpha\ \beta\ \gamma]$
  input is consistent with $X \to \alpha\ \beta\ \gamma$
- $[X \to \alpha\ \bullet\ \beta\ \gamma]$
  input is consistent with $X \to \alpha\ \beta\ \gamma$ and we have already recognized α
- $[X \to \alpha\ \beta\ \bullet\ \gamma]$
  input is consistent with $X \to \alpha\ \beta\ \gamma$ and we have already recognized $\alpha\ \beta$
- $[X \to \alpha\ \beta\ \gamma\ \bullet]$
  input is consistent with $X \to \alpha\ \beta\ \gamma$ and we can reduce to X

Generating the States

- Start with start production.
- In this case, "S $\rightarrow$ E\$"

$$\boxed{S \rightarrow \bullet E\$}$$

0 S $\rightarrow$ E\$
1 E $\rightarrow$ T + E
2 E $\rightarrow$ T
3 T $\rightarrow$ *identifier*

- Each state is consistent with what we have already shifted from the input and what is possible to reduce. So, what other items should be in this state?

Completing a state

- For each item in a state, add in all other consistent items.

$$\boxed{\begin{array}{l} S \to \bullet E\$ \\ E \to \bullet T + E \\ E \to \bullet T \\ T \to \bullet \textit{identifier} \end{array}}$$

0 $S \to E\$$
1 $E \to T + E$
2 $E \to T$
3 $T \to \textit{identifier}$

- This is called, taking the closure of the state.

Closure*

```
closure(state)
   repeat
      foreach item A → a•Xb in state
         foreach production X → w
            state.add(X → •w)
   until state does not change
   return state
```

Intuitively:

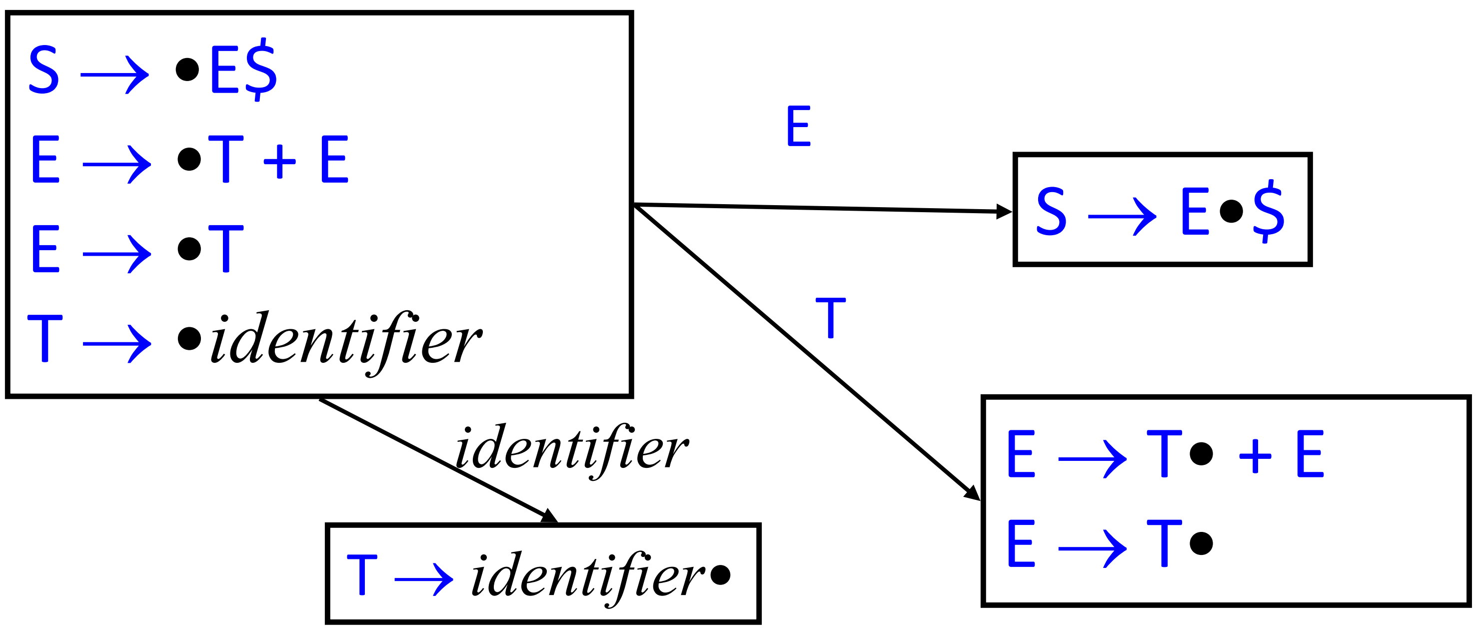
Given a set of items, add all production rules that could produce the nonterminal(s) at the current position in each item

*: for LR(0) items

What about the other states?

- How do we decide what the other states are?
- How do we decide what the transitions between states are?

0 $S \rightarrow E\$$
1 $E \rightarrow T + E$
2 $E \rightarrow T$
3 $T \rightarrow \mathit{identifier}$



Next(state, sym)

- Next function determines what state to goto based on current state and symbol being recognized.
- For Non-terminal, this is used to determine the GOTO table.
- For terminal, this is used to determine the shift action.

Constructing states

```
initial_state = closure({start production})
state_set.add(initial_state)
state_queue.push(initial_state)

while(!state_queue.empty())
   s = state_queue.pop()
   foreach item A → a•Xb in s
      n = closure(next(s, X))
      if(!state_set.contains(n))
         state_set.add(n)
         state_queue.push(n)
```

A state is a set of LR(0) items

get "next" state

Closure*

closure({$S \rightarrow \bullet E\$$}) =

$S \rightarrow \bullet E\$$

0 $S \rightarrow E\$$
1 $E \rightarrow T + E$
2 $E \rightarrow T$
3 $T \rightarrow \textit{identifier}$

*: for LR(0) items

Closure*

closure({$S \rightarrow \bullet E\$$}) =

$S \rightarrow \bullet E\$$

$E \rightarrow \bullet T + E$

$E \rightarrow \bullet T$

$T \rightarrow \bullet \mathit{identifier}$

0 $S \rightarrow E\$$

1 $E \rightarrow T + E$

2 $E \rightarrow T$

3 $T \rightarrow \mathit{identifier}$

*: for LR(0) items

Next

```
next(state, X)
    ret = empty
    foreach item A → a•Xb in state
        ret.add(A → aX•b)
    return ret
```

0 $S \to E\$$

1 $E \to T + E$

2 $E \to T$

3 $T \to \textit{identifier}$

initial:

$S \to \bullet E\$$

$E \to \bullet T + E$

$E \to \bullet T$

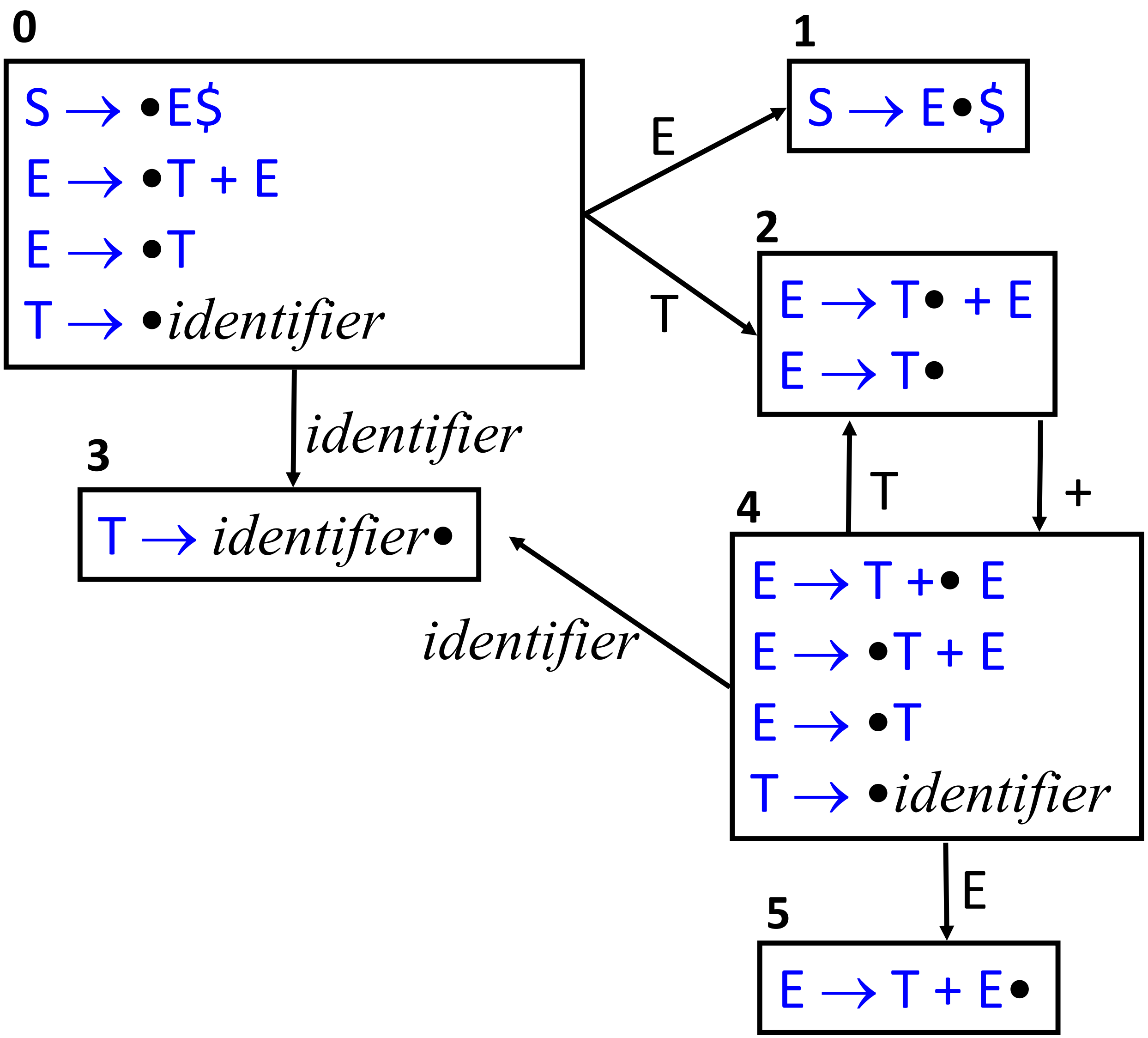
$T \to \bullet \textit{identifier}$

next(initial, E)

next(initial, T)

next(initial, *identifier*)

Example



0 $S \rightarrow E\$$

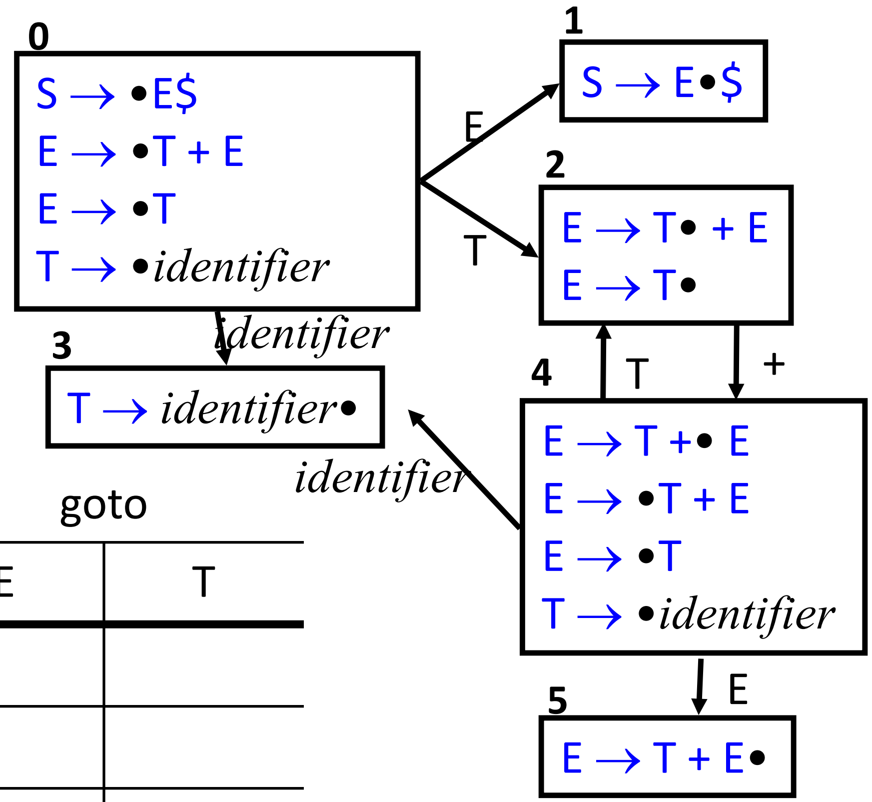
1 $E \rightarrow T + E$

2 $E \rightarrow T$

3 $T \rightarrow \textit{identifier}$

Parse Tables for LR(0) parser

What can we fill out?



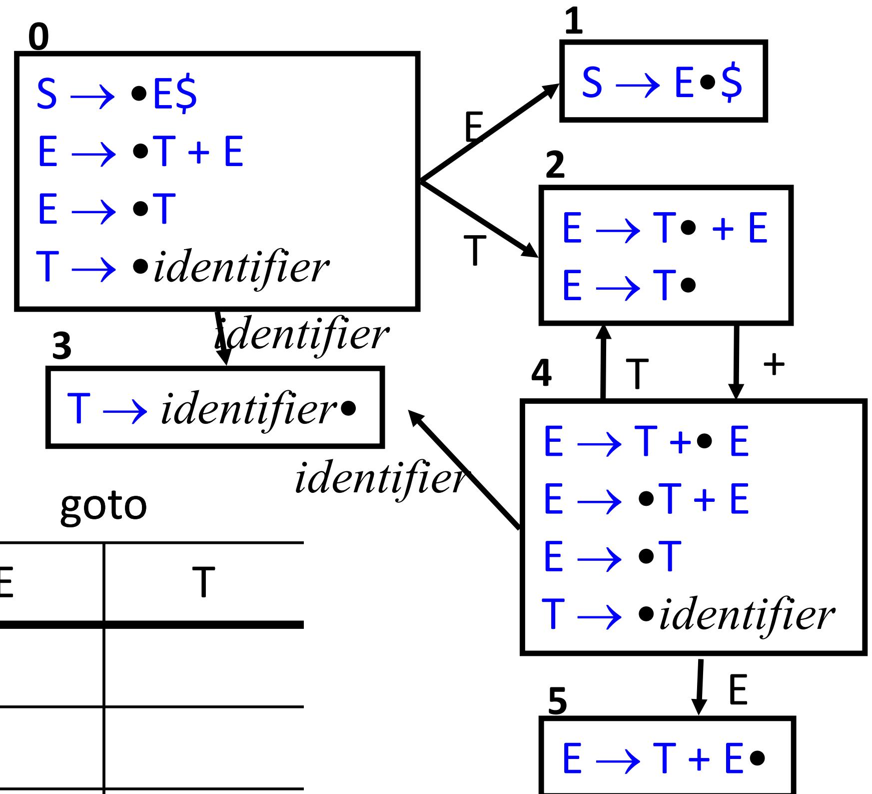
	action			goto	
state	*ident*	+	$	E	T
0					
1					
2					
3					
4					
5					

0 S → E$
1 E → T + E
2 E → T
3 T → *identifier*

Parse Tables for LR(0) parser

shift

transition on terminal



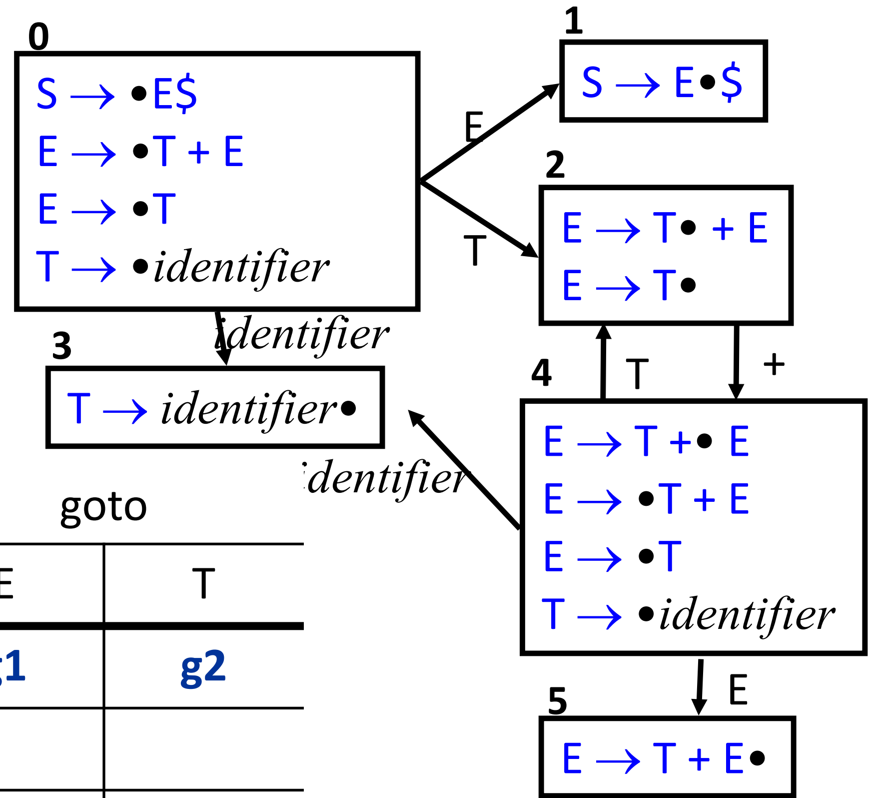
	action			goto	
state	*ident*	+	$	E	T
0	s3				
1					
2		s4			
3					
4	s3				
5					

0 S → E$
1 E → T + E
2 E → T
3 T → *identifier*

Parse Tables for LR(0) parser

goto

transition on nonterminal



state	action			goto	
	ident	+	$	E	T
0	s3			g1	g2
1					
2		s4			
3					
4	s3			g5	g2
5					

0 $S \rightarrow E\$$

1 $E \rightarrow T + E$

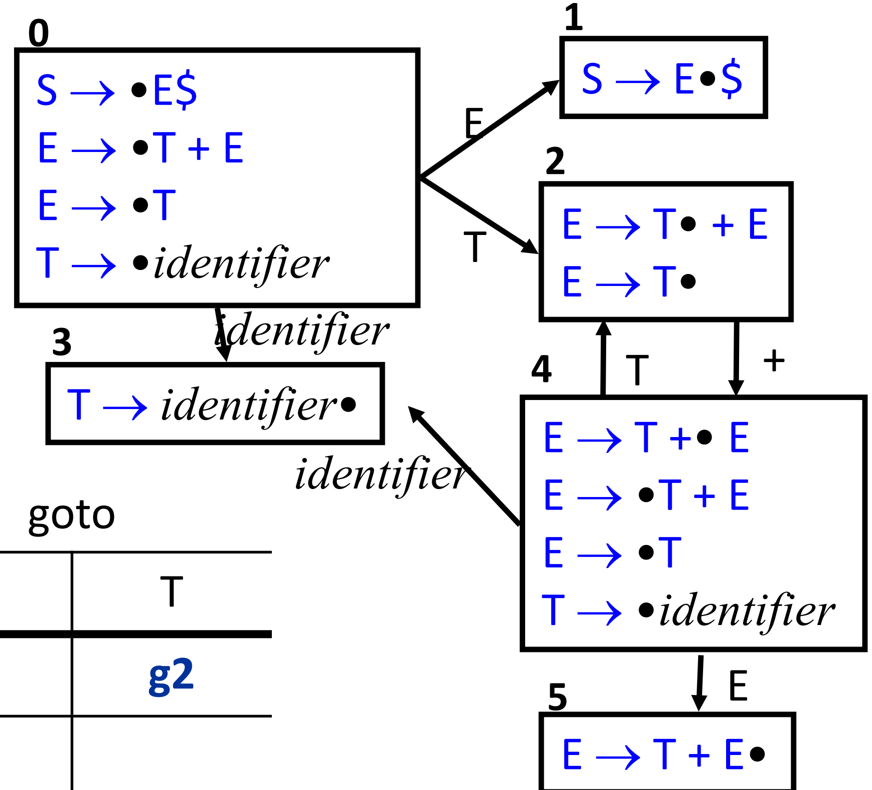
2 $E \rightarrow T$

3 $T \rightarrow \mathit{identifier}$

Parse Tables for LR(0) parser

accept

about to shift $



	action			goto	
state	*ident*	+	$	E	T
0	s3			g1	g2
1			**a**		
2		s4			
3					
4	s3			g5	g2
5					

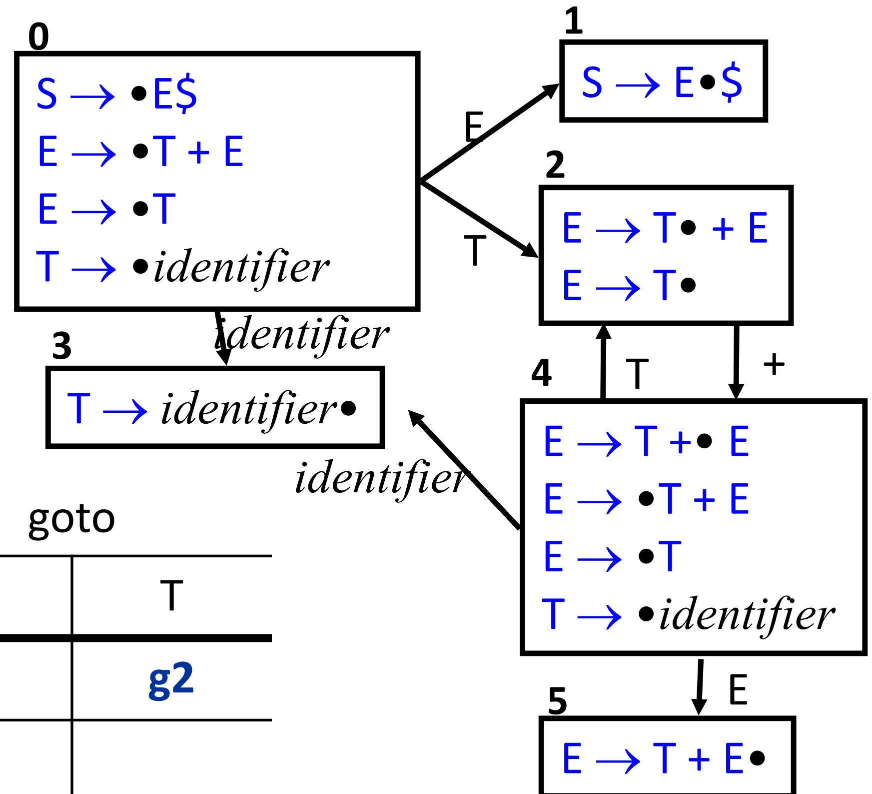
0 $S \rightarrow E\$$
1 $E \rightarrow T + E$
2 $E \rightarrow T$
3 $T \rightarrow \textit{identifier}$

Parse Tables for LR(0) parser

reduce

item has dot at end

$A \rightarrow w\bullet$



	action			goto	
state	*ident*	+	$	E	T
0	s3			g1	g2
1			a		
2		s4			
3					
4	s3			g5	g2
5					

0 $S \rightarrow E\$$

1 $E \rightarrow T + E$

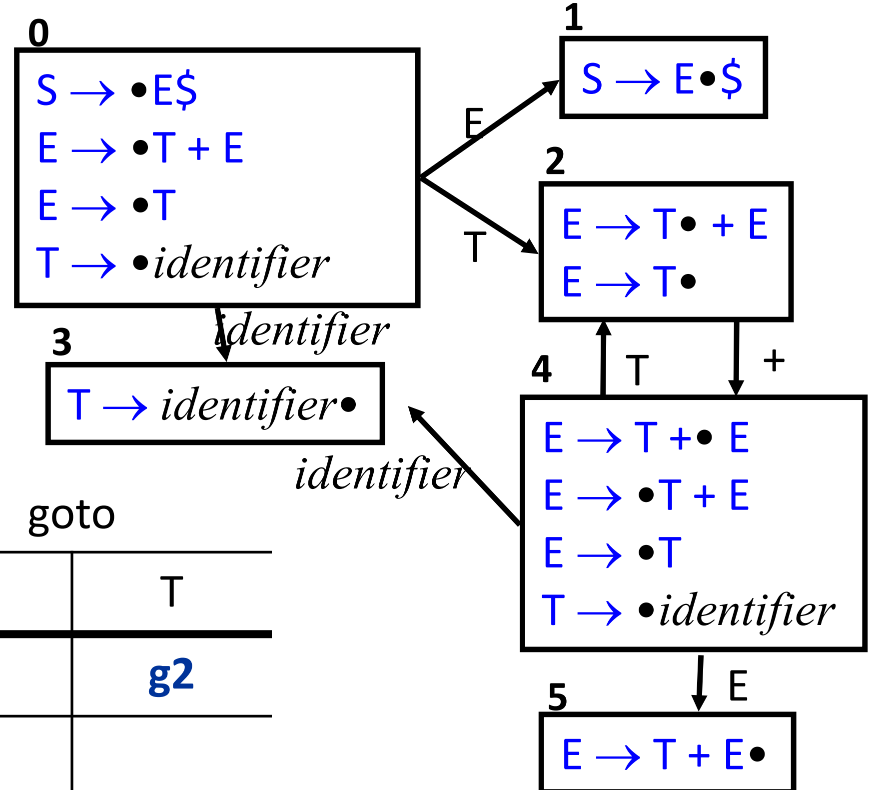
2 $E \rightarrow T$

3 $T \rightarrow$ *identifier*

LR(0)

No lookahead

reduce state for *all* nonterminals



	action			goto	
state	*ident*	+	$	E	T
0	s3			g1	g2
1			**a**		
2	r2	r2/s4	r2		
3	r3	r3	r3		
4	s3			g5	g2
5	r1	r1	r1		

0 S → E$

1 E → T + E

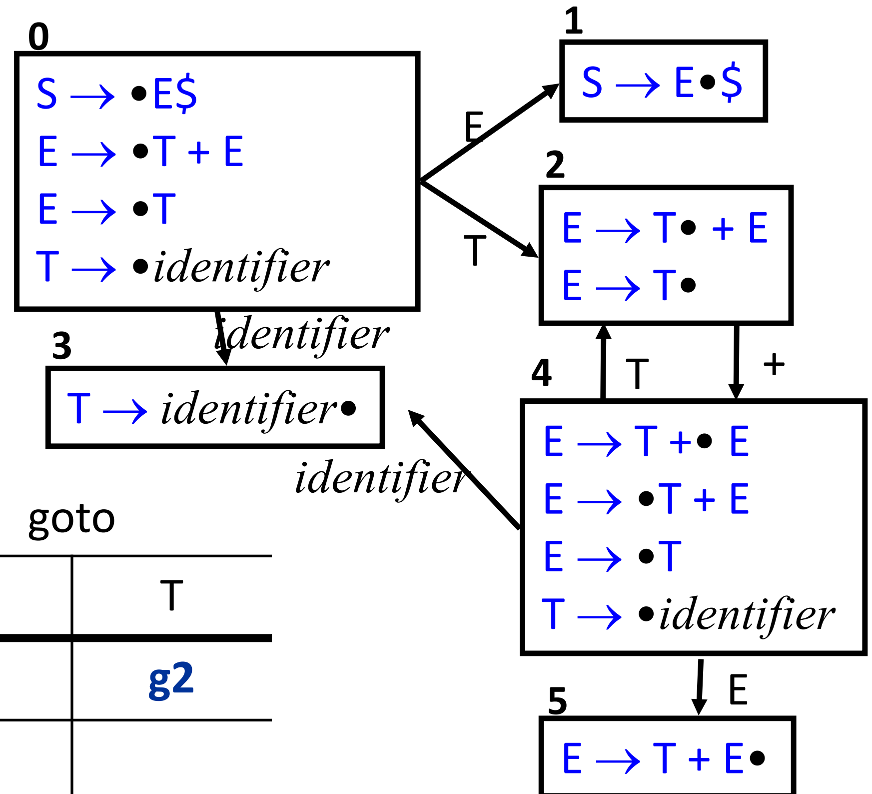
2 E → T

3 T → *identifier*

LR(0)

shift/reduce conflict

need to be pickier about when we reduce

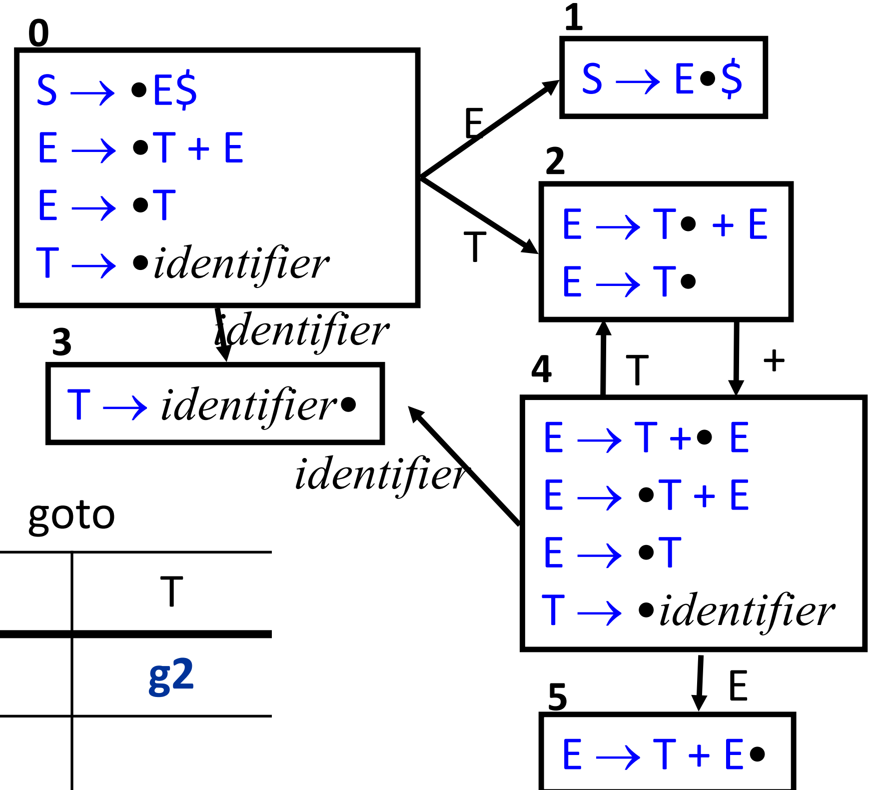


	action			goto	
state	*ident*	+	$	E	T
0	s3			g1	g2
1			a		
2	r2	r2/s4	r2		
3	r3	r3	r3		
4	s3			g5	g2
5	r1	r1	r1		

0 S → E$
1 E → T + E
2 E → T
3 T → *identifier*

SLR - Simple LR

Only reduce in position (**s**,**a**) by rule R:A $\rightarrow$ *w* if **a** is in the *follow set* of A



state	action			goto	
	ident	+	$	E	T
0	s3			g1	g2
1			a		
2		s4			
3					
4	s3			g5	g2
5					

0 $S \rightarrow E\$$

1 $E \rightarrow T + E$

2 $E \rightarrow T$

3 $T \rightarrow \textit{identifier}$

Reminder: Follow sets

follow(X)

set of terminals that can appear immediately after the nonterminal X in some sentential form

0 $S \to E\$$
1 $E \to T + E$
2 $E \to T$
3 $T \to \mathit{identifier}$

I.e., $t \in \text{FOLLOW}(X)$ iff $S \Rightarrow^* \alpha X t \beta$ for some α and β

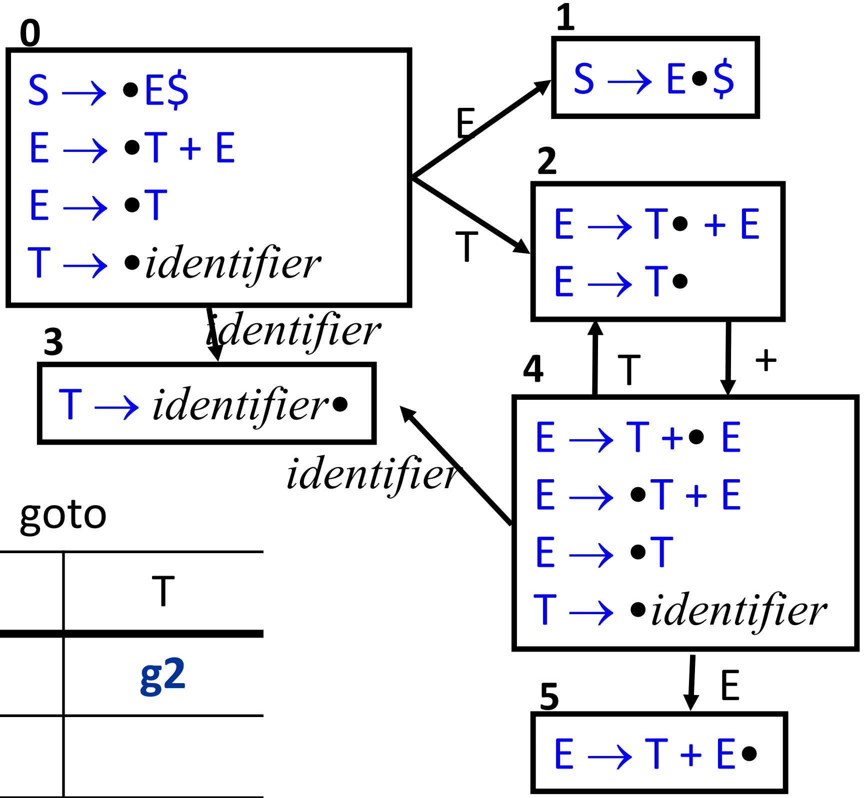
follow(E**) =** {\$}

follow(T**) =** {+,\$}

SLR - Reduce using follow sets

follow(E) = {\$}

follow(T) = {+,\$}



	action			goto	
state	*ident*	+	\$	E	T
0	s3			g1	g2
1			a		
2		s4	r2		
3		r3	r3		
4	s3			g5	g2
5			r1		

0 S → E\$
1 E → T + E
2 E → T
3 T → *identifier*

SLR Limitations

- SLR uses LR(0) item sets
- Can remove some (but not all) shift/reduce conflicts using follow set
- Consider

0 $S \rightarrow E\$$

1 $E \rightarrow L = R$

2 $E \rightarrow R$

3 $L \rightarrow id$

4 $L \rightarrow *R$

5 $R \rightarrow L$

Example

0 $S \to E\$$

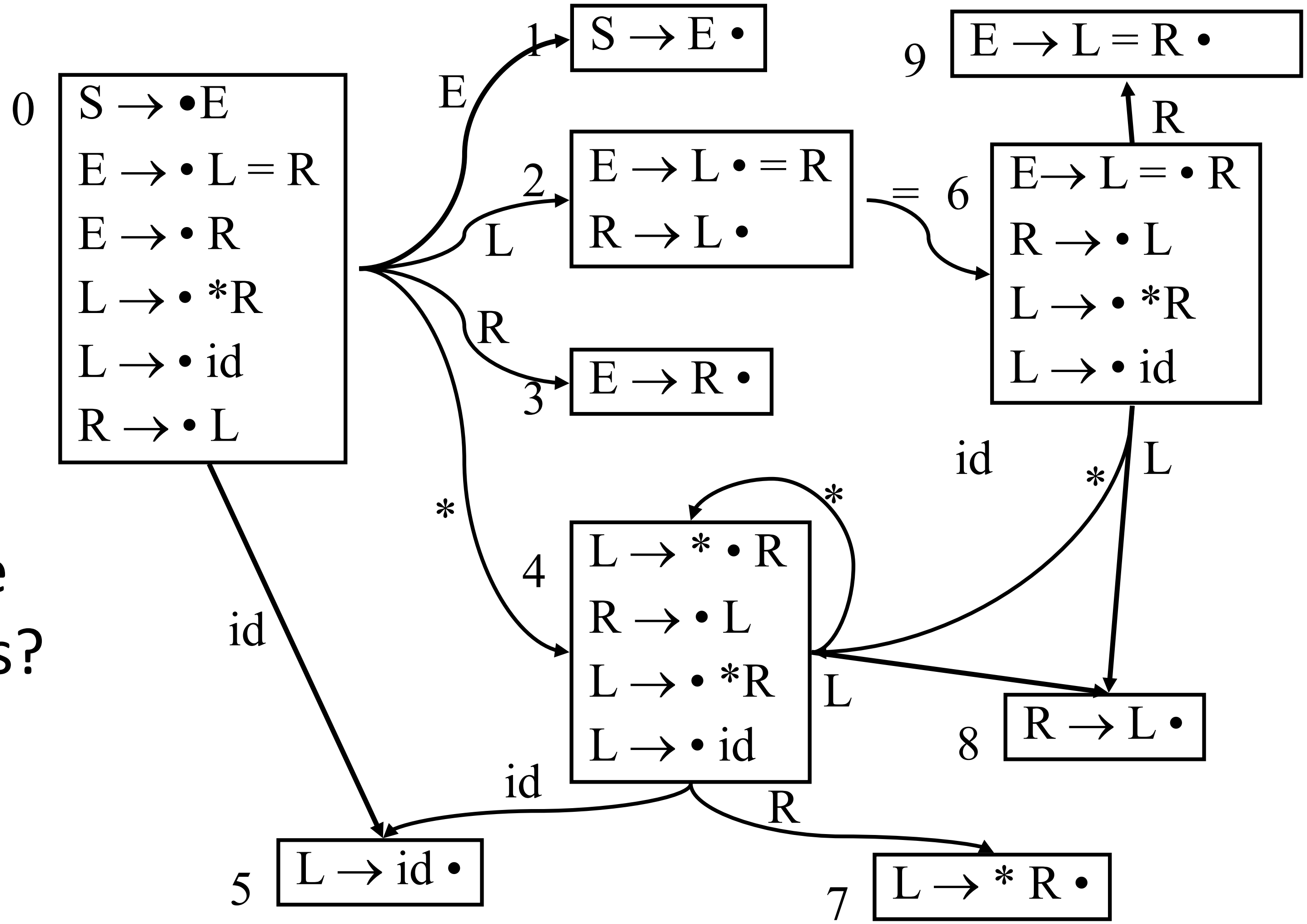
1 $E \to L = R$

2 $E \to R$

3 $L \to id$

4 $L \to *R$

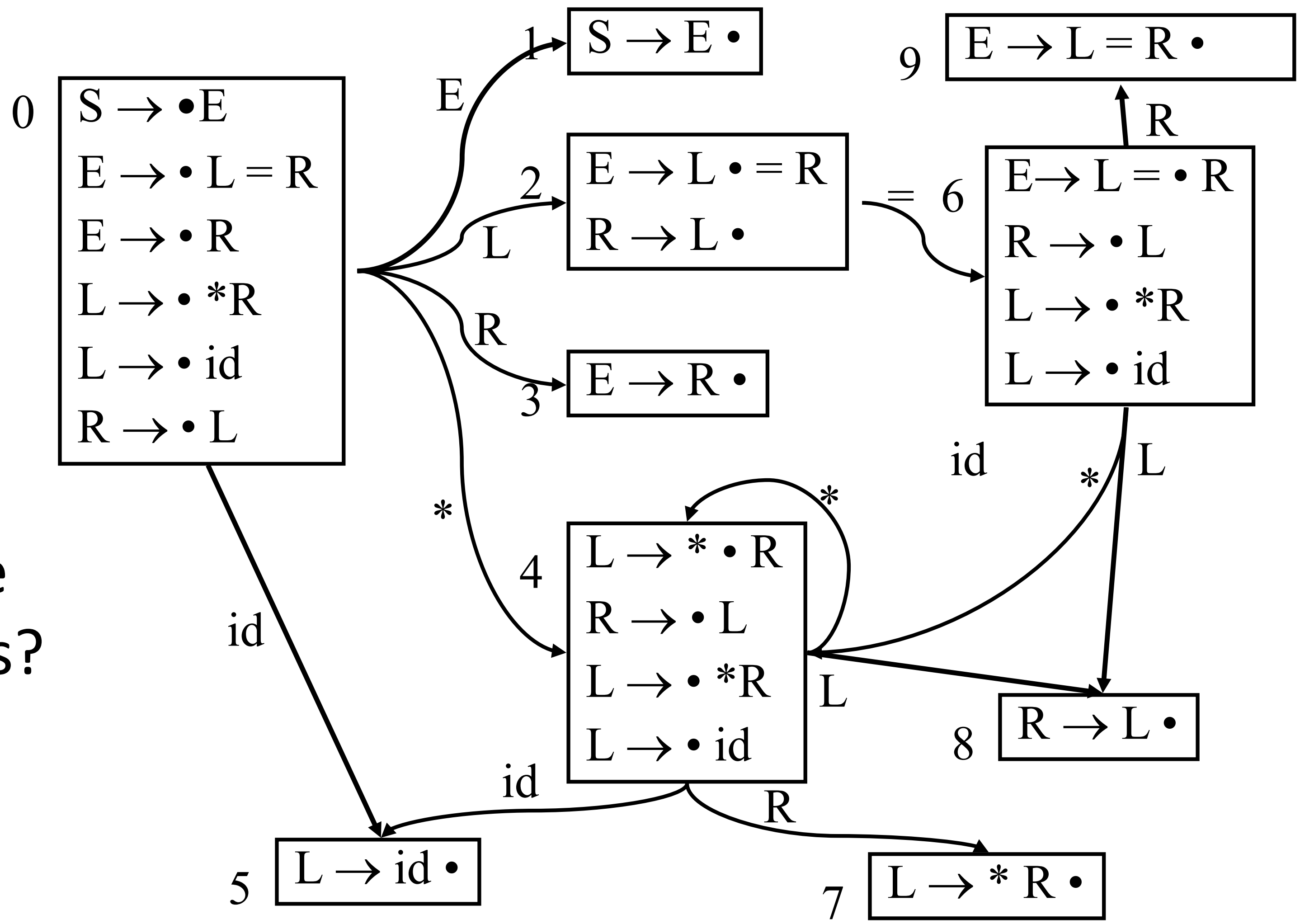
5 $R \to L$



What are the reduce states?

Example

0 $S \rightarrow E\$$
1 $E \rightarrow L = R$
2 $E \rightarrow R$
3 $L \rightarrow id$
4 $L \rightarrow *R$
5 $R \rightarrow L$



What are the reduce states?

1,2,3,5,7,8,9

Example

0 $S \rightarrow E\$$

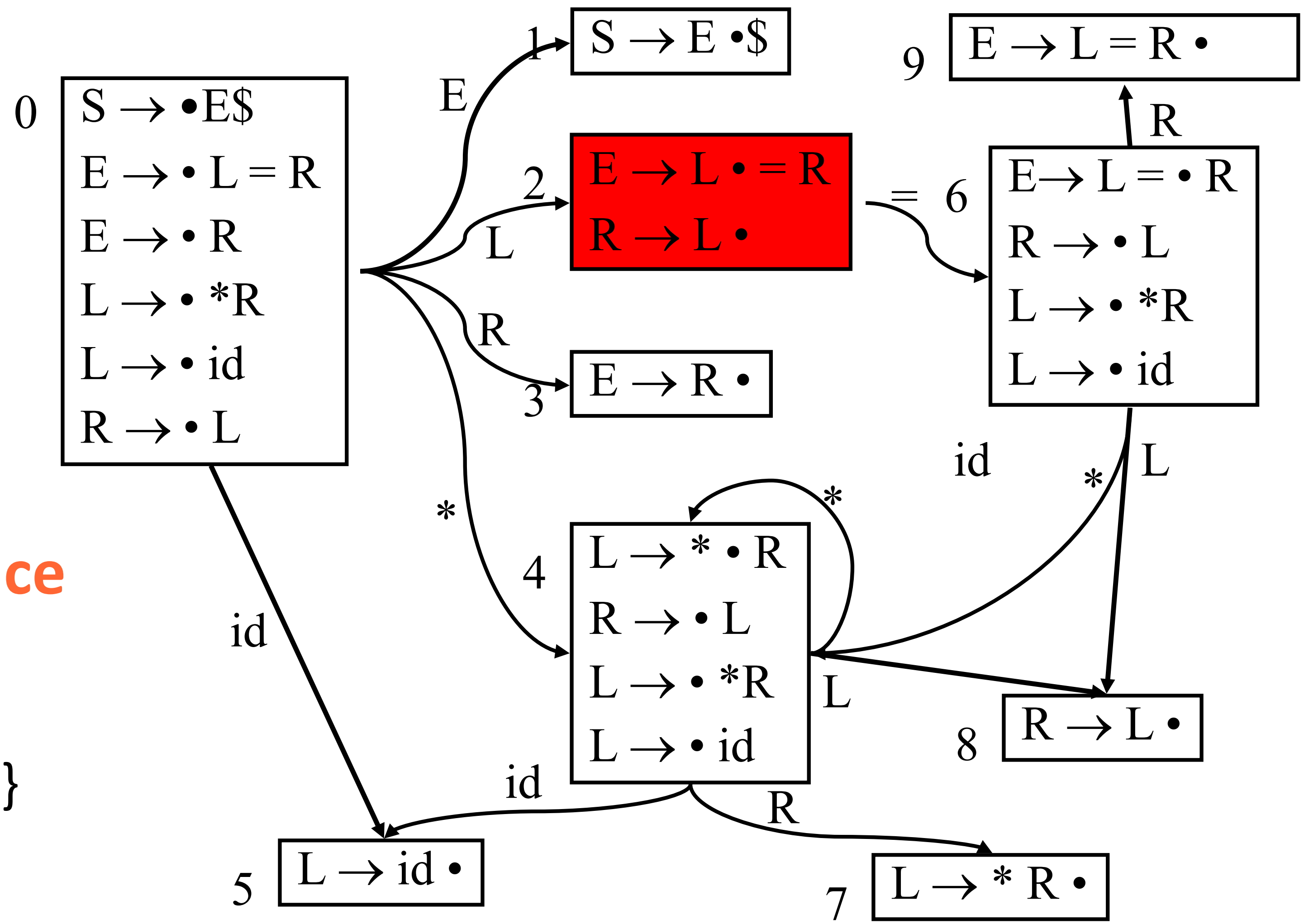
1 $E \rightarrow L = R$

2 $E \rightarrow R$

3 $L \rightarrow id$

4 $L \rightarrow *R$

5 $R \rightarrow L$



shift/reduce conflict

follow(R) = {=,$}

Problem with SLR

- Reduce on ALL terminals in FOLLOW set

S	$\rightarrow$	L = R
	|	R
L	$\rightarrow$	* R
	|	**id**
R	$\rightarrow$	L

2

$S \rightarrow L \bullet = R$
$R \rightarrow L \bullet$

- FOLLOW(R) = FOLLOW(L)
- But, we should never reduce $R \rightarrow L$ on '=' I.e., R=… is not a viable prefix for a right sentential form
- Thus, there should be no reduction in state 2
- How can we solve this?

LR(1) Items

- An LR(1) item is an LR(0) item combined with a single terminal (the *lookahead*)
- $[X \to \alpha \bullet \beta, a]$ Means
  - α is at top of stack
  - Input string is derivable from βa
- In other words, when we reduce $X \to \alpha\beta$, a had better be the look ahead symbol.
- Or, Only put 'reduce by $X \to \alpha\beta$' in **action**[s,a]
- Can construct states as before, but have to modify closure

What LR(1) Items Mean

- $[X \rightarrow \bullet\ \alpha\ \beta\ \gamma$, a]
  input is consistent with $X \rightarrow \alpha\ \beta\ \gamma$
- $[X \rightarrow \alpha\ \bullet\ \beta\ \gamma$, a]
  input is consistent with $X \rightarrow \alpha\ \beta\ \gamma$ and we have already recognized α
- $[X \rightarrow \alpha\ \beta\ \bullet\ \gamma$, a]
  input is consistent with $X \rightarrow \alpha\ \beta\ \gamma$ and we have already recognized $\alpha\ \beta$
- $[X \rightarrow \alpha\ \beta\ \gamma\ \bullet$, a]
  input is consistent with $X \rightarrow \alpha\ \beta\ \gamma$ and if lookahead symbol is a, then we can reduce to X

LR(1) Closure

```
closure(state)
    repeat
        foreach item A → a•Xb, t in state
            foreach production X → w
               and each terminal t′ in FIRST(bt)
                state.add(X → •w, t′)
    until state does not change
    return state
```

Closure

closure({$S \rightarrow \bullet E\$$, ?}) =

$S \rightarrow \bullet E\$$, ?

0 $S \rightarrow E\$$
1 $E \rightarrow L = R$
2 $E \rightarrow R$
3 $L \rightarrow id$
4 $L \rightarrow *R$
5 $R \rightarrow L$

Closure

closure({$S \rightarrow \bullet E\$$, ?}) =

$S \rightarrow \bullet E\$$,	?
$E \rightarrow \bullet L = R$,	\$
$E \rightarrow \bullet R$,	\$

0 $S \rightarrow E\$$
1 $E \rightarrow L = R$
2 $E \rightarrow R$
3 $L \rightarrow id$
4 $L \rightarrow *R$
5 $R \rightarrow L$

Closure

closure({$S \rightarrow \bullet E\$$, ?}) =

$S \rightarrow \bullet E\$$,	?
$E \rightarrow \bullet L = R$,	\$
$E \rightarrow \bullet R$,	\$
$L \rightarrow \bullet id$,	=
$L \rightarrow \bullet *R$,	=

0 $S \rightarrow E\$$
1 $E \rightarrow L = R$
2 $E \rightarrow R$
3 $L \rightarrow id$
4 $L \rightarrow *R$
5 $R \rightarrow L$

Closure

0 $S \to E\$$

1 $E \to L = R$

2 $E \to R$

3 $L \to id$

4 $L \to *R$

5 $R \to L$

closure($\{S \to \bullet E\$, ?\}$) =

$S \to \bullet E\$,$	?
$E \to \bullet L = R,$	\$
$E \to \bullet R,$	\$
$L \to \bullet id,$	=
$L \to \bullet *R,$	=
$R \to \bullet L,$	\$

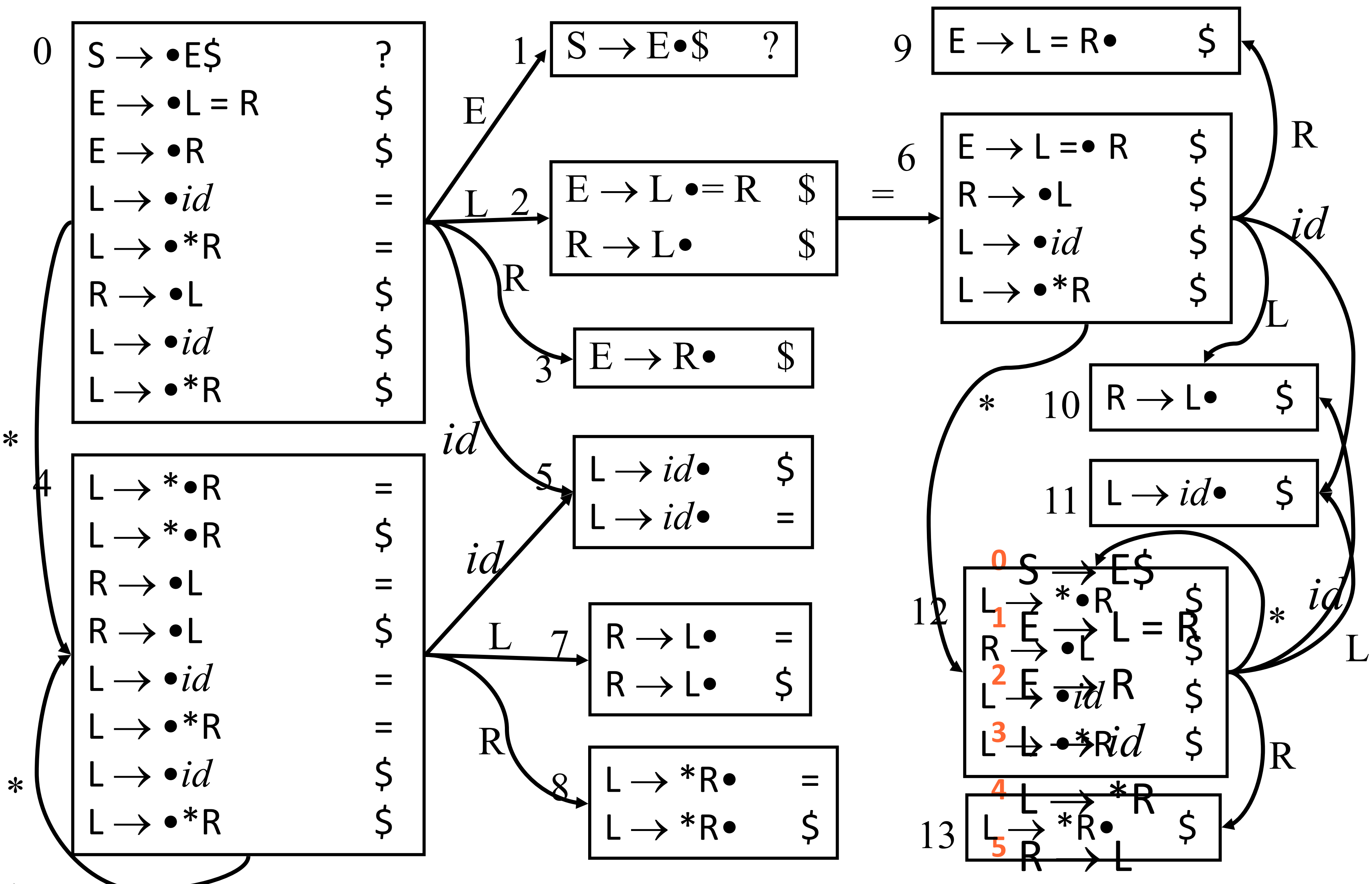
Closure

closure({$S \rightarrow \bullet E\$$, ?}) =

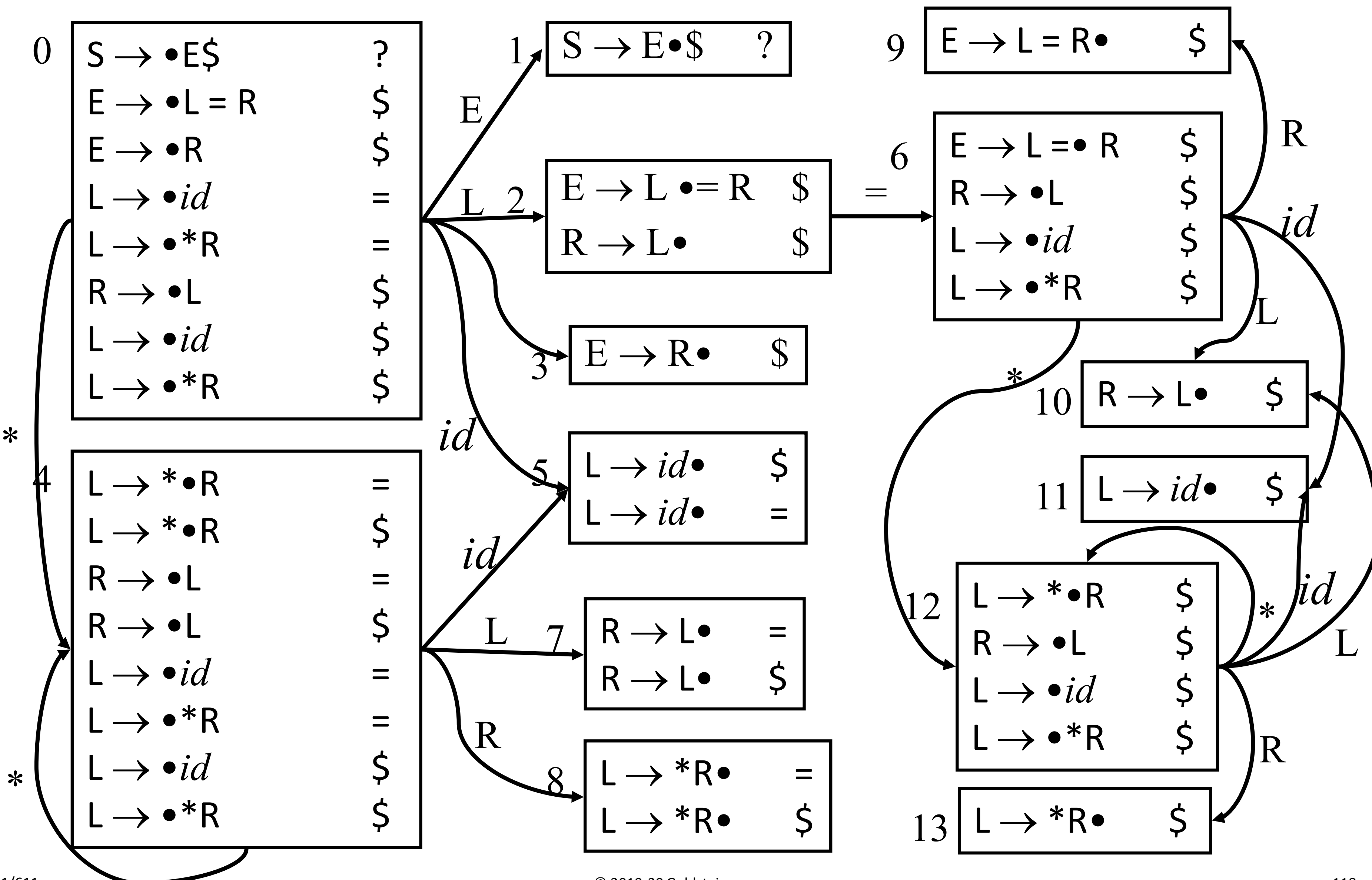
$S \rightarrow \bullet E\$$,	?
$E \rightarrow \bullet L = R$,	\$
$E \rightarrow \bullet R$,	\$
$L \rightarrow \bullet id$,	=
$L \rightarrow \bullet {*}R$,	=
$R \rightarrow \bullet L$,	\$
$L \rightarrow \bullet id$,	\$
$L \rightarrow \bullet {*}R$,	\$

0 $S \rightarrow E\$$
1 $E \rightarrow L = R$
2 $E \rightarrow R$
3 $L \rightarrow id$
4 $L \rightarrow {*}R$
5 $R \rightarrow L$

LR(1) Example


0
S → •E$?
E → •L = R $
E → •R $
L → •id =
L → •*R =
R → •L $
L → •id $
L → •*R $
1
S → E•$?
2
E → L •= R $
R → L• $
3
E → R• $
4
L → *•R =
L → *•R $
R → •L =
R → •L $
L → •id =
L → •*R =
L → •id $
L → •*R $
5
L → id• $
L → id• =
6
E → L =• R $
R → •L $
L → •id $
L → •*R $
7
R → L• =
R → L• $
8
L → *R• =
L → *R• $
9
E → L = R• $
10
R → L• $
11
L → id• $
12
L → *•R $
R → •L $
L → •id $
L → •*R $
13
L → *R• $
0 S → E$
1 E → L = R
2 E → R
3 L → *R id
4 L → *R
5 R → L
E
L
R
id
*
=

LR(1) Example



0
S → •E$?
E → •L = R $
E → •R $
L → •id =
L → •*R =
R → •L $
L → •id $
L → •*R $
1
S → E•$?
E
L
2
E → L •= R $
R → L• $
3
E → R• $
id
5
L → id• $
L → id• =
4
L → *•R =
L → *•R $
R → •L =
R → •L $
L → •id =
L → •*R =
L → •id $
L → •*R $
*
*
id
L
7
R → L• =
R → L• $
R
8
L → *R• =
L → *R• $
=
6
E → L =• R $
R → •L $
L → •id $
L → •*R $
9
E → L = R• $
R
id
L
*
10
R → L• $
11
L → id• $
12
L → *•R $
R → •L $
L → •id $
L → •*R $
*
id
L
R
13
L → *R• $

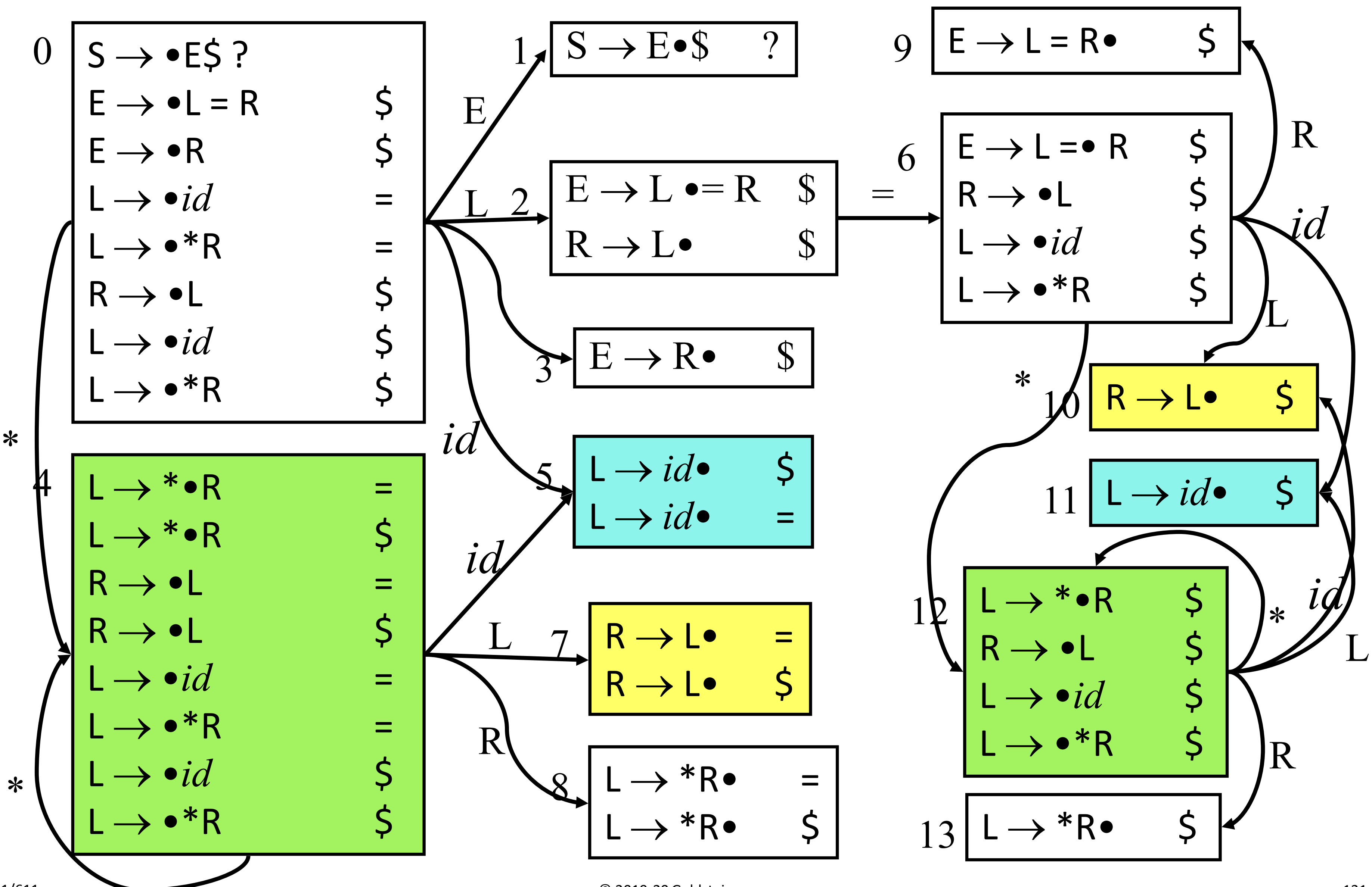
Parsing Table

- 14 states versus 10 LR(0) states
- In general, the number of states (and therefore size of the parsing table) is much larger with LR(1) items

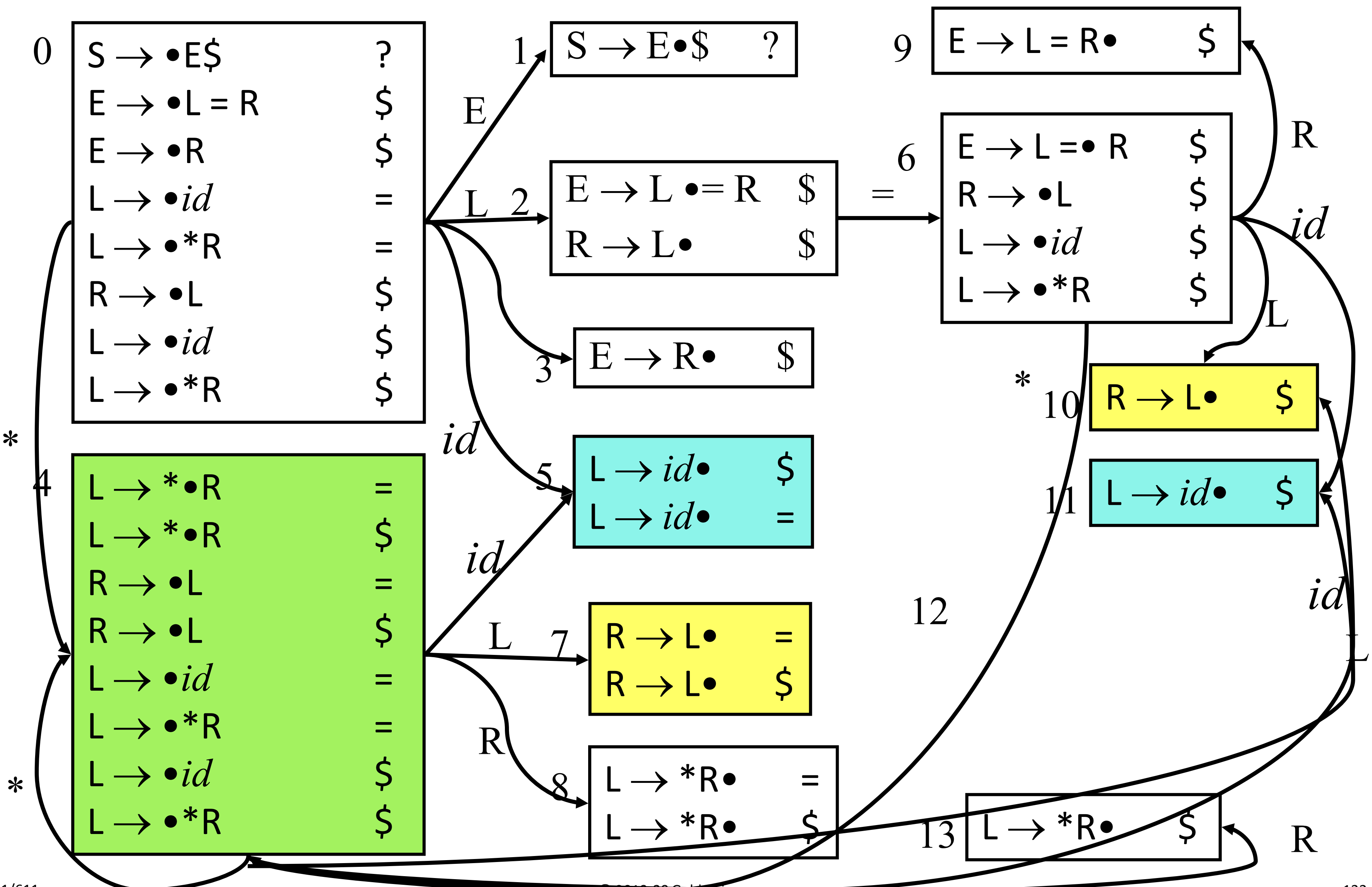
LALR: Lookahead LR

- More powerful than SLR
- Given LR(1) states, merge states that are identical except for lookaheads
- End up with same size table as SLR
- Can this introduce conflicts?

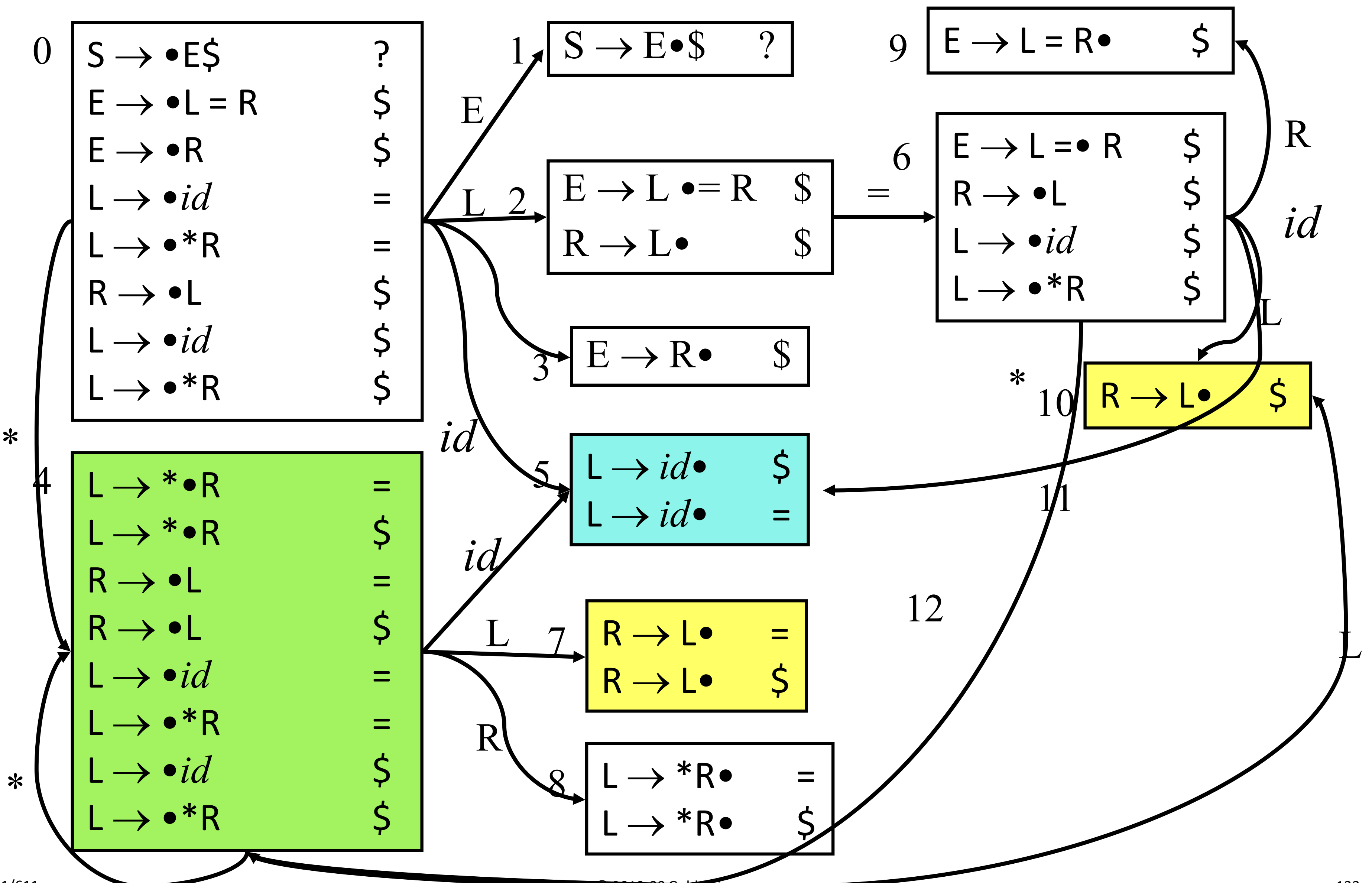
Merge-able states



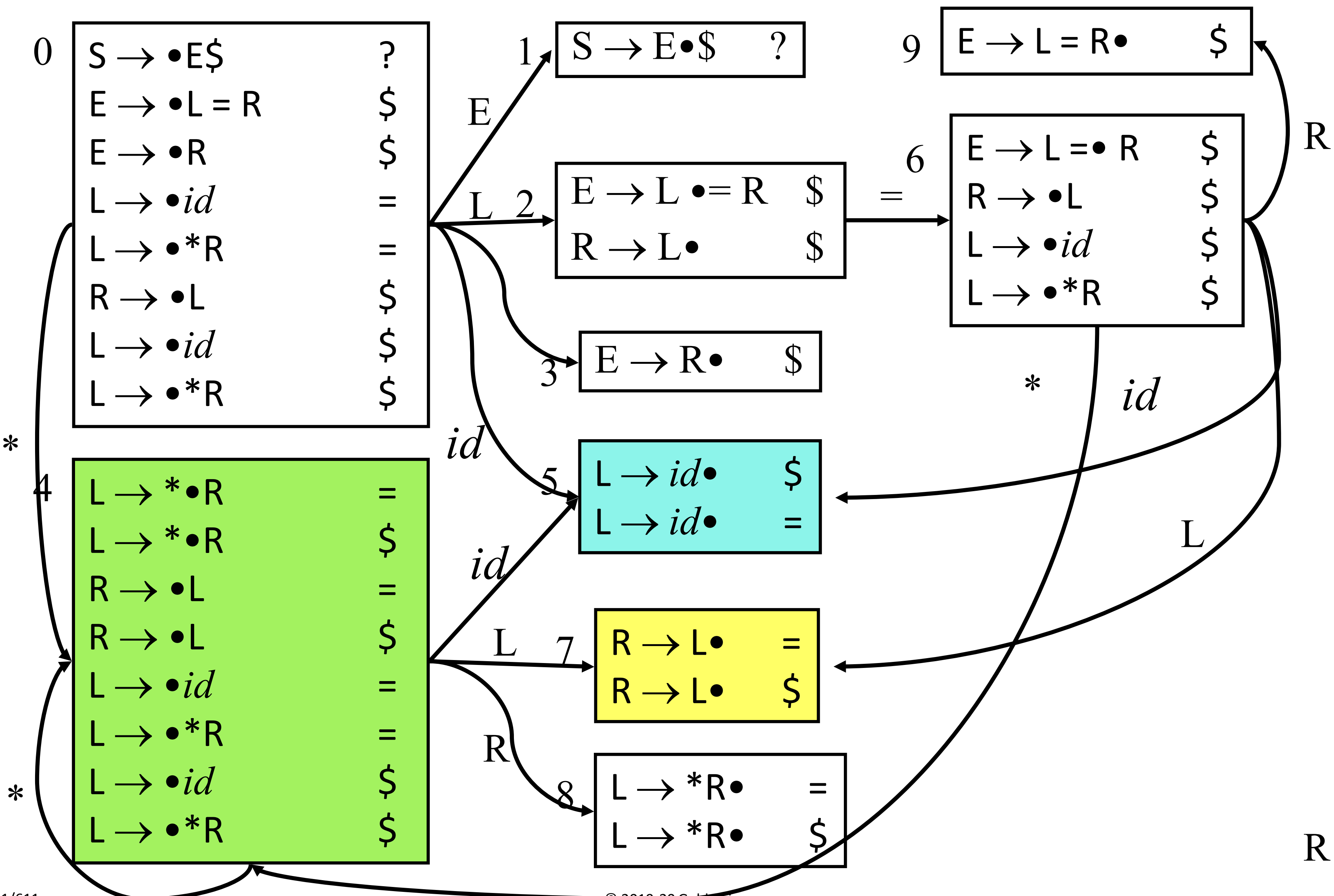
Merge-able states



Merge-able states



Merge-able states



LALR

- Can generate parse table without constructing LR(1) item sets
  - construct LR(0) item sets
  - compute *lookahead* sets
    - more precise than follow sets
- LALR is used by most parser generators (e.g., bison)

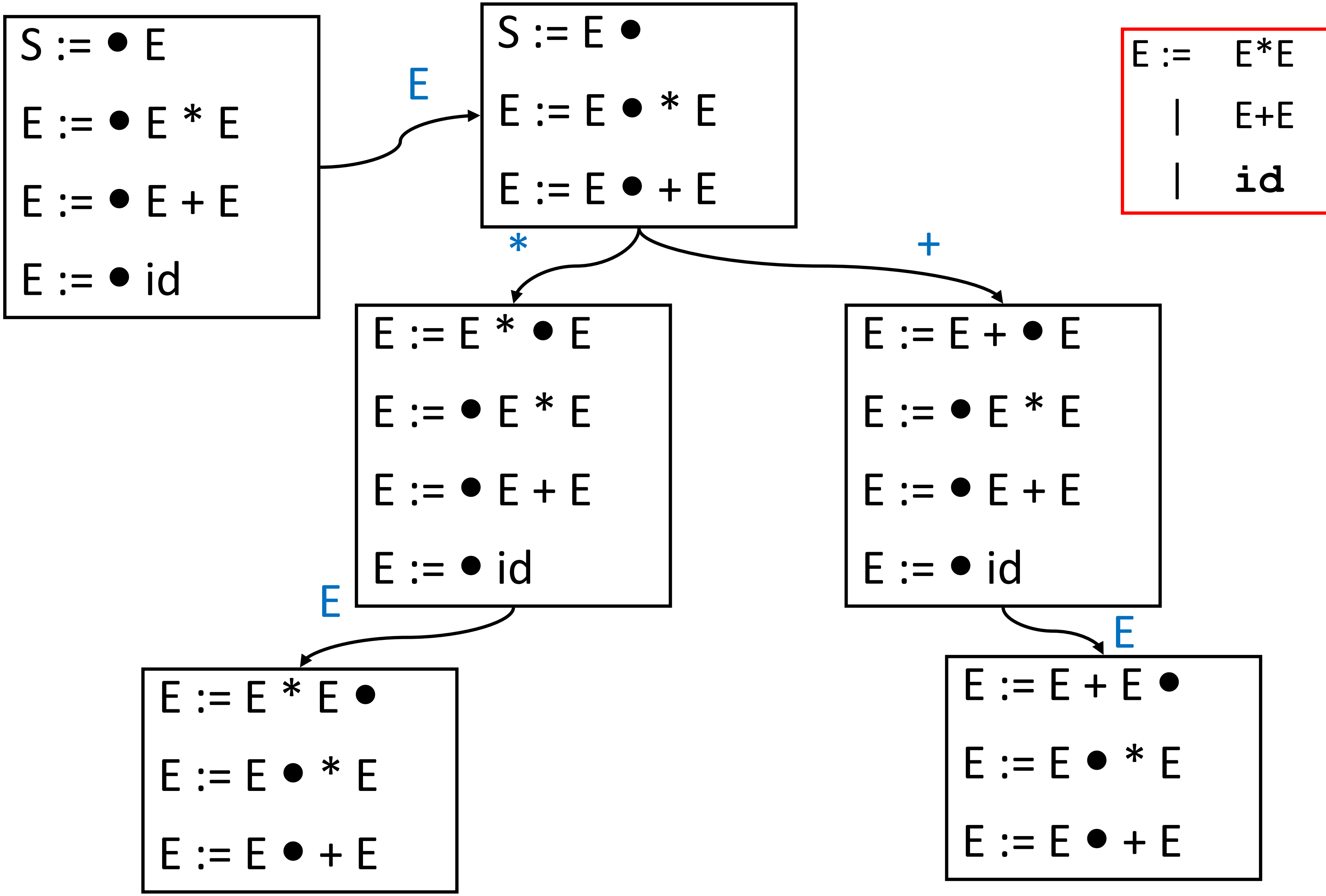
Recap

- LR(0) not very useful
- SLR uses follow sets to reduce
- LALR uses lookahead sets
- LR(1) uses full lookahead context

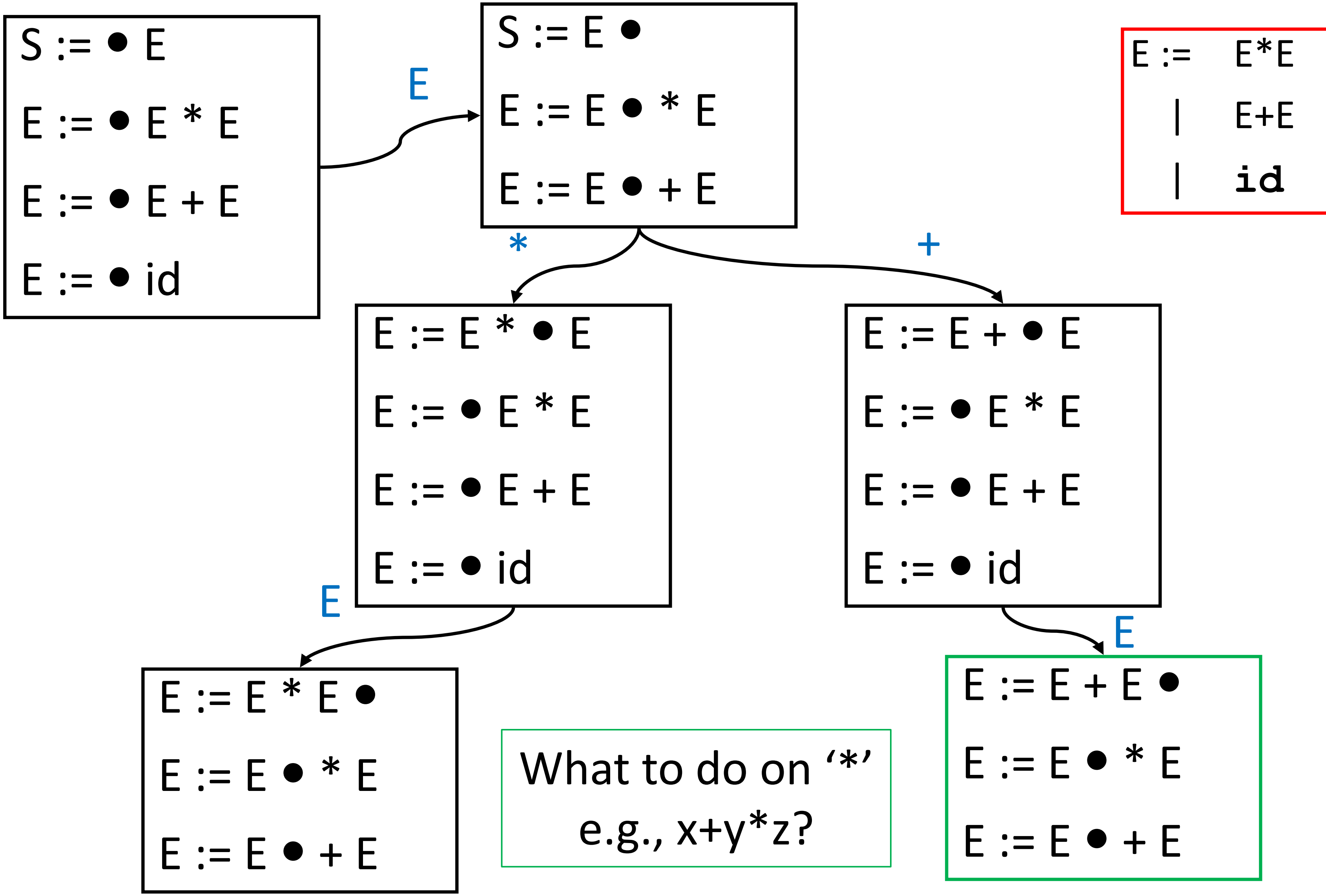
Power of shift-reduce parsers

- There are unambiguous grammars which which cannot be parsed with shift-reduce parsers.
- Such grammars can have
  - shift/reduce conflicts
  - reduce/reduce conflicts
- There grammars are not LR(k)
- But, we can often choose shift or reduce to recognize what want.

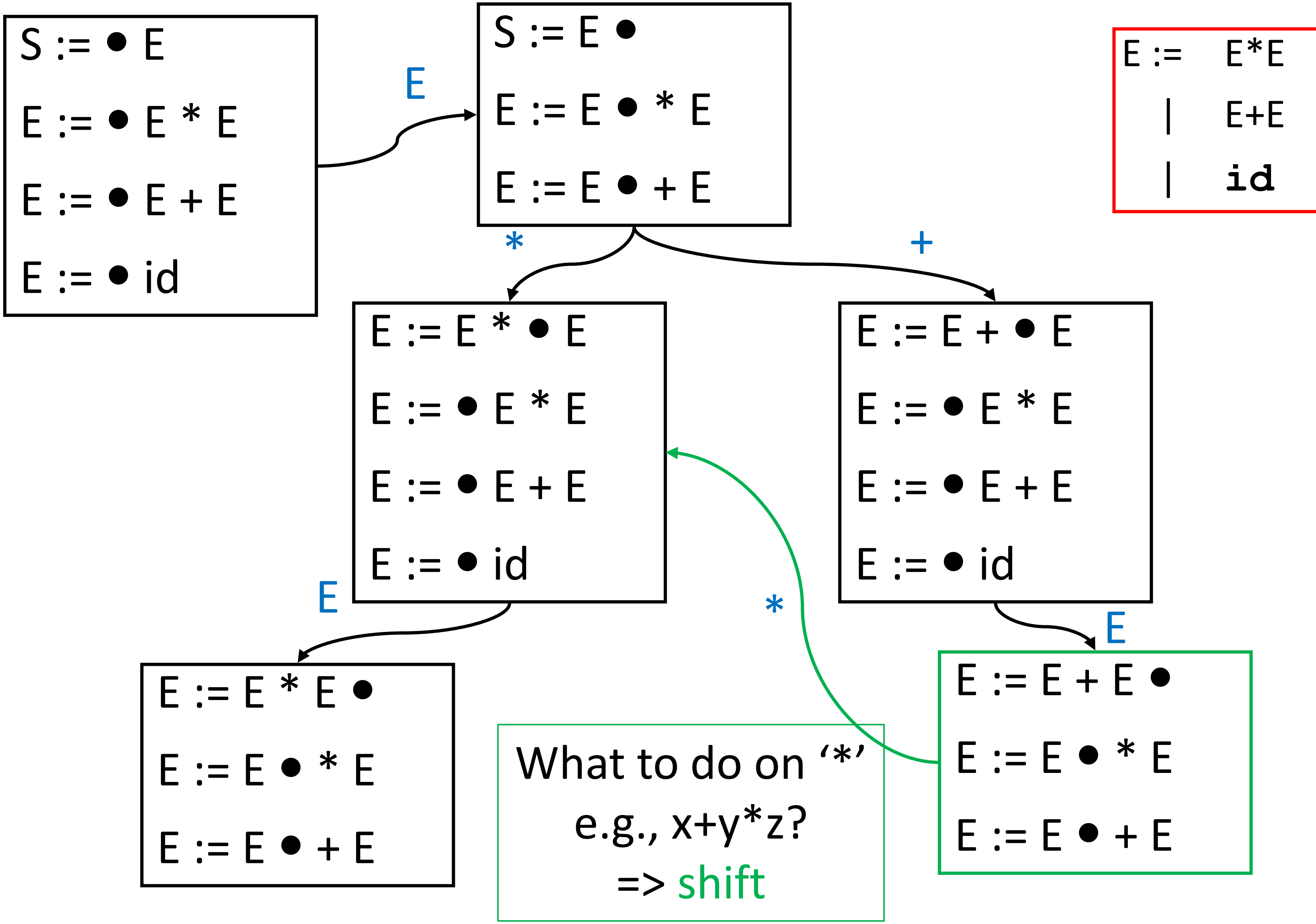
Expression Grammars & Precedence



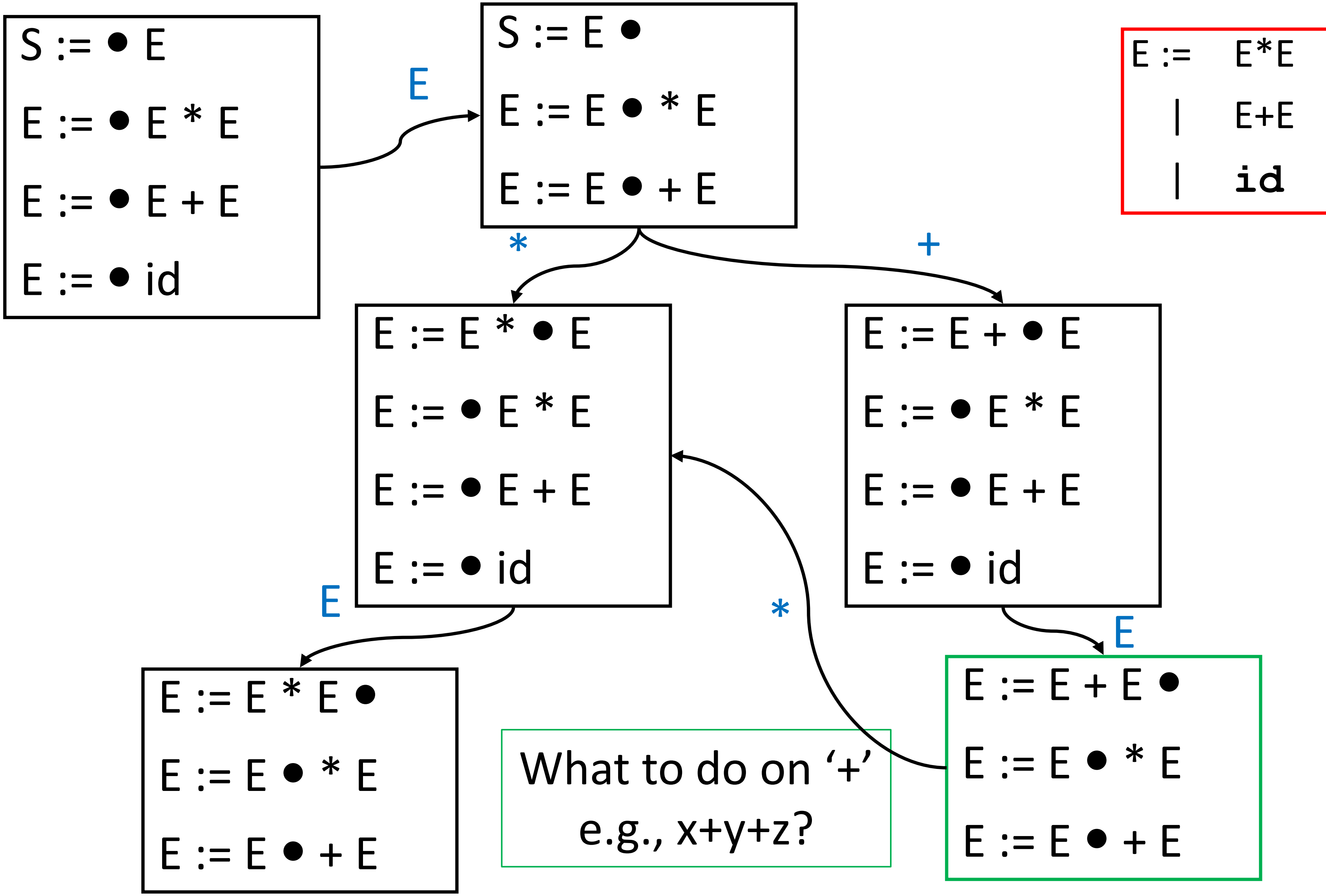
Expression Grammars & Precedence



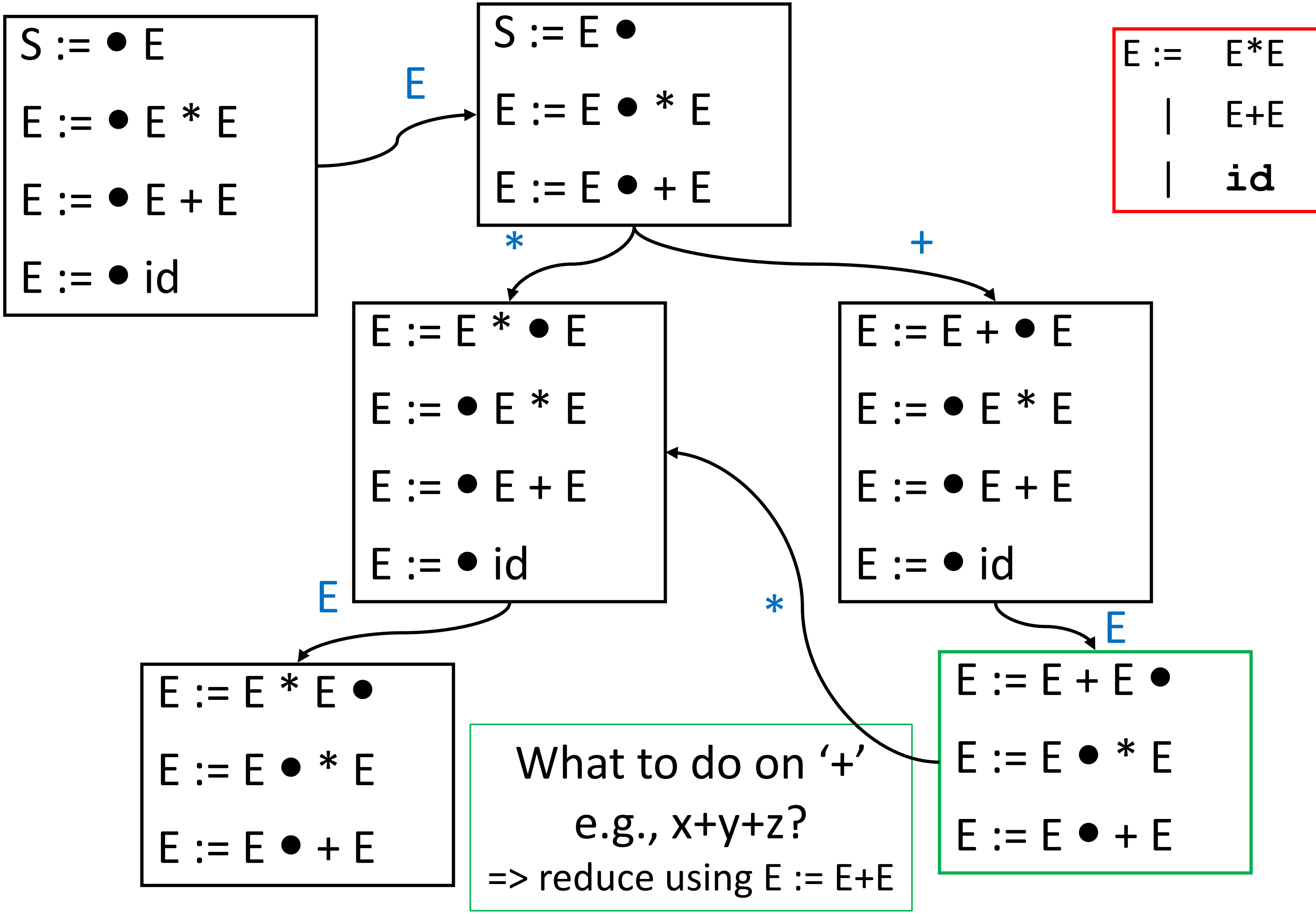
Expression Grammars & Precedence



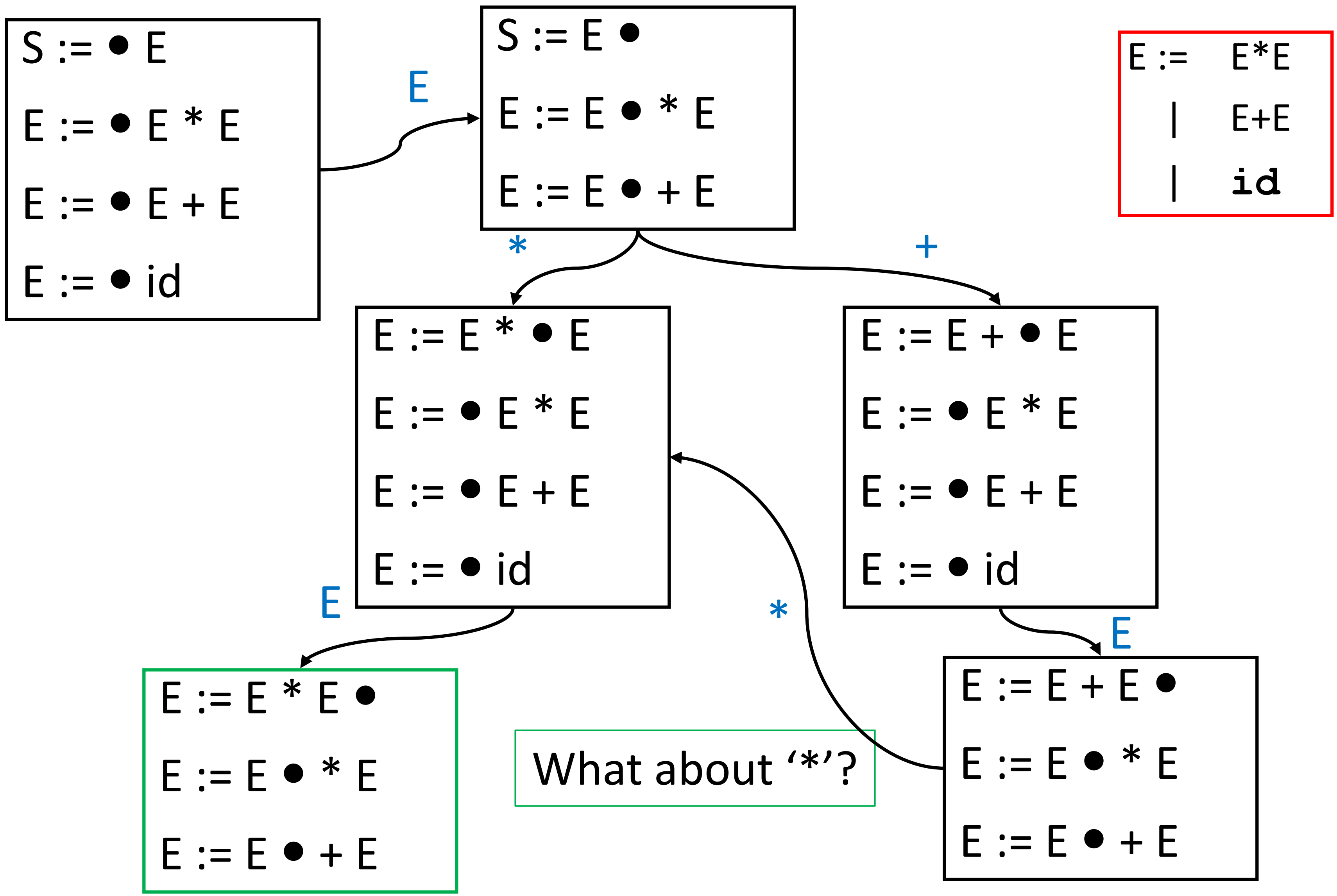
Expression Grammars & Precedence



Expression Grammars & Precedence



Expression Grammars & Precedence



Bison

- Precedence and Associativity declarations
- Precedence derived from order of directives: from lowest to highest
- Associativity from %left, %right, %nonassoc
- Can be attached to rules as well (This can solve the dangling if-else problem
- Use output showing items and transitions to debug s/r and r/r errors.