Compilation

0368-3133

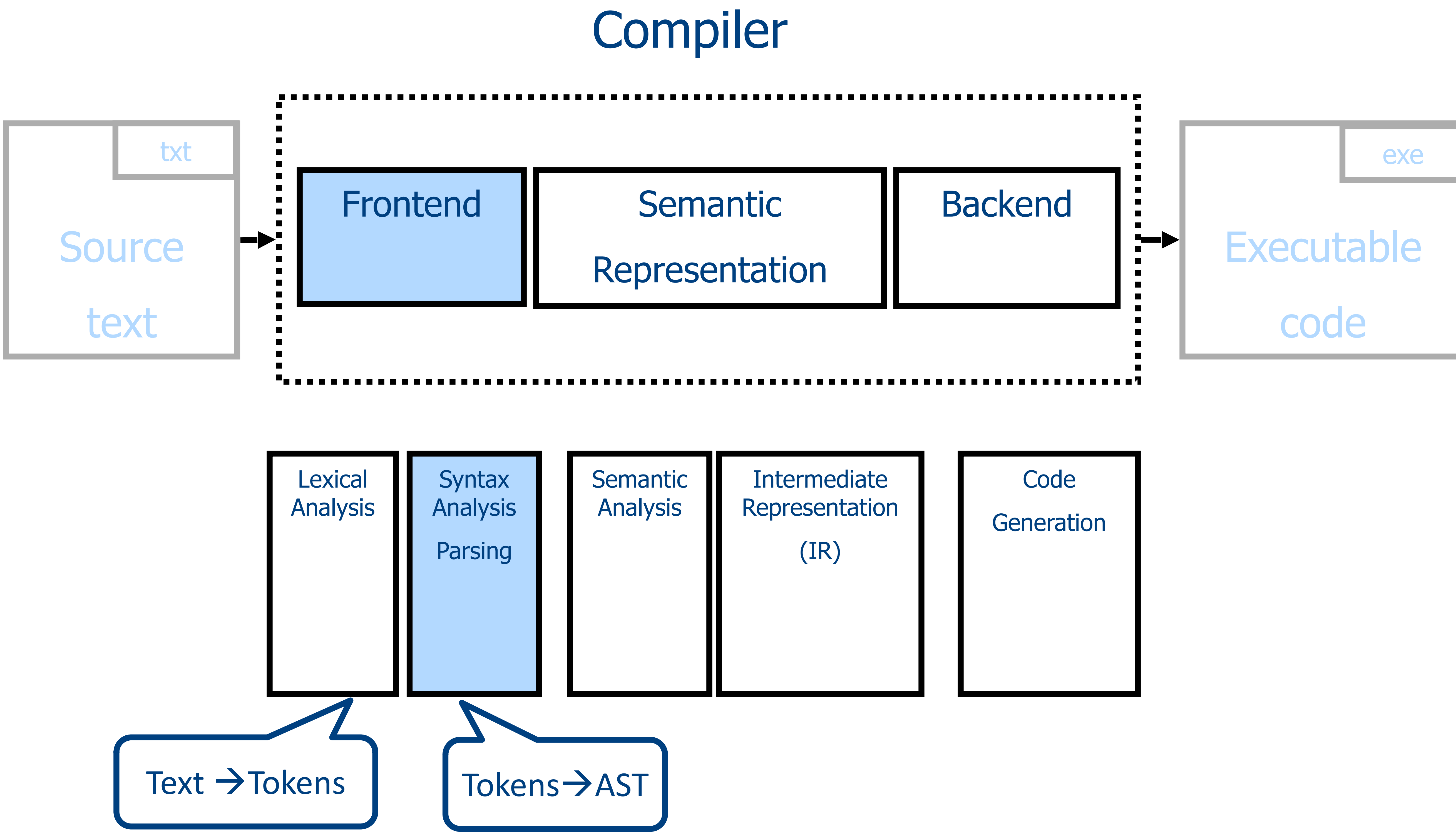
Lecture 3b:

Syntax Analysis:

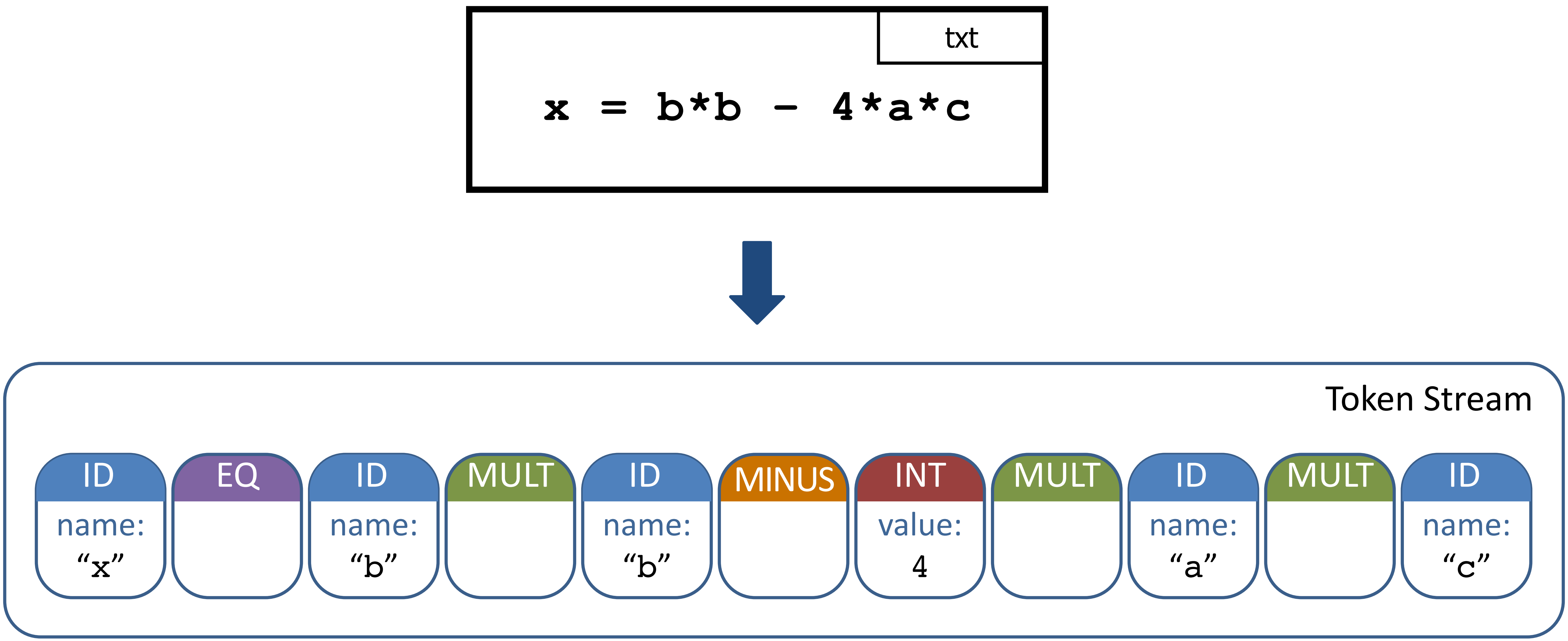
Bottom-Up Parsing

Slides' credit: Eran Yahav, Shachar Itzhaky, Roman Manevich

Conceptual Structure of a Compiler

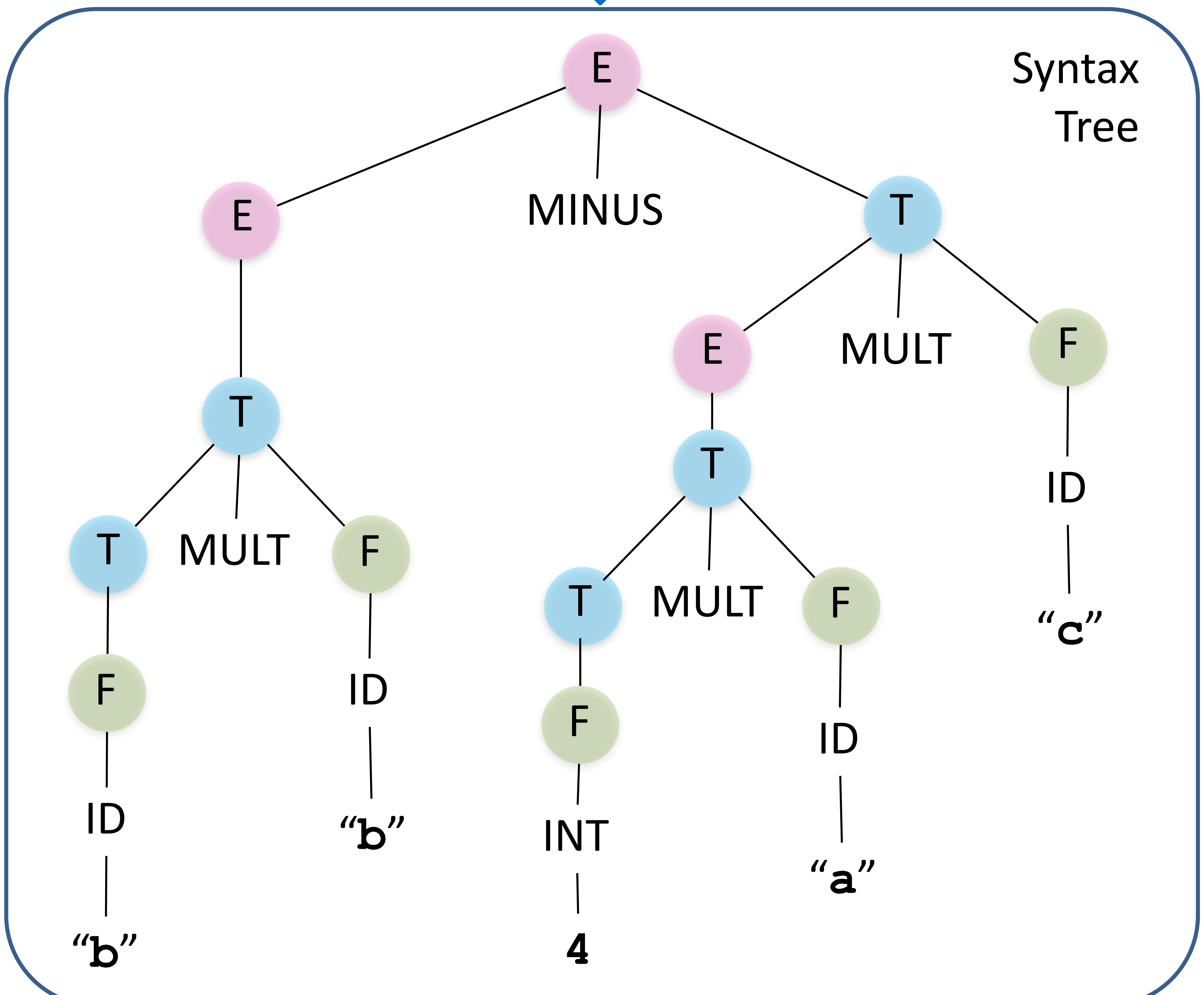


Lexing: from Characters to Tokens



Parsing: from Tokens to Syntax Tree

⟨ID,"b"⟩ ⟨MULT⟩ ⟨ID,"b"⟩ ⟨MINUS⟩ ⟨INT,4⟩ ⟨MULT⟩ ⟨ID,"a"⟩ ⟨MULT⟩ ⟨ID,"c"⟩



Lexical Analysis	Syntax Analysis	Sem. Analysis	Inter. Rep.	Code Gen.

Broad kinds of parsers

- Top-Down parsers (*LL(k) grammars*)
  - Construct parse tree in a top-down matter
  - Find the leftmost derivation
- Bottom-Up parsers (*LR(k) grammars*)
  - Construct parse tree in a bottom-up manner
  - Find the rightmost derivation (in a reverse order)

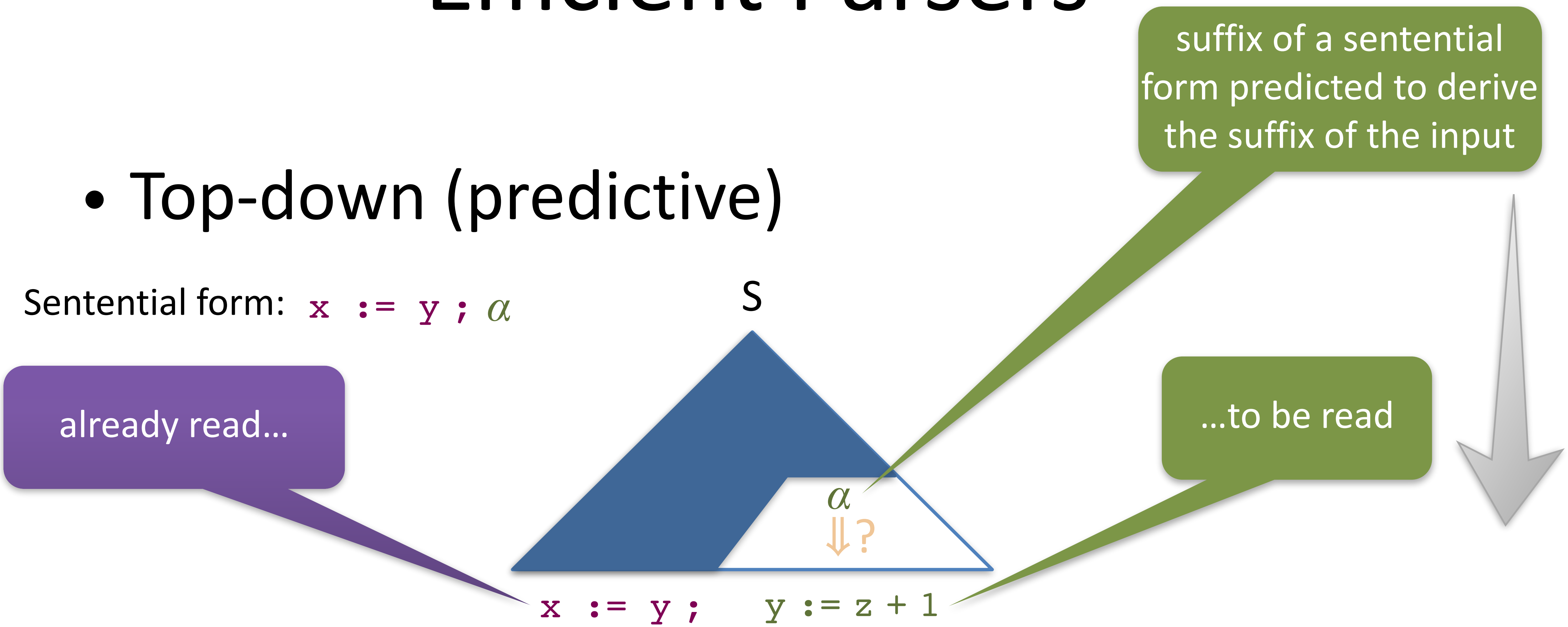
Efficient (O(n=|input|))

- Parsers for arbitrary grammars
  - Earley's method ($O(n^3)$ for ambiguoug grammars. $O(n^2)$ for unambiguoug grammars), CYK method ($O(n^3)$)
  - Usually, not used in practice (though might change)

Efficient Parsers

- Top-down (predictive)

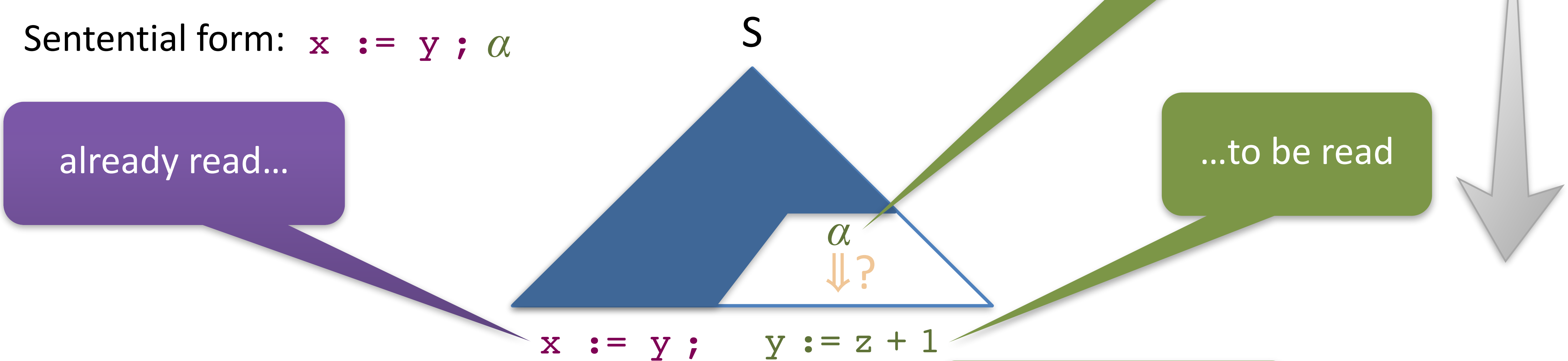
Sentential form: `x := y ;` α



Efficient Parsers

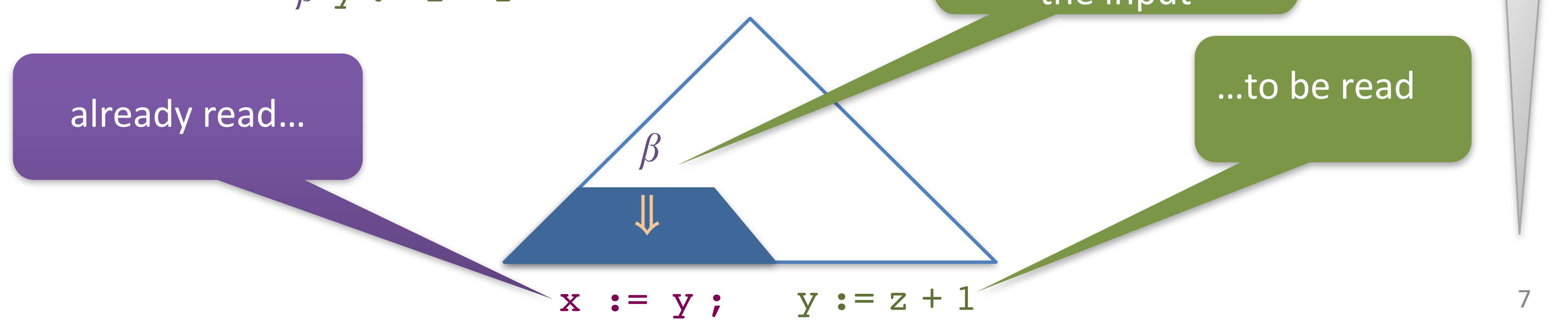
- Top-down (predictive)

Sentential form: `x := y ;` α



- Bottom-up (shift-reduce)

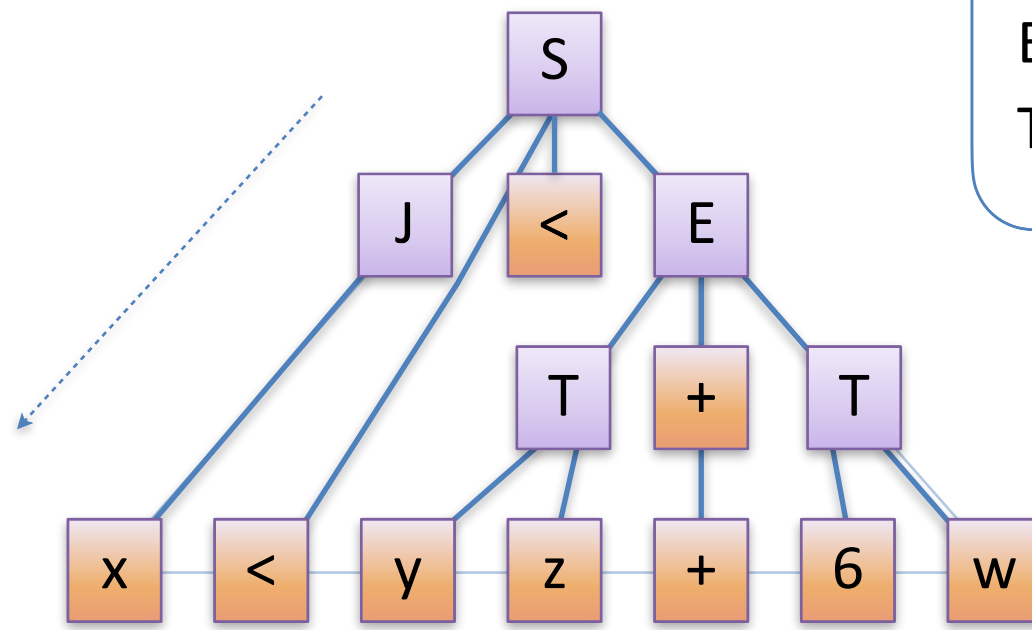
Sentential form? β `y := z + 1`



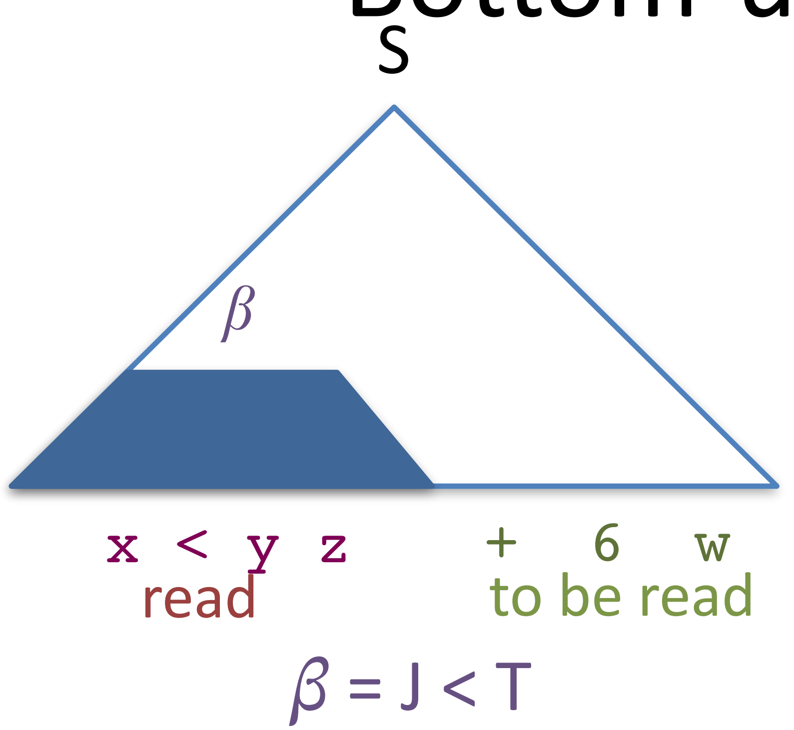
Efficient Parsers – Illustrated

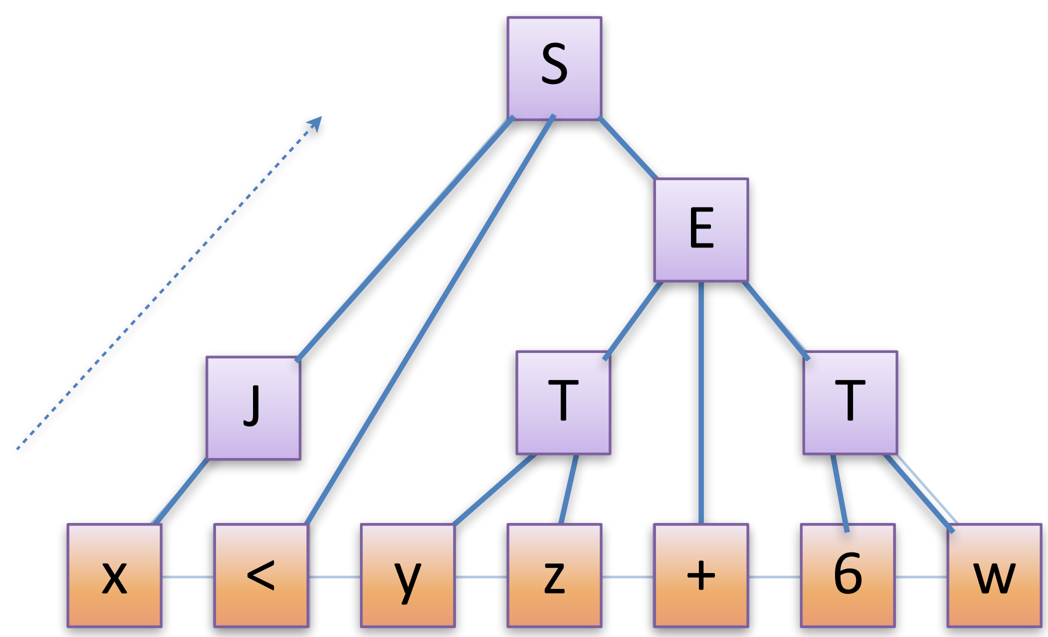
$S \longrightarrow J < E$
$J \longrightarrow id$
$E \longrightarrow T + T$
$T \longrightarrow id\ id \mid num\ id$

- Top-down (predictive)



- Bottom-up (shift-reduce)





Terminology: Reductions

- The opposite of derivation is called *reduction*
  - Let $\mathbf{A} \rightarrow \boldsymbol{\alpha}$ be a production rule
  - Derivation (step): $\beta A \mu \Rightarrow \beta \alpha \mu$
  - Reduction (step): $\beta \alpha \mu$ ➡ $\beta A \mu$

- A *reduction* is a sequence of reduction steps

Bottom-Up Parsing

- Goal: Build a parse tree*
  - Report error if the input is not a legal program

- How:
  - Scan input left-to-right
  - Construct subtrees once all their leaves have been scanned
    - Constructing the first left-most subtree whose children have been constructed
  - **Reduces** the input to the start symbol
    - Finds a rightmost derivation (in reverse order)

* Actually, we want to construct an abstract syntax tree (AST) ; we'll get to it later

Rightmost Derivation

```
x := z;
y := x + z
```

```
S → S;S | id := E
E → id | num | E + E
```

```
              S
          S ; S
          S ; id := E
          S ; id := E + E
          S ; id := E + id
          S ; id := id + id
    id := E ; id := id + id
    id := id ; id := id + id
```

<id,”x”> ASS <id,”z”> ; <id,”y”> ASS <id,”x”> PLUS <id,”z”>

```
S → S;S
S → id := E
E → E + E
E → id
E → id
S → id := E
E → id
```

A Reduction in the reverse order of a Rightmost Derivation

```
x := z;
y := x + z
```

S → S;S | id := E

E → id | num | E **+** E

S	S → S;S
S ; S	S → id := E
S ; id := E	E → E + E
S ; id := E + E	E → id
S ; id := E + id	E → id
S ; id := id + id	S → id := E
id := E ; id := id + id	E → id
id := id ; id := id + id	

<id,"x"> ASS <id,"z"> ; <id,"y"> ASS <id,"x"> PLUS <id,"z">

Reduction Example

$$E \rightarrow E * B \mid E + B \mid B$$

$$B \rightarrow 0 \mid 1$$

- Let us number the rules:

  (1) $E \rightarrow E * B$

  (2) $E \rightarrow E + B$

  (3) $E \rightarrow B$

  (4) $B \rightarrow 0$

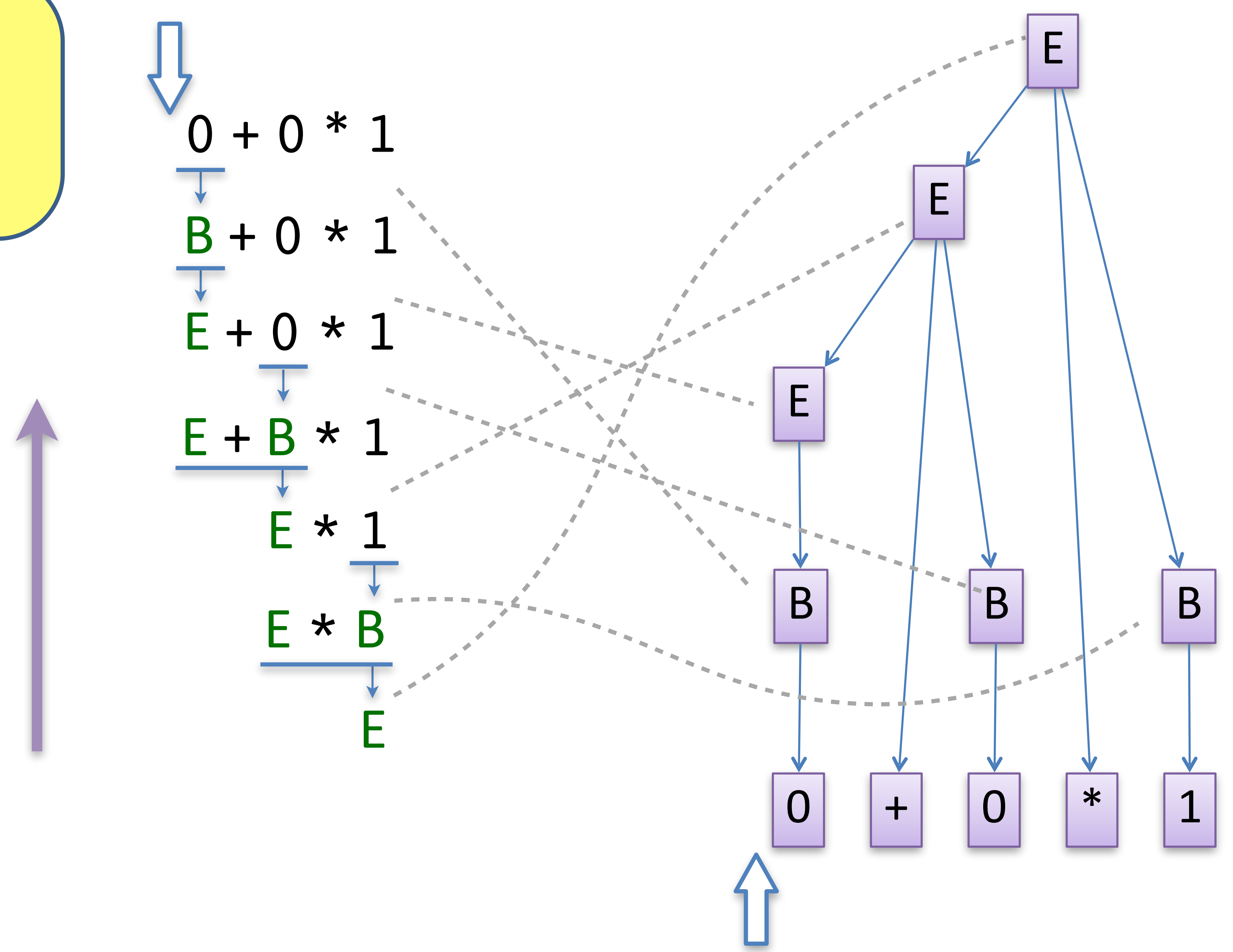
  (5) $B \rightarrow 1$

Goal: Reduce the Input to the Start Symbol

Building the parse tree during the reduction

$$E \longrightarrow \overset{(1)}{E * B} \mid \overset{(2)}{E + B} \mid \overset{(3)}{B}$$

$$B \longrightarrow \underset{(4)}{0} \mid \underset{(5)}{1}$$



Rightmost derivation

Let **A** ⟶ **α** be a production rule

- Derivation: $\beta A \mu \Rightarrow \beta \alpha \mu$
- Reduction: $\beta \alpha \mu \Rightarrow \beta A \mu$

Goal: Reduce the Input to the Start Symbol

$$E \longrightarrow \overset{(1)}{E * B} \mid \overset{(2)}{E + B} \mid \overset{(3)}{B}$$

$$B \longrightarrow \underset{(4)}{0} \mid \underset{(5)}{1}$$

Rightmost derivation

0 + 0 * 1

B + 0 * 1

E + 0 * 1

E + B * 1

E * 1

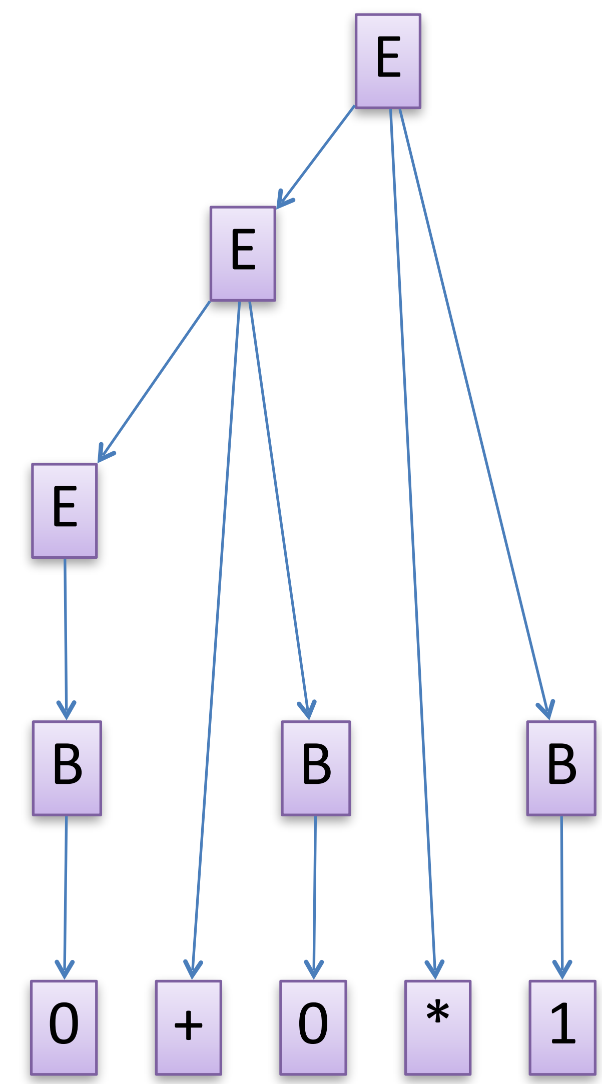
E * B

E

Using (3) would be a mistake … *(check it out…)*

Using (3) would be a mistake … *(check it out…)*

Building the parse tree during the reduction



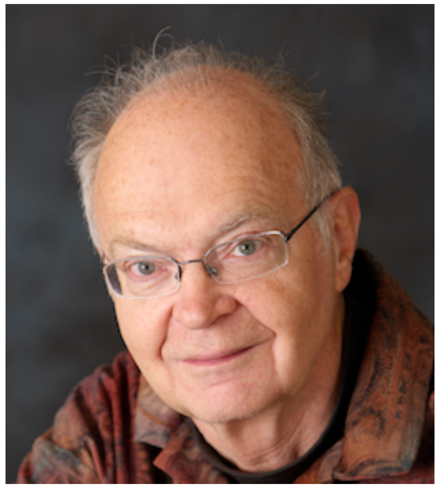
Let **A** $\longrightarrow$ **α** be a production rule

- Derivation: $\beta A \mu \Rightarrow \beta \alpha \mu$
- Reduction: $\beta \alpha \mu$ ➡ $\beta A \mu$

LR(*k*) Grammars

- A grammar is in the class LR(*k*) when it can be derived via:
  - Bottom-up analysis
  - Scanning the input from left to right (L)
  - Producing the rightmost derivation (R)
    - In reverse order
  - With lookahead of k tokens (*k*)

Donald Knuth



- A language is said to be LR(*k*) if it has an LR(*k*) grammar

Note: The lookahead is different from the one in LL(*k*) grammars ...

How does the Magic Work?

- Main idea: Delay decision on which rule to reduce with next to the last possible minute

How does the Magic Work? (by Example)

$S \longrightarrow A \mid C$

$A \longrightarrow BC \mid BB \mid BbC \mid CB$

$B \longrightarrow bb$

$C \longrightarrow cc$

Regular Language!

L = { ? }

Is the grammar LL(1)?

Is the grammar LL(*k*) for some k?

How does the Magic Work? (by Example)

$S \rightarrow A \mid C$

$A \rightarrow BC \mid BB \mid BbC \mid CB$

$B \rightarrow bb$

$C \rightarrow cc$

<u>Rightmost derivation</u>

w = bbbcc

$S \Rightarrow$

$A \Rightarrow$

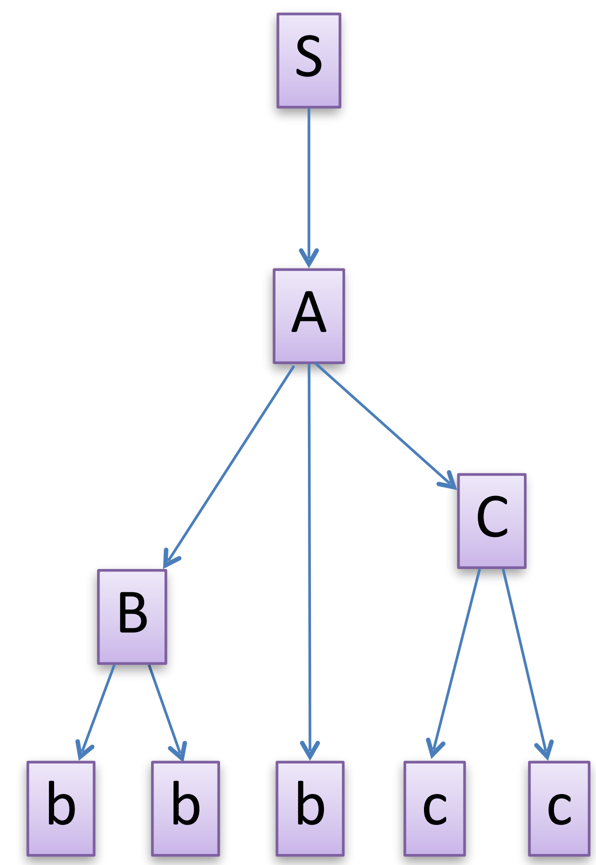
$BbC \Rightarrow$

$Bbcc \Rightarrow$

$bbbcc$

desired reduction

<u>Parse tree</u>



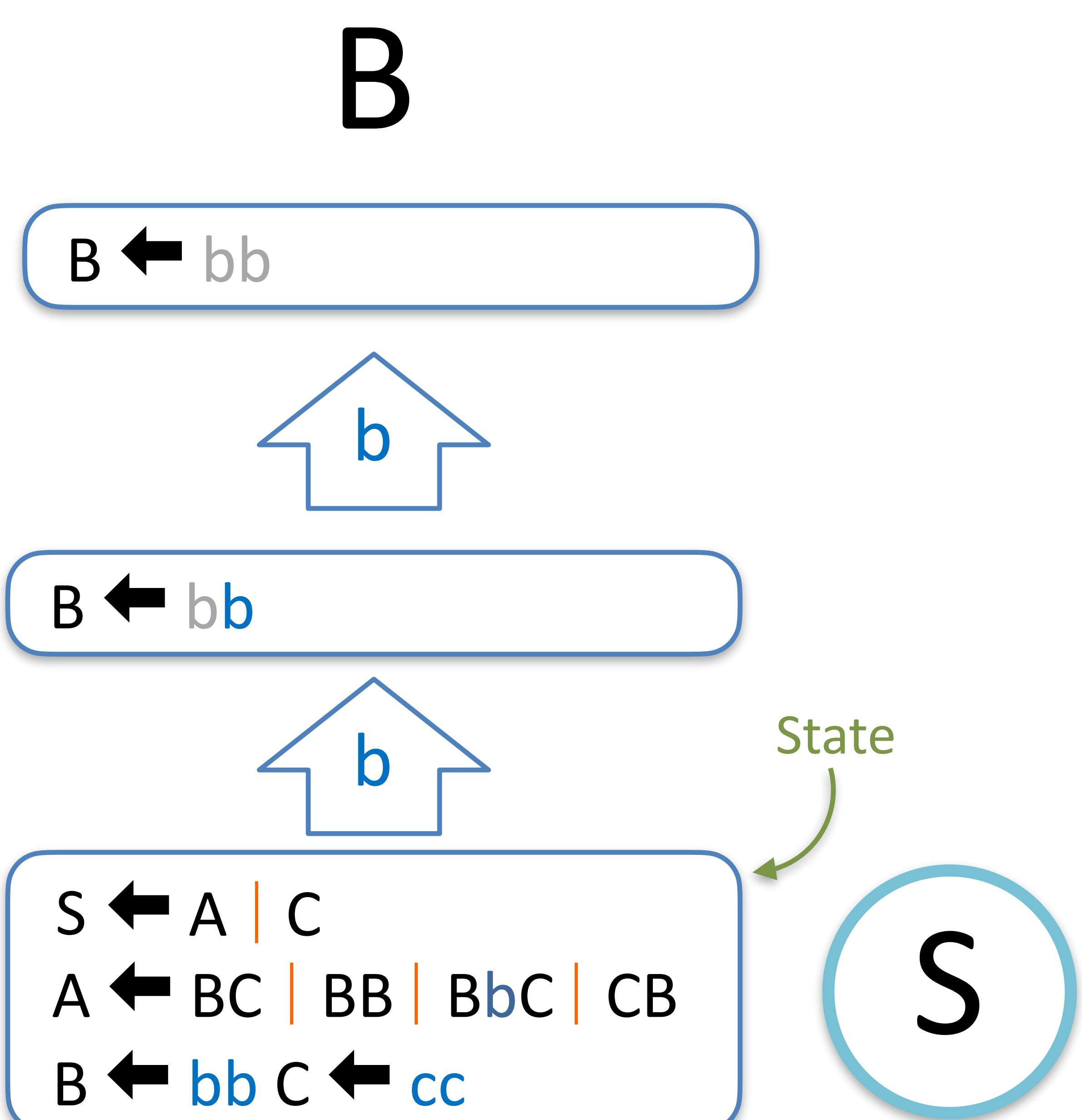
How does the Magic Work? (by Example)

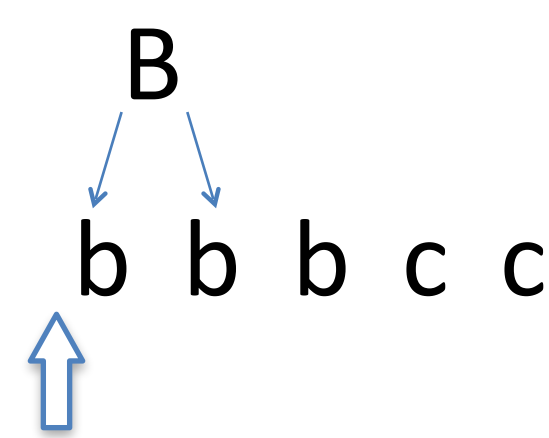
$S \rightarrow A \mid C$

$A \rightarrow BC \mid BB \mid BbC \mid CB$

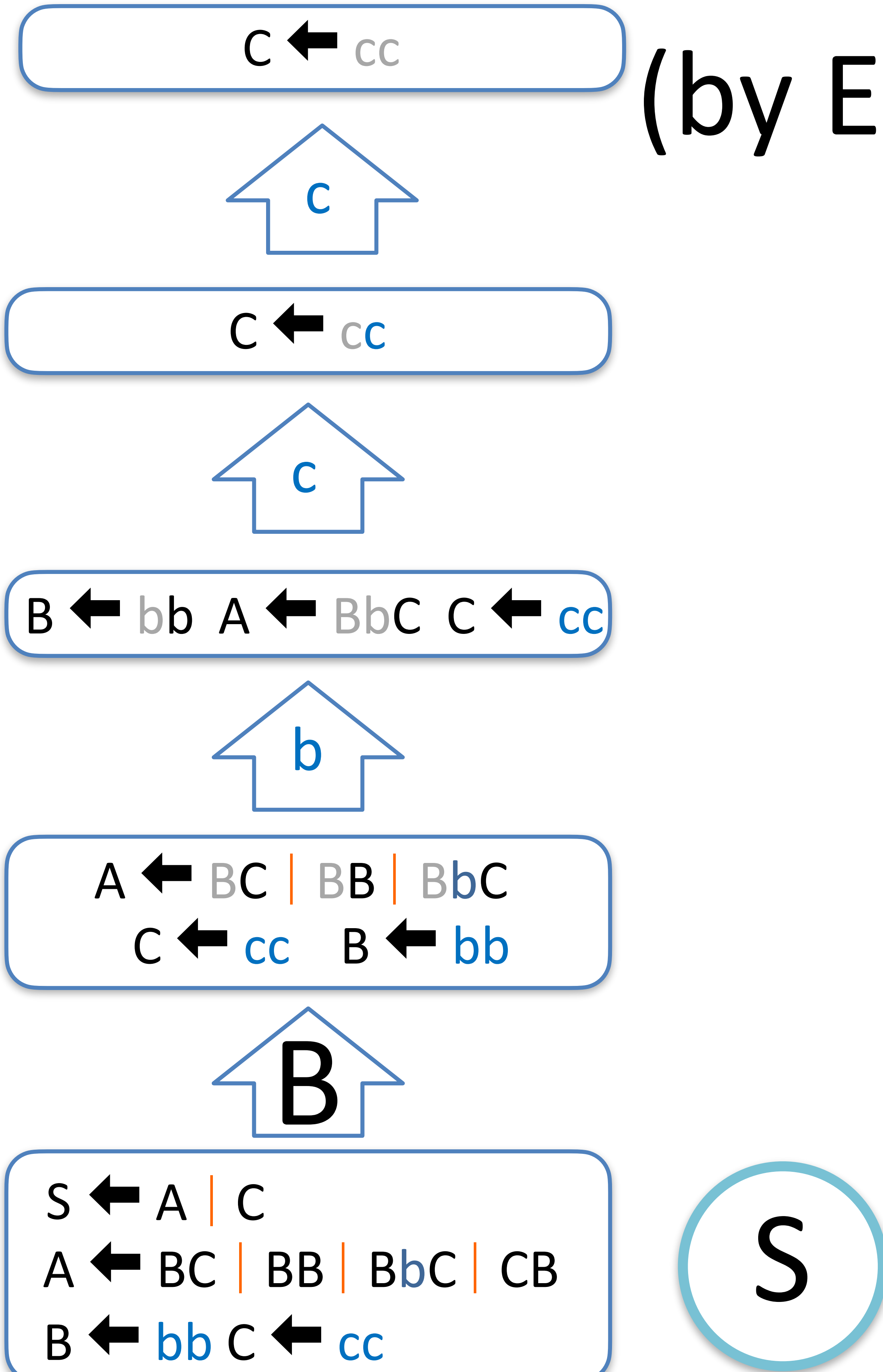
$B \rightarrow bb$

$C \rightarrow cc$





C How does the Magic Work? (by Example)

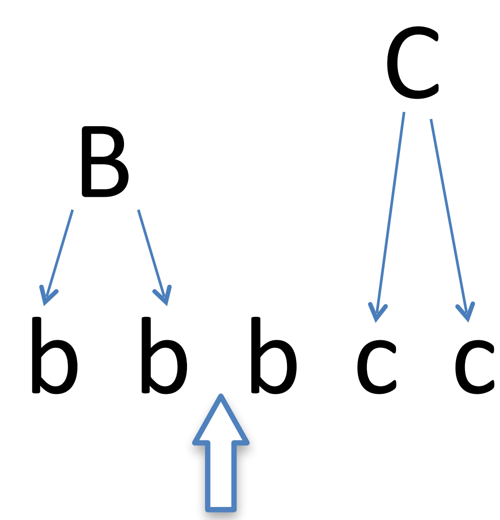


S → A | C

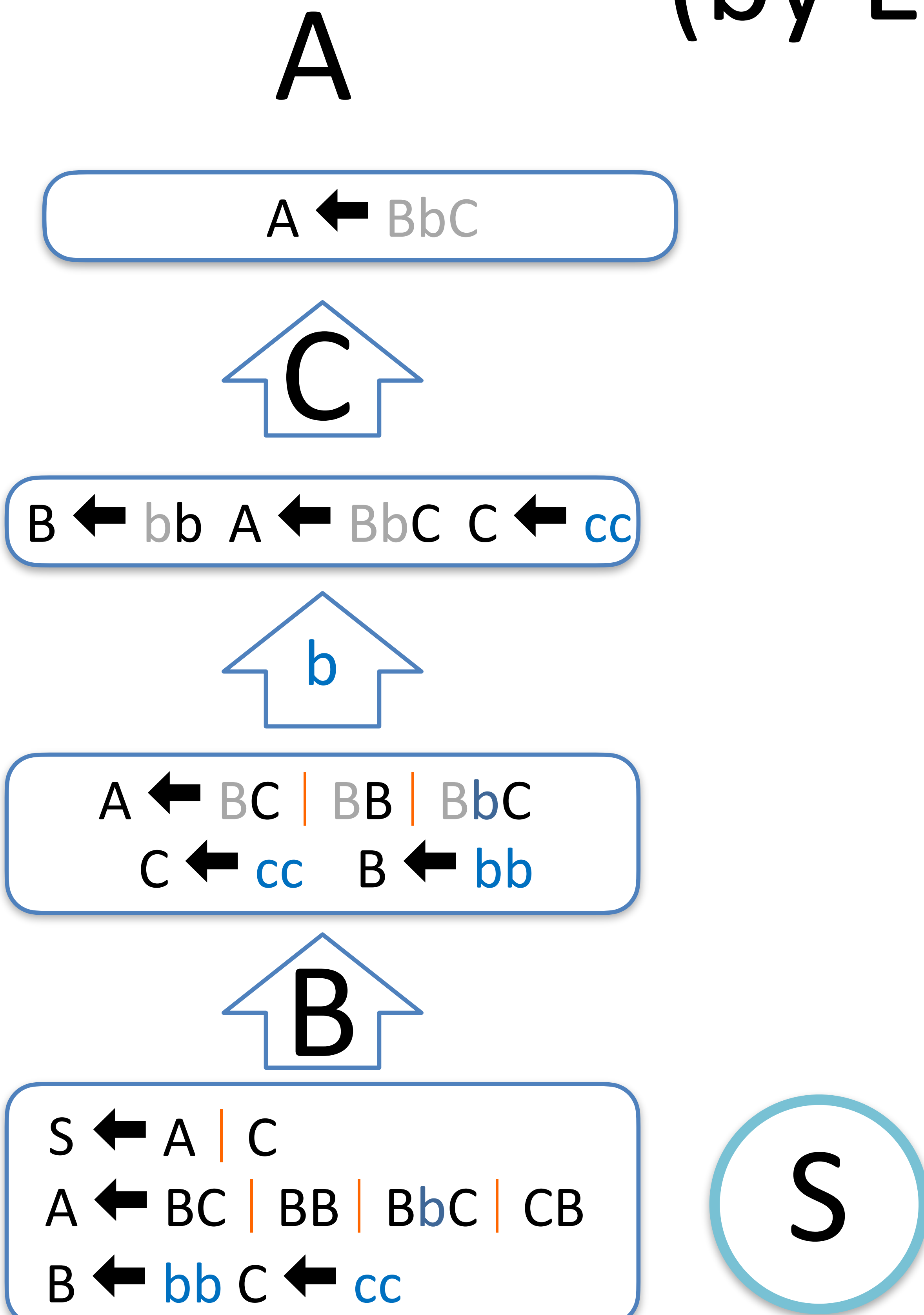
A → BC | BB | BbC | CB

B → bb

C → cc



How does the Magic Work? (by Example)

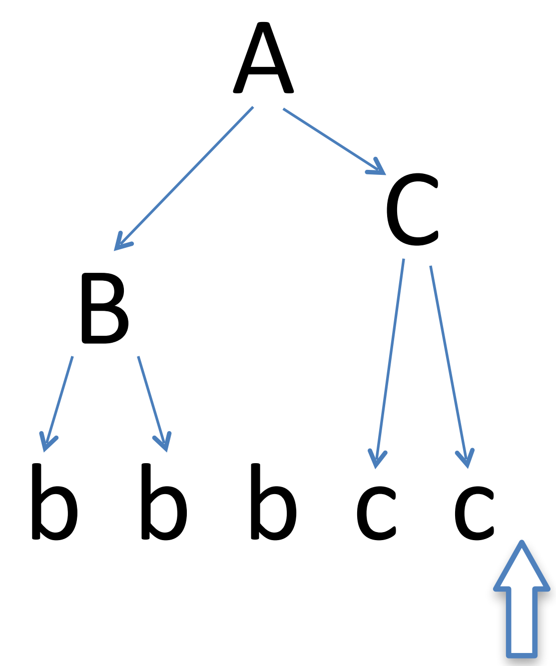


$S \rightarrow A \mid C$

$A \rightarrow BC \mid BB \mid BbC \mid CB$

$B \rightarrow bb$

$C \rightarrow cc$



How does the Magic Work? (by Example)

BTW. what data structure did we use?

S

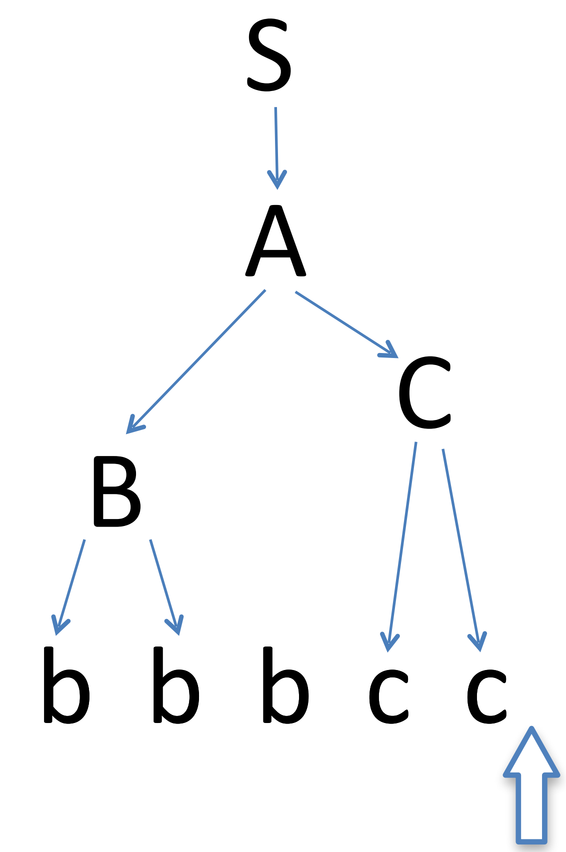
S → A

A

S ← A | C
A ← BC | BB | BbC | CB
B ← bb C ← cc

S

S → A | C
A → BC | BB | BbC | CB
B → bb
C → cc



Shift & Reduce (LR) Parsers

The parser maintains a stack of grammar symbols and states

In each step, we either

shift moving a symbol from the input to the stack, and compute and push a new state

or

reduce popping the symbols and states pertaining to the right-hand side of the discovered (reduced) rule,
push the non-terminal at the left-hand side of the rule, and
compute and push a new state

Shift & Reduce (LR) Parsers

The parser maintains a stack of grammar symbols and states

B → bb

b

B → bb

b

S → A | C
A → BC | BB | BbC | CB
A → aa B → bb C → cc

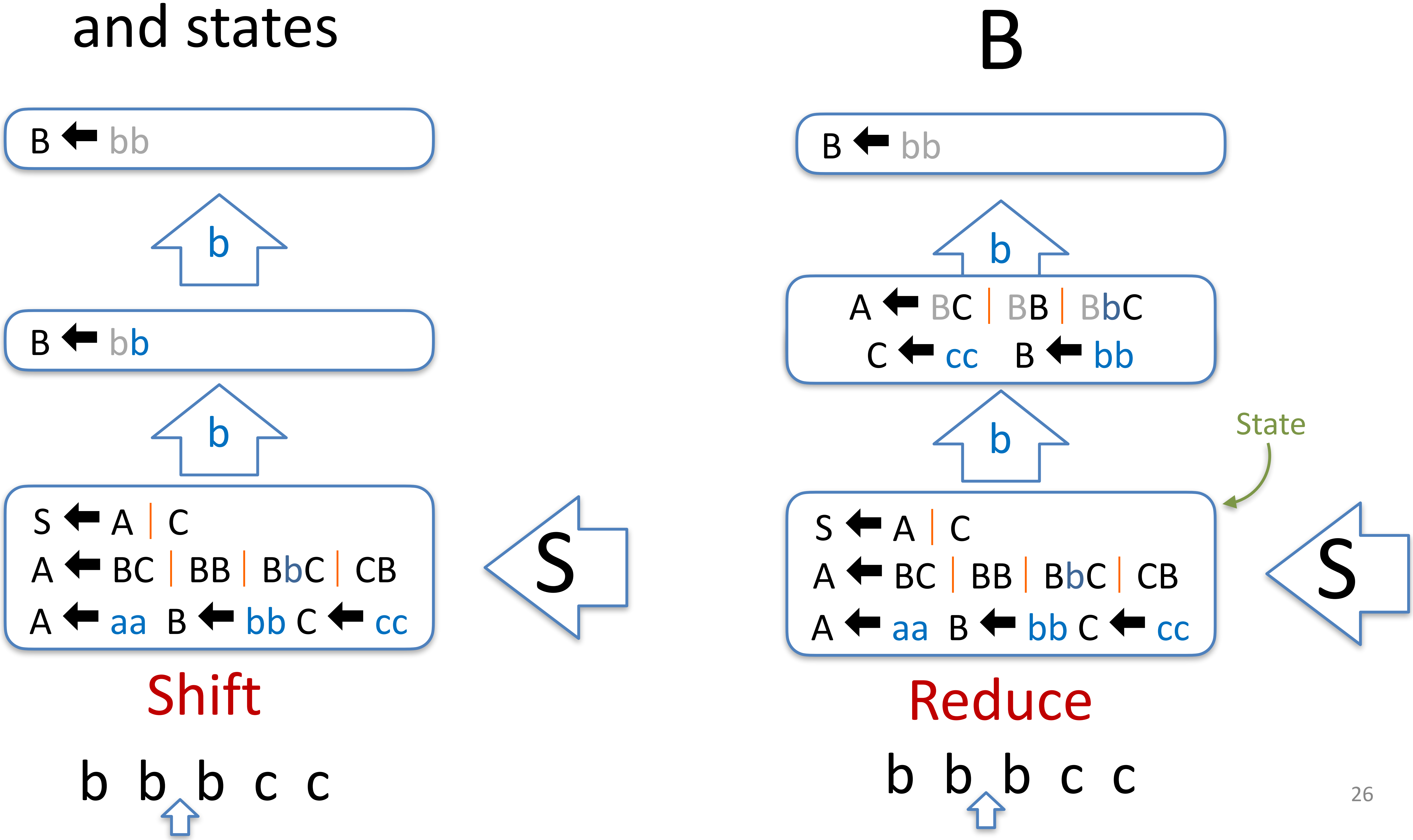
S

Shift

b b b c c

Shift & Reduce (LR) Parsers

The parser maintains a stack of grammar symbols and states



Important Bottom-Up LR-Parsers

Does an LL(0) parser make sense?

- LR(0) – simplest, explains basic ideas
- SLR(1) – simple, explains lookahead
- LR(1) – complicated, very powerful, expensive
- LALR(1) – complicated, powerful enough, used by automatic tools

The lookahead is different from the one in LL(*k*) grammars:

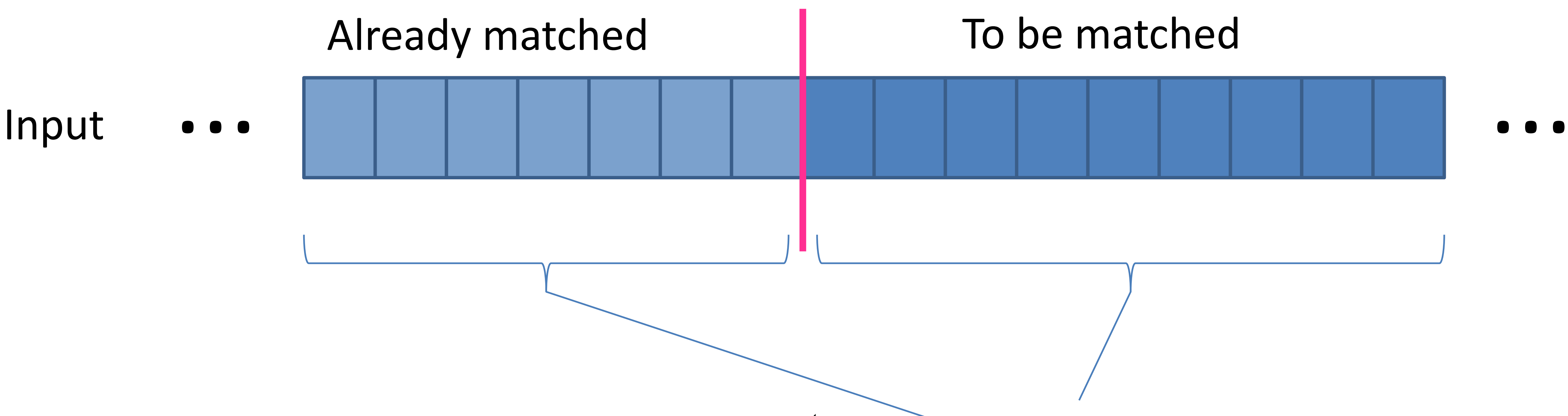
LL(k): k = number of tokens we look ahead to do a prediction

LR(k): *k* = number of tokens we look ahead when we do a reduce operation

LR(0)

"LR(0) Item"

For a production rule $N \rightarrow \alpha\ \beta$ in the grammar,



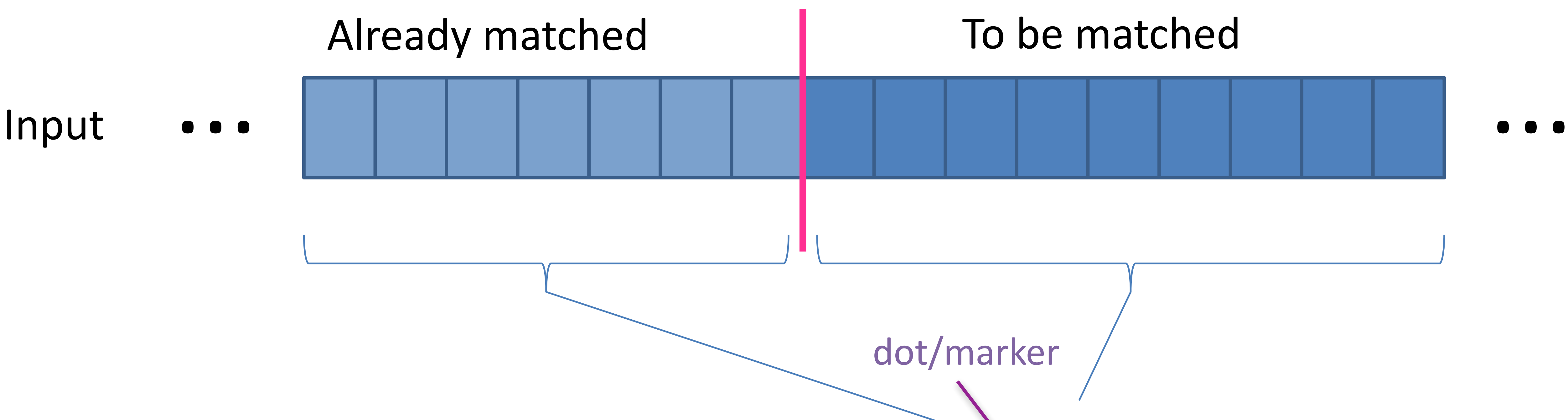
$$N \Leftarrow \alpha \quad \beta$$

||

So far we've matched α, expecting to see β

LR(0) Item

For a production rule $N \rightarrow \alpha\ \beta$ in the grammar,



$$N \rightarrow \alpha \bullet \beta$$

||

So far we've matched α, expecting to see β

LR(0) Item

$S \rightarrow A \mid C$

$A \rightarrow BC \mid BB \mid \boxed{BbC} \mid CB$

$B \rightarrow bb$

$C \rightarrow cc$

$A \rightarrow \bullet BbC$ $A \rightarrow B \bullet bC$ $A \rightarrow Bb \bullet C$ Shift Item

$A \rightarrow BbC \bullet$ Reduce Item

LR(0) Item

$$E \rightarrow E * B \mid E + B \mid B$$

$$B \rightarrow 0 \mid \boxed{1}$$

?

Shift Item

?

Reduce Item

LR(0) Item

$$Z \longrightarrow E\ \$$$

$$E \longrightarrow T \mid \boxed{E + T}$$

$$T \longrightarrow \mathbf{i} \mid (E)$$

?

Shift Item

?

Reduce Item

Example: Parsing with LR(0) Items

- LR(0) state = set of LR(0) items
- LR(0) item = a derivation rule with a marker
  - *Separates already determined symbols (past) from the one we hope to determine (future)*

Convention: A single initial rule

Convention: $ marks end of input

$Z \rightarrow E\ \$$

$E \rightarrow T \mid E + T$

$T \rightarrow \mathbf{i} \mid (\ E\ )$

$Z \rightarrow \bullet E\ \$$

$E \rightarrow \bullet T$

$E \rightarrow \bullet E + T$

$T \rightarrow \bullet \mathbf{i}$

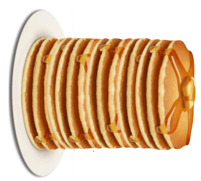
$T \rightarrow \bullet (\ E\ )$

Initial state

input

i + i $

Shift



i

$Z \rightarrow E\ \$$

$E \rightarrow T \mid E + T$

$T \rightarrow i \mid (\ E\ )$

No need to actually push states with reduce items to the stack

$Z \rightarrow \bullet E\ \$$

$E \rightarrow \bullet T$

$E \rightarrow \bullet E + T$

$T \rightarrow \bullet i$

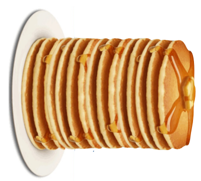
$T \rightarrow \bullet (\ E\ )$

$T \rightarrow i \bullet$

Reduce item!

input

i + i $



	T

i

$Z \longrightarrow \bullet E\ \$$

$E \longrightarrow \bullet T$

$E \longrightarrow \bullet E + T$

$T \longrightarrow \bullet \mathbf{i}$

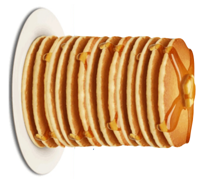
$T \longrightarrow \bullet (E)$

$Z \longrightarrow E\ \$$

$E \longrightarrow T \mid E + T$

$T \longrightarrow \mathbf{i} \mid (E)$

input **i + i $**



	T
	i

$Z \longrightarrow E\ \$$

$E \longrightarrow T \mid E + T$

$T \longrightarrow \mathbf{i} \mid (\ E\ )$

$Z \longrightarrow \bullet E\ \$$

$E \longrightarrow \bullet T$

$E \longrightarrow \bullet E + T$

$T \longrightarrow \bullet \mathbf{i}$

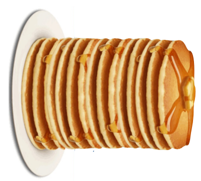
$T \longrightarrow \bullet (\ E\ )$

$E \longrightarrow T \bullet$

Reduce item!

input

i + **i** $



	E

T

i

$Z \longrightarrow E\ \$$

$E \longrightarrow T \mid E + T$

$T \longrightarrow \mathbf{i} \mid (\ E\ )$

$Z \longrightarrow \bullet E\ \$$

$E \longrightarrow \bullet T$

$E \longrightarrow \bullet E + T$

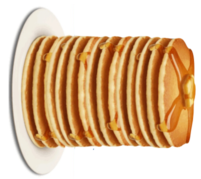
$T \longrightarrow \bullet \mathbf{i}$

$T \longrightarrow \bullet (\ E\ )$

input

i + i $

Shift



	E		+

T

i

$Z \rightarrow E\ \$$

$E \rightarrow T \mid E + T$

$T \rightarrow i \mid (E)$

$Z \rightarrow \bullet E\ \$$

$E \rightarrow \bullet T$

$E \rightarrow \bullet E + T$

$T \rightarrow \bullet i$

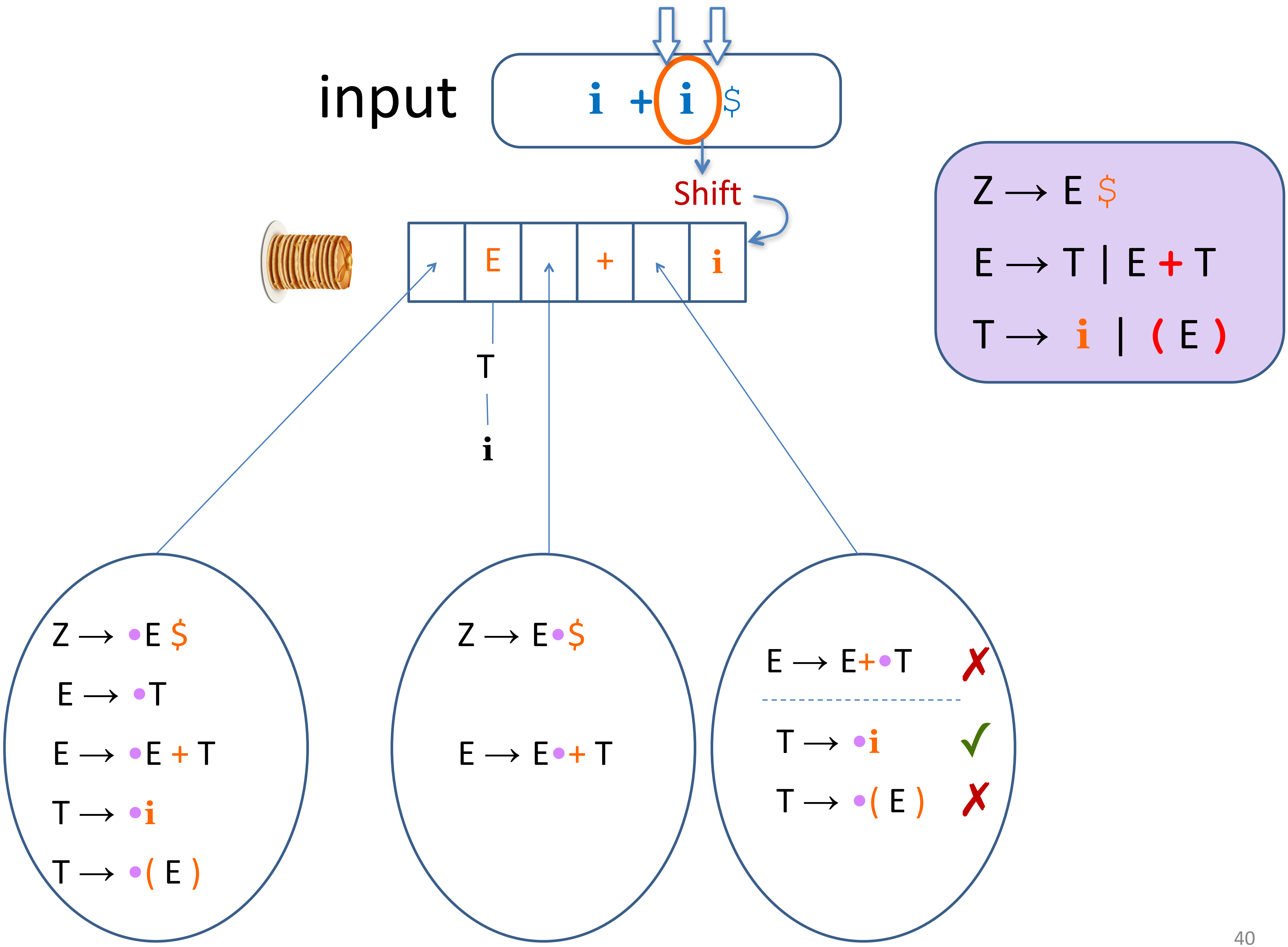
$T \rightarrow \bullet (E)$

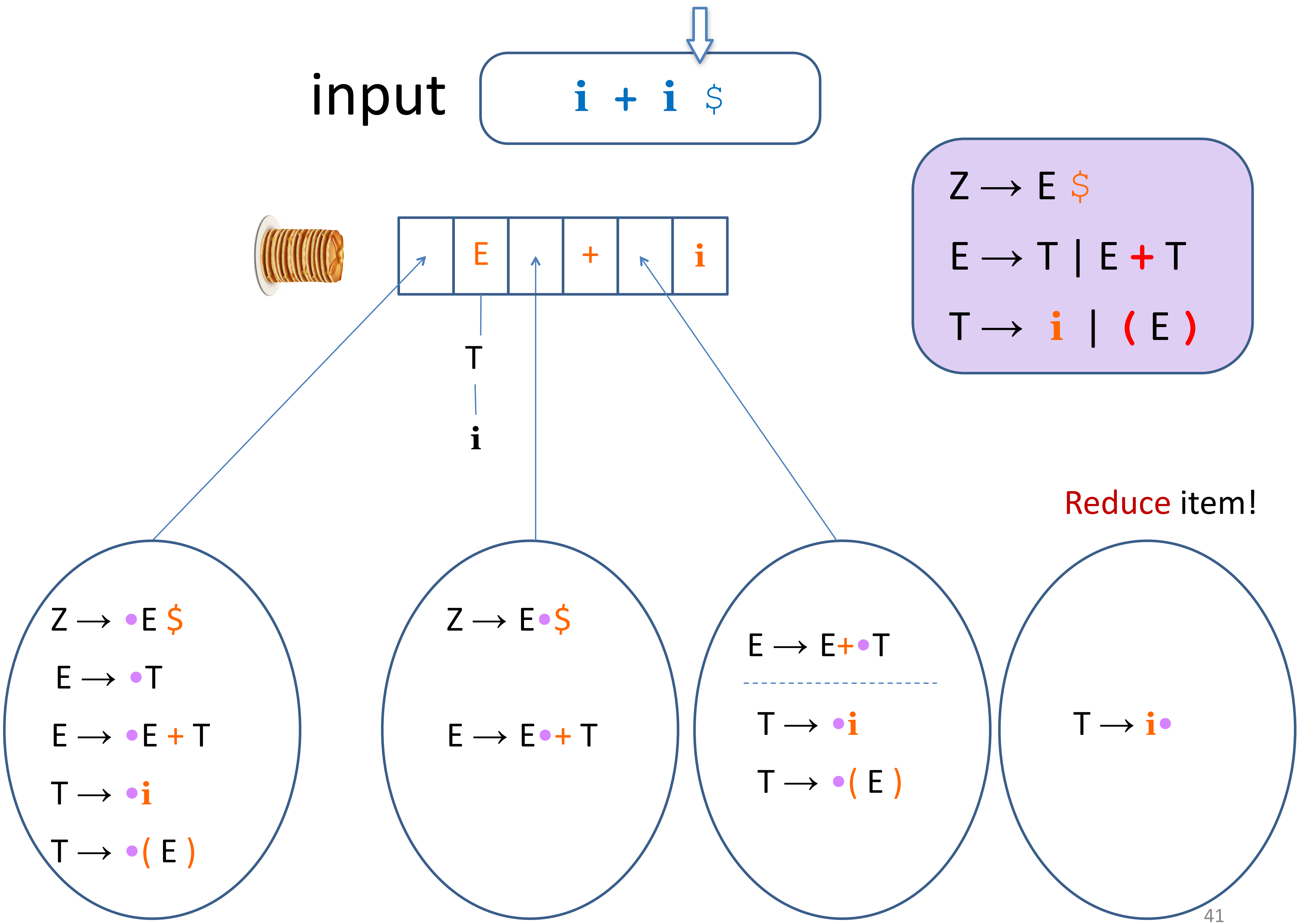
✗

$Z \rightarrow E \bullet \$$

$E \rightarrow E \bullet + T$

✓


input
i + i $
Shift
E + i
T
i
Z → E $
E → T | E + T
T → i | (E)
Z → •E $
E → •T
E → •E + T
T → •i
T → •(E)
Z → E•$
E → E•+ T
E → E+•T ✗
T → •i ✓
T → •(E) ✗

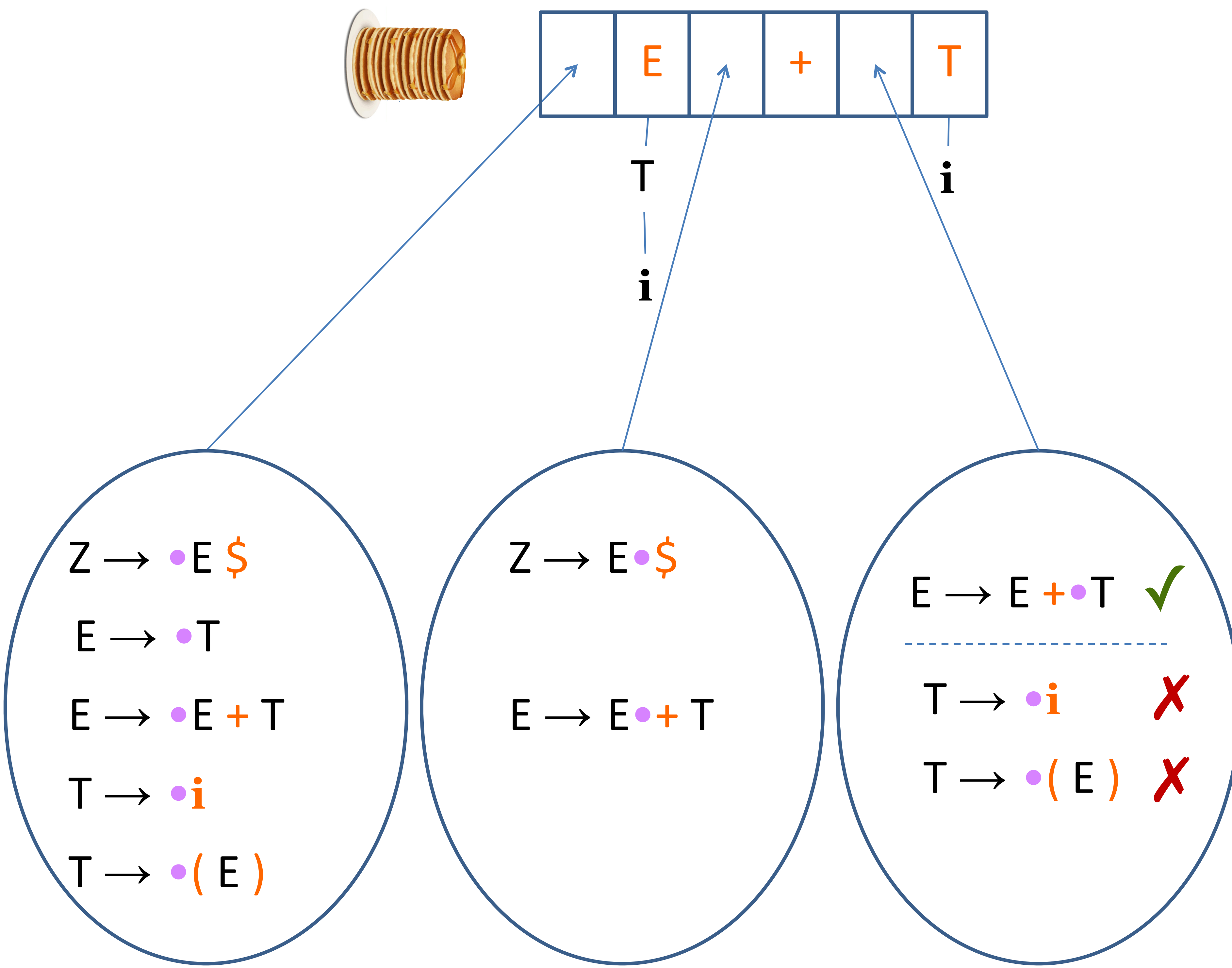

input
i + i $
E
+
i
T
i
Z → E $
E → T | E + T
T → i | (E)
Reduce item!
Z → •E $
E → •T
E → •E + T
T → •i
T → •(E)
Z → E•$
E → E•+ T
E → E+•T
T → •i
T → •(E)
T → i•

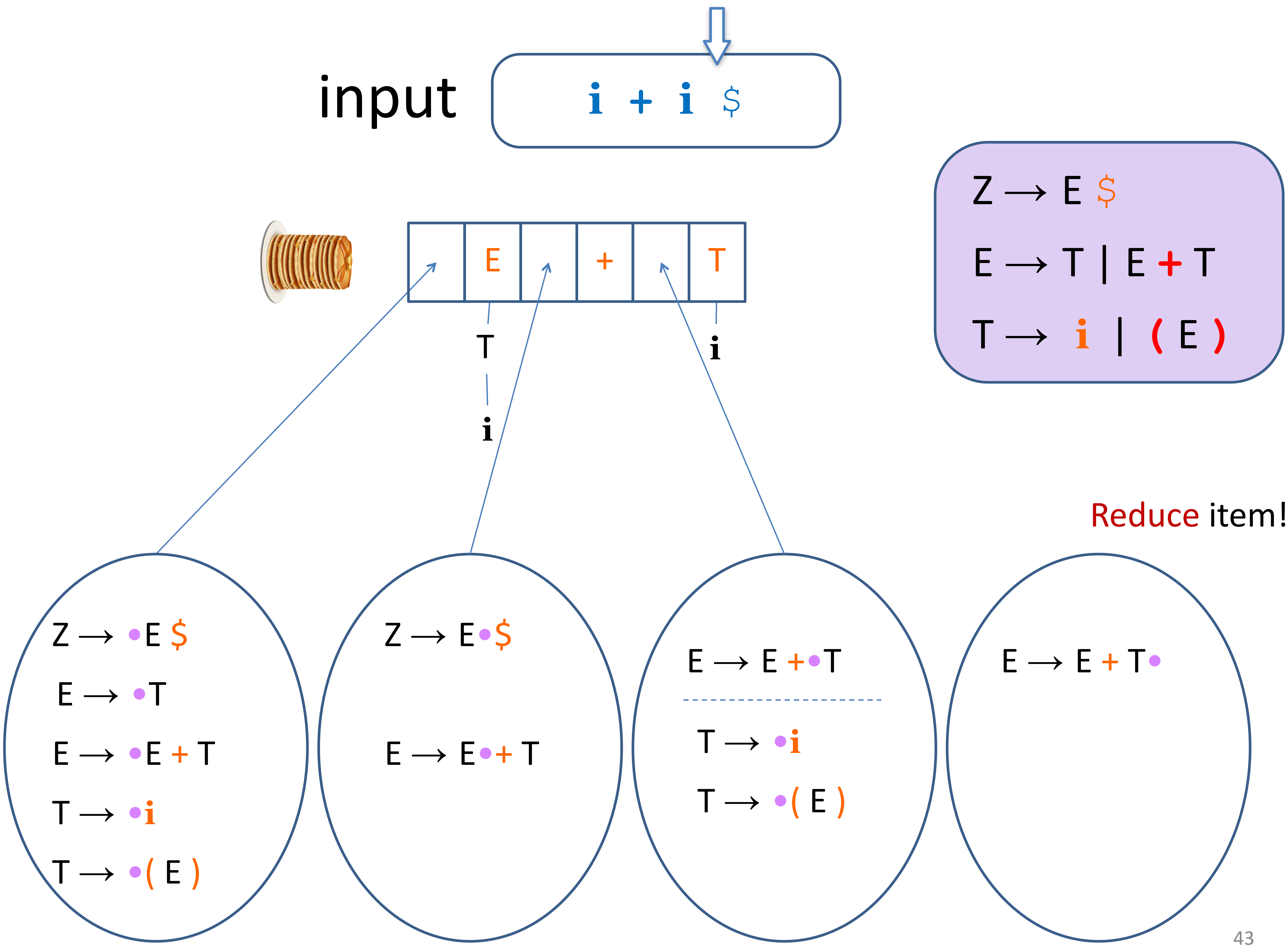
input $\quad$ **i + i $**

$Z \longrightarrow E\ \$$

$E \longrightarrow T \mid E + T$

$T \longrightarrow \mathbf{i} \mid (E)$

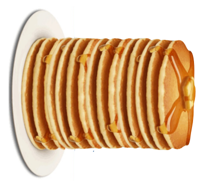


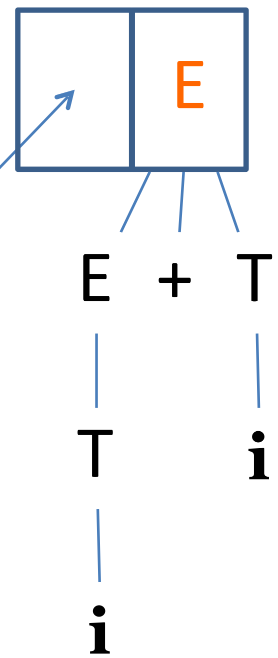


input
i + i $
Z ⟶ E $
E ⟶ T | E + T
T ⟶ i | (E)
E
+
T
T
i
i
Reduce item!
Z ⟶ •E $
E ⟶ •T
E ⟶ •E + T
T ⟶ •i
T ⟶ •(E)
Z ⟶ E•$
E ⟶ E•+ T
E ⟶ E +•T
T ⟶ •i
T ⟶ •(E)
E ⟶ E + T•

input

i + i $





$Z \rightarrow E\ \$$

$E \rightarrow T \mid E + T$

$T \rightarrow i \mid (\ E\ )$

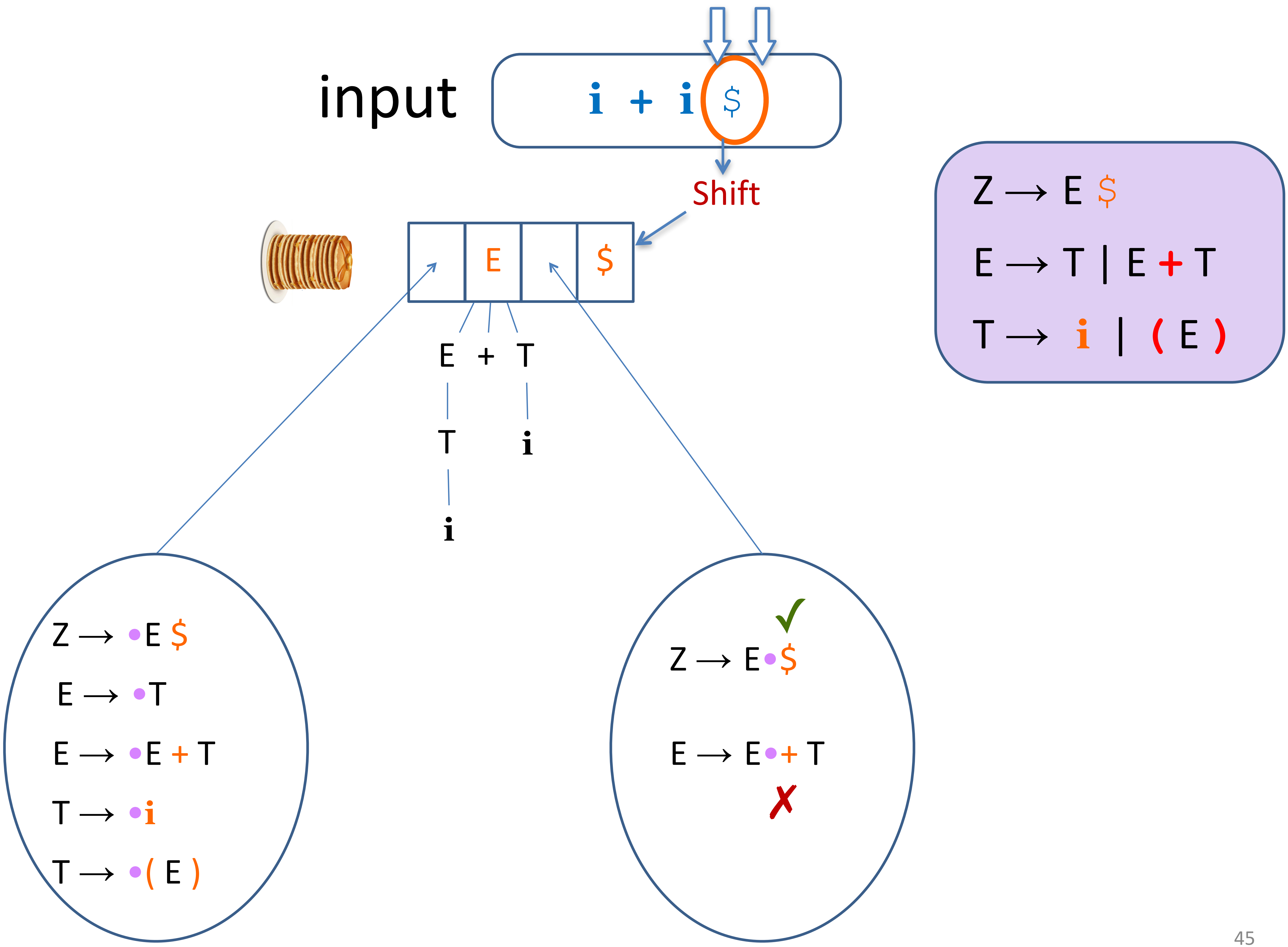
$Z \rightarrow \bullet E\ \$$

$E \rightarrow \bullet T$

$E \rightarrow \bullet E + T$

$T \rightarrow \bullet i$

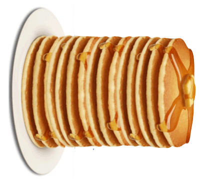
$T \rightarrow \bullet (\ E\ )$



input
i + i $
Shift
E
$
E + T
T
i
i
Z → E $
E → T | E + T
T → i | (E)
Z → •E $
E → •T
E → •E + T
T → •i
T → •(E)
Z → E•$
E → E•+ T

input

i + **i** $



	E		$

E + T

T   **i**

i

$Z \longrightarrow E\ \$$

$E \longrightarrow T \mid E + T$

$T \longrightarrow \mathbf{i} \mid (E)$

Reduce item!

$Z \longrightarrow \bullet E\ \$$

$E \longrightarrow \bullet T$

$E \longrightarrow \bullet E + T$

$T \longrightarrow \bullet \mathbf{i}$

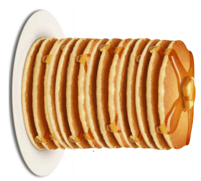
$T \longrightarrow \bullet (E)$

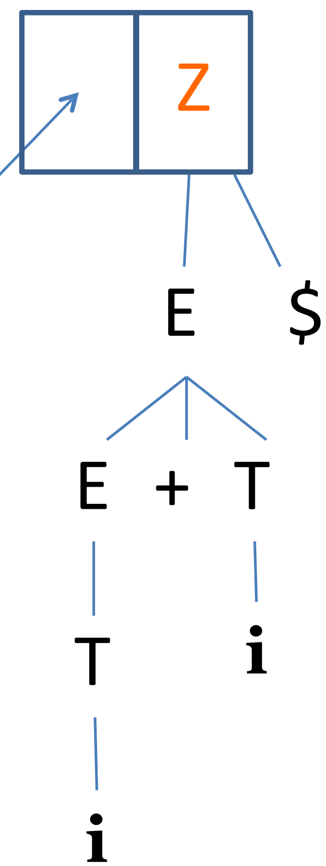
$Z \longrightarrow E \bullet \$$

$E \longrightarrow E \bullet + T$

$Z \longrightarrow E\ \$ \bullet$

input **i + i** $





$Z \rightarrow E\ \$$

$E \rightarrow T \mid E + T$

$T \rightarrow \mathbf{i} \mid (E)$

$Z \rightarrow \bullet E\ \$$

$E \rightarrow \bullet T$

$E \rightarrow \bullet E + T$

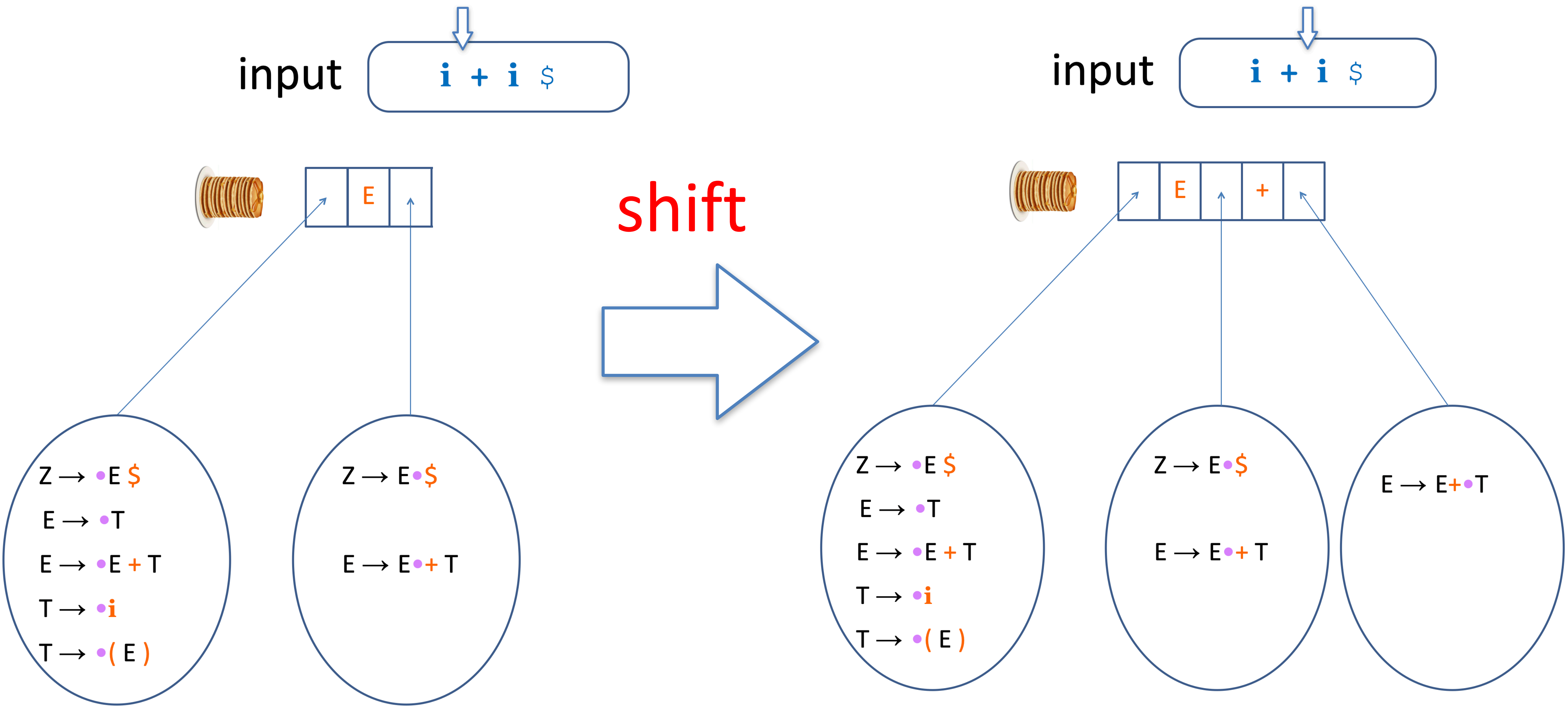
$T \rightarrow \bullet \mathbf{i}$

$T \rightarrow \bullet (E)$

Reducing the initial rule means accept

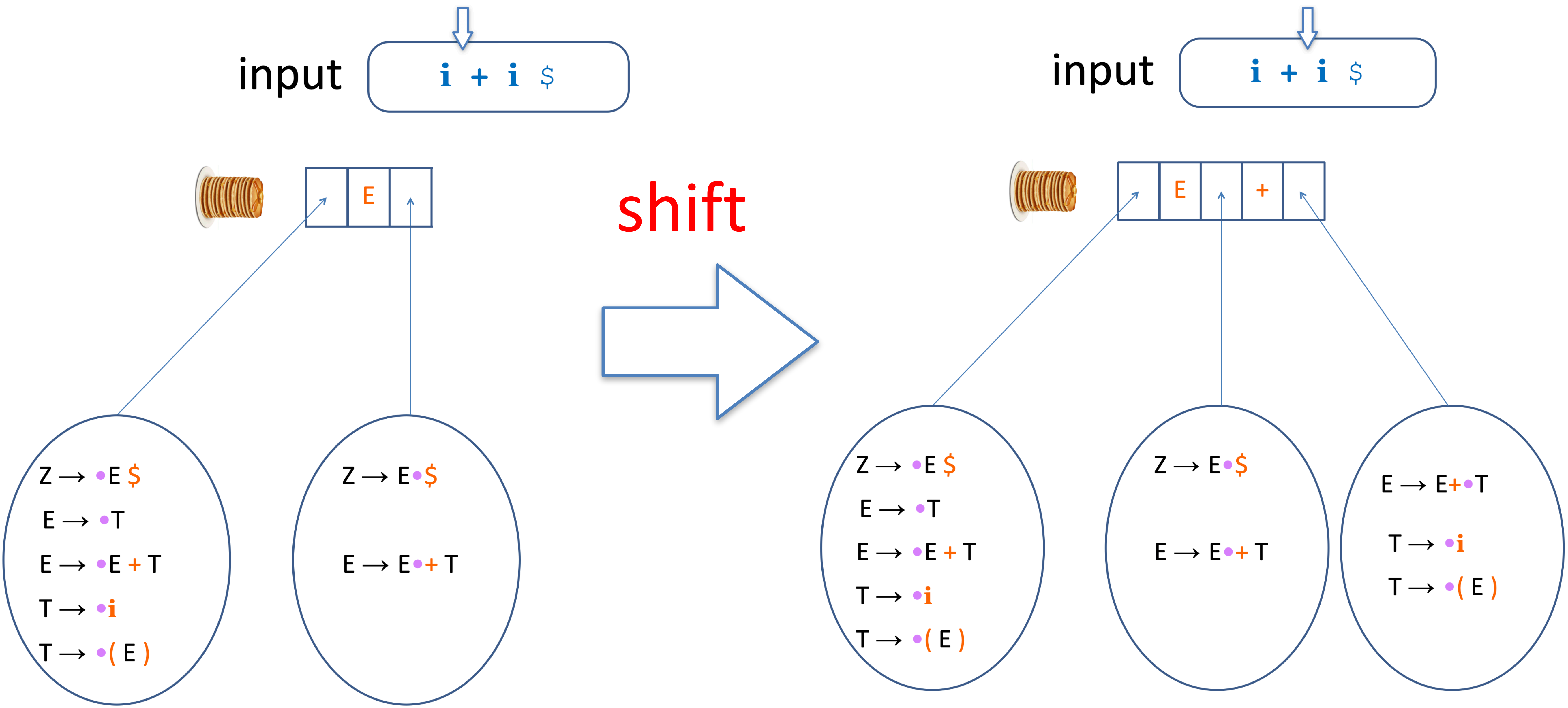
Computing State Transitions

- Do we need to compute the state transitions during parsing?
  - Or can we do it ahead?



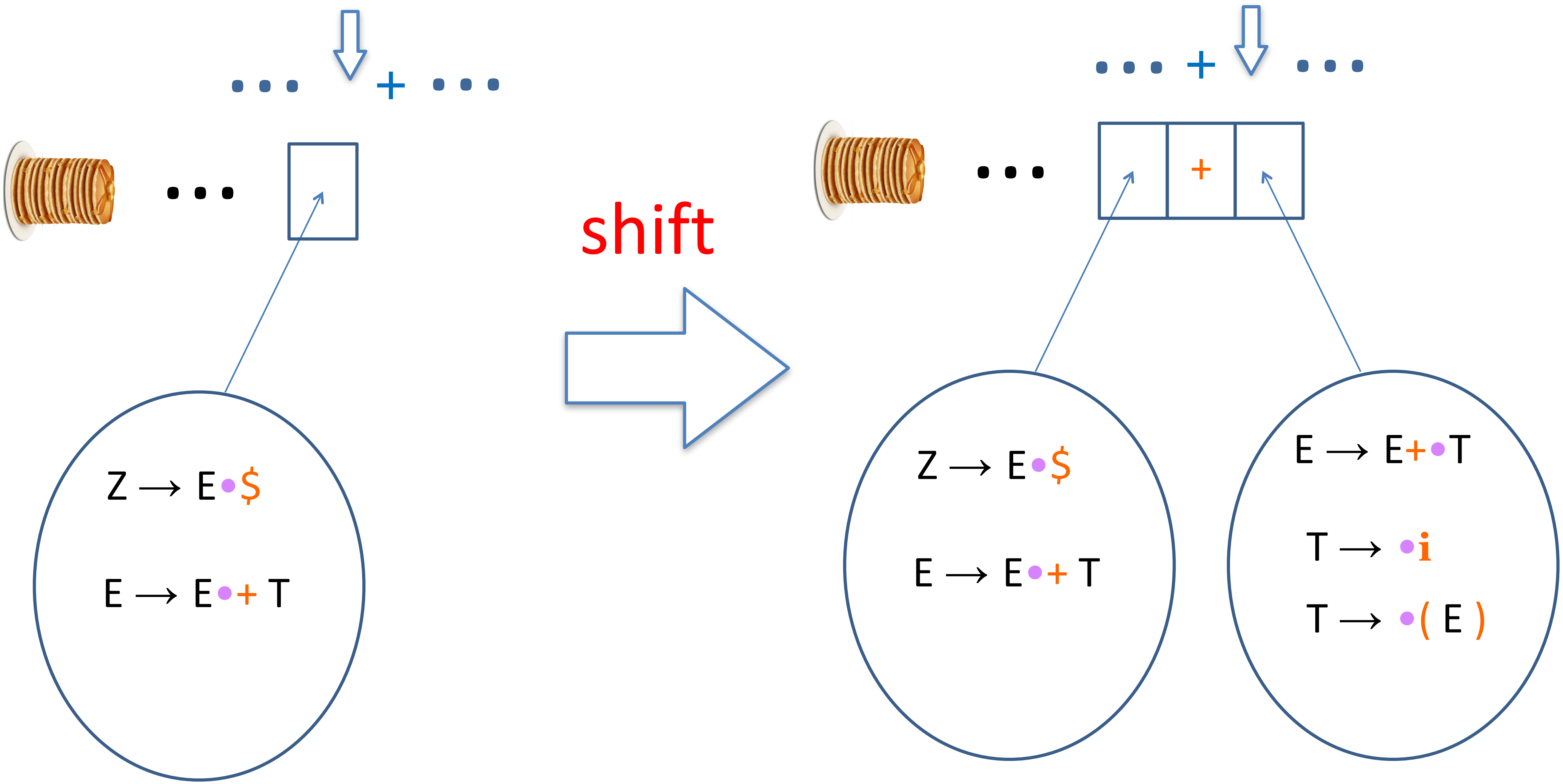
Computing State Transitions

- Do we need to compute the state transitions during parsing?
  - Or can we do it ahead?



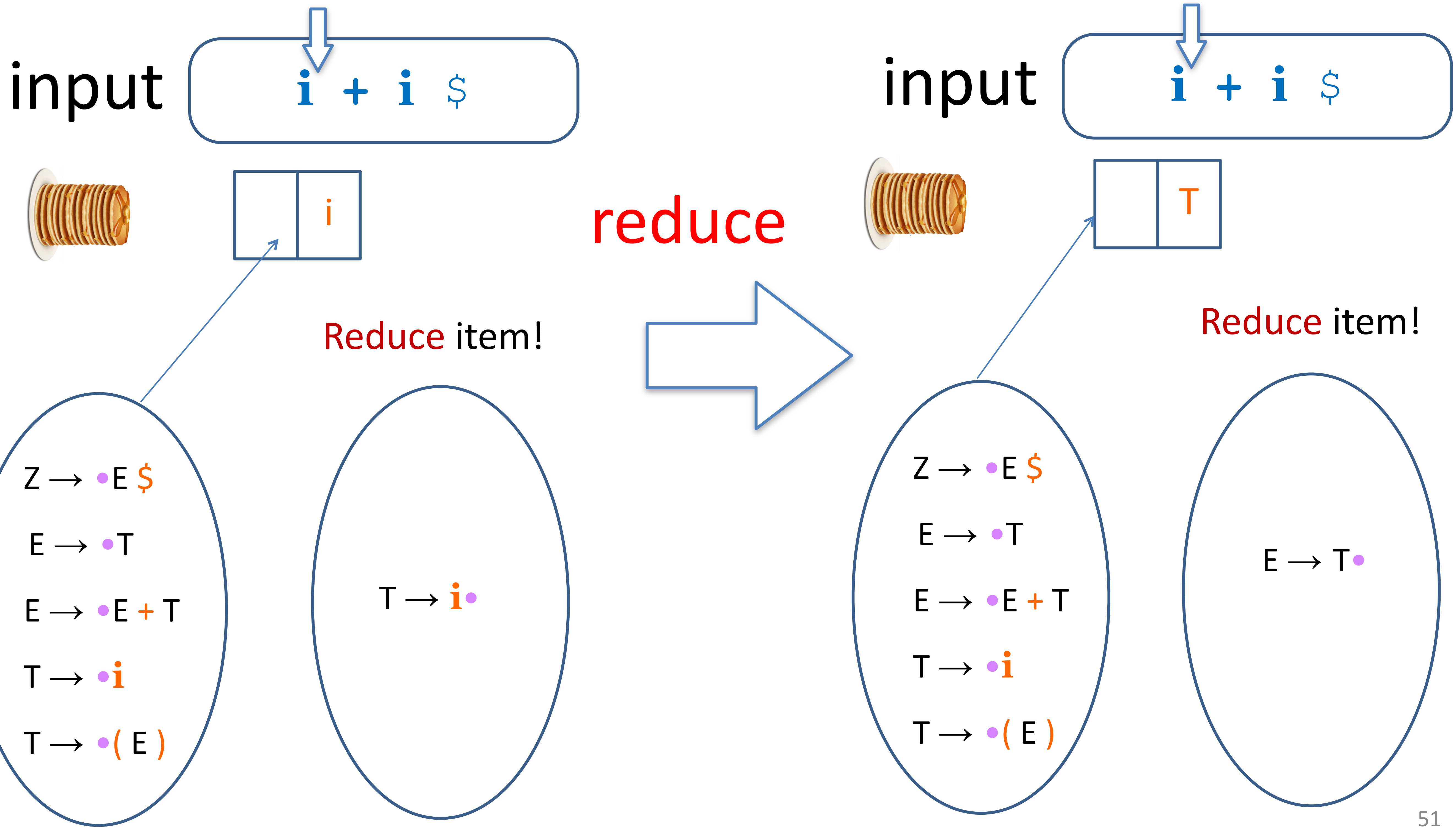
Computing State Transitions

- Do we need to compute the state transitions during parsing?
  - Or can we do it ahead?



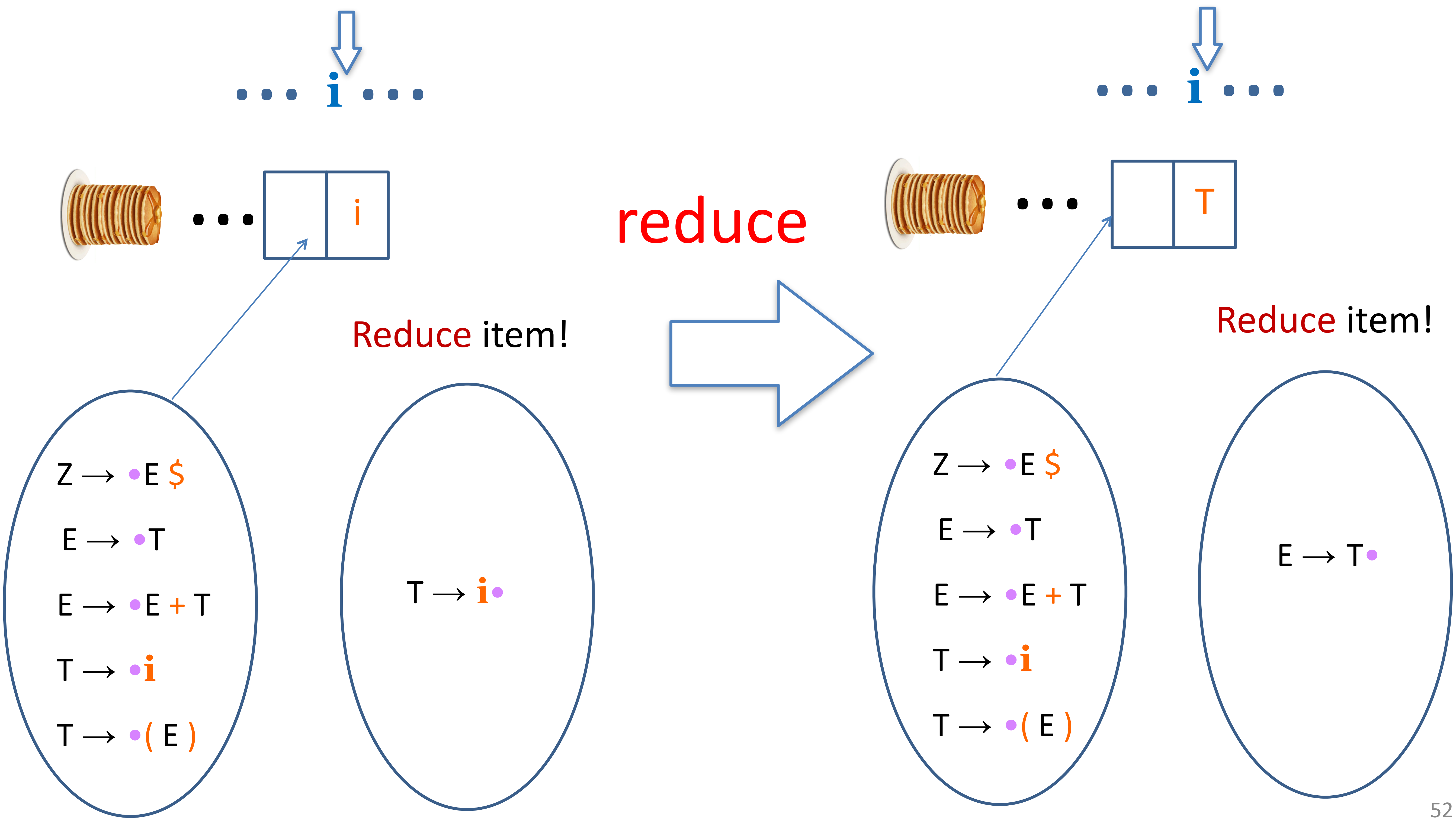
Computing State Transitions

- Do we need to compute the state transitions during parsing?
  - Or can we do it ahead?



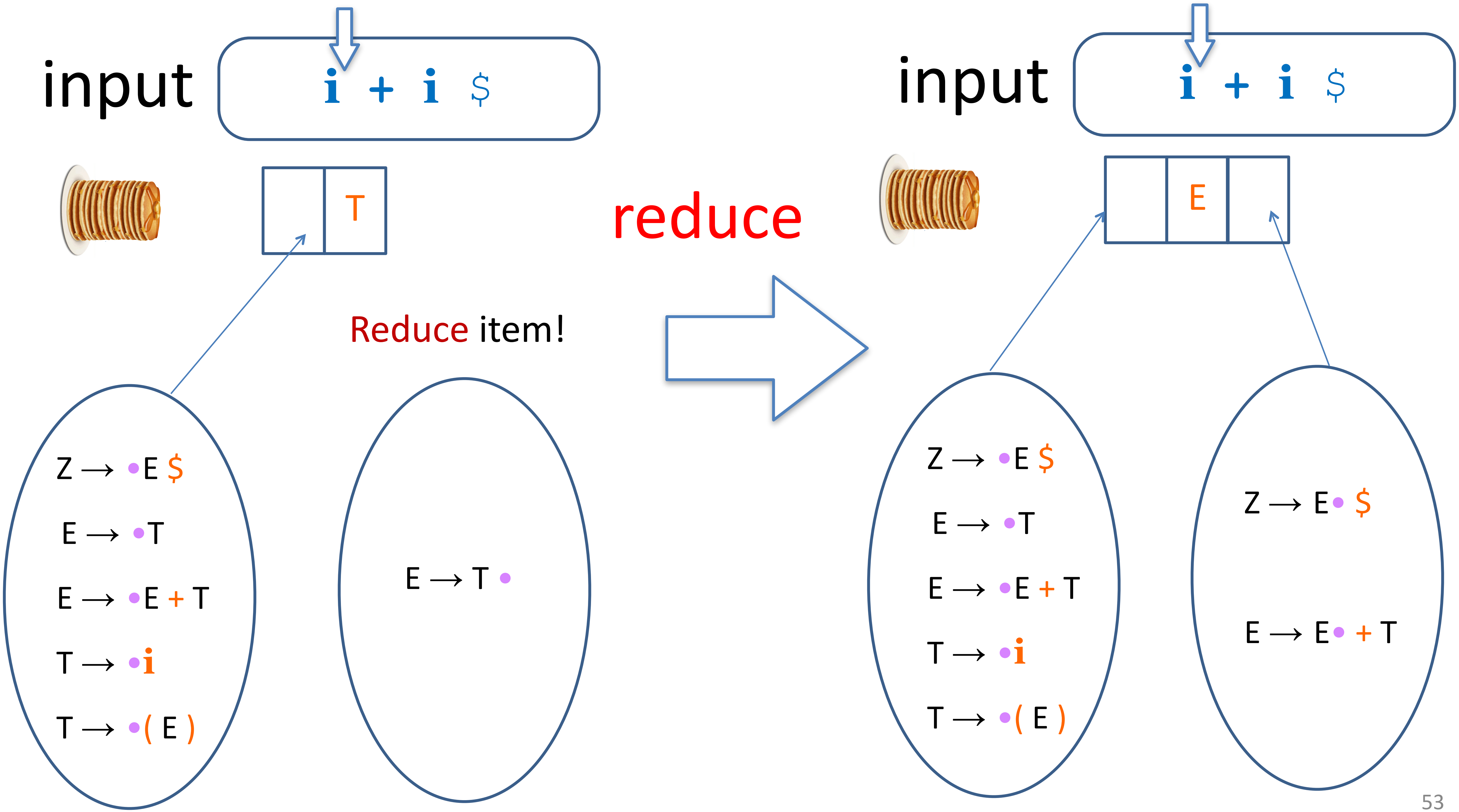
Computing State Transitions

- Do we need to compute the state transitions during parsing?
  - Or can we do it ahead?



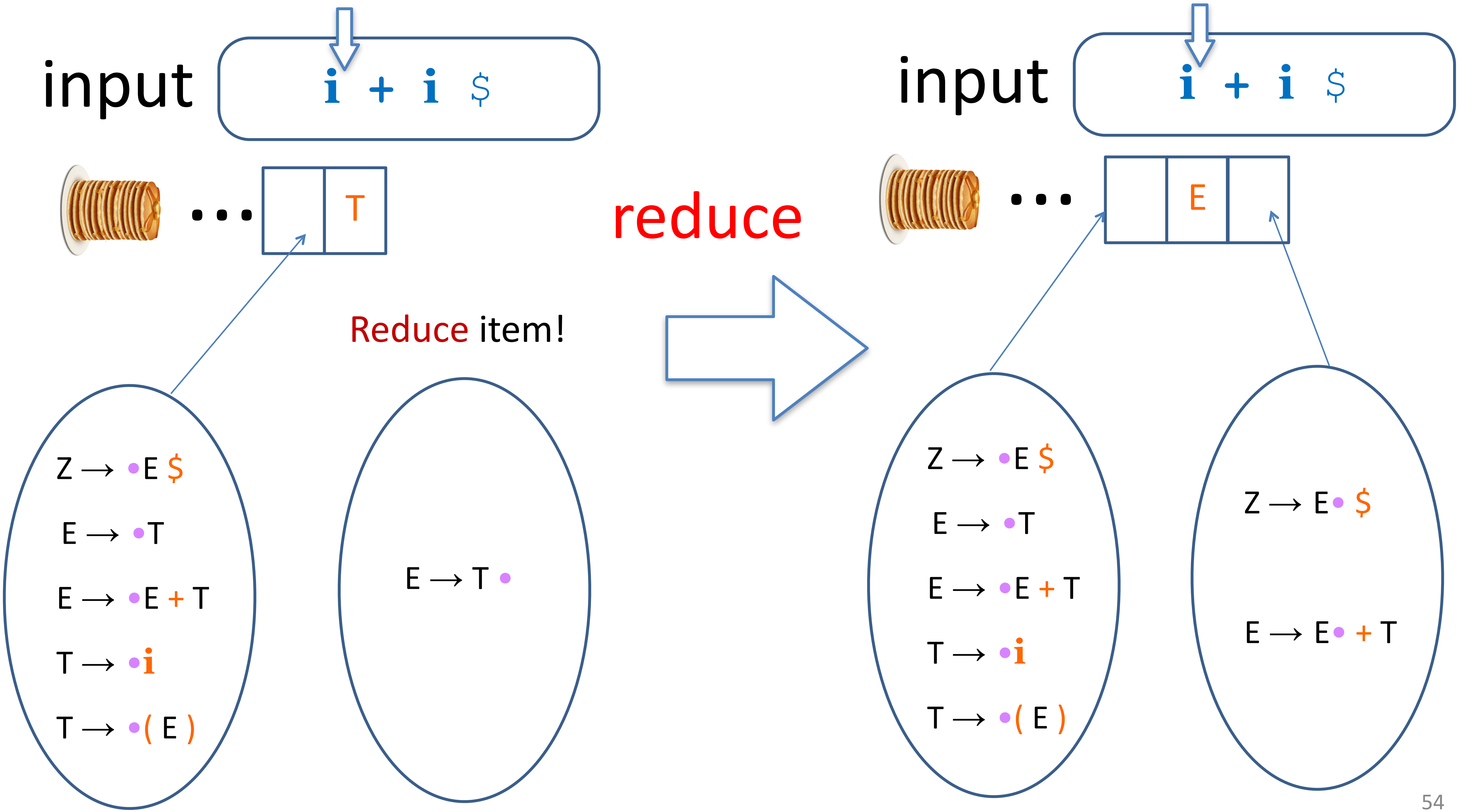
Computing State Transitions

- Do we need to compute the state transitions during parsing?
  - Or can we do it ahead?



Computing State Transitions

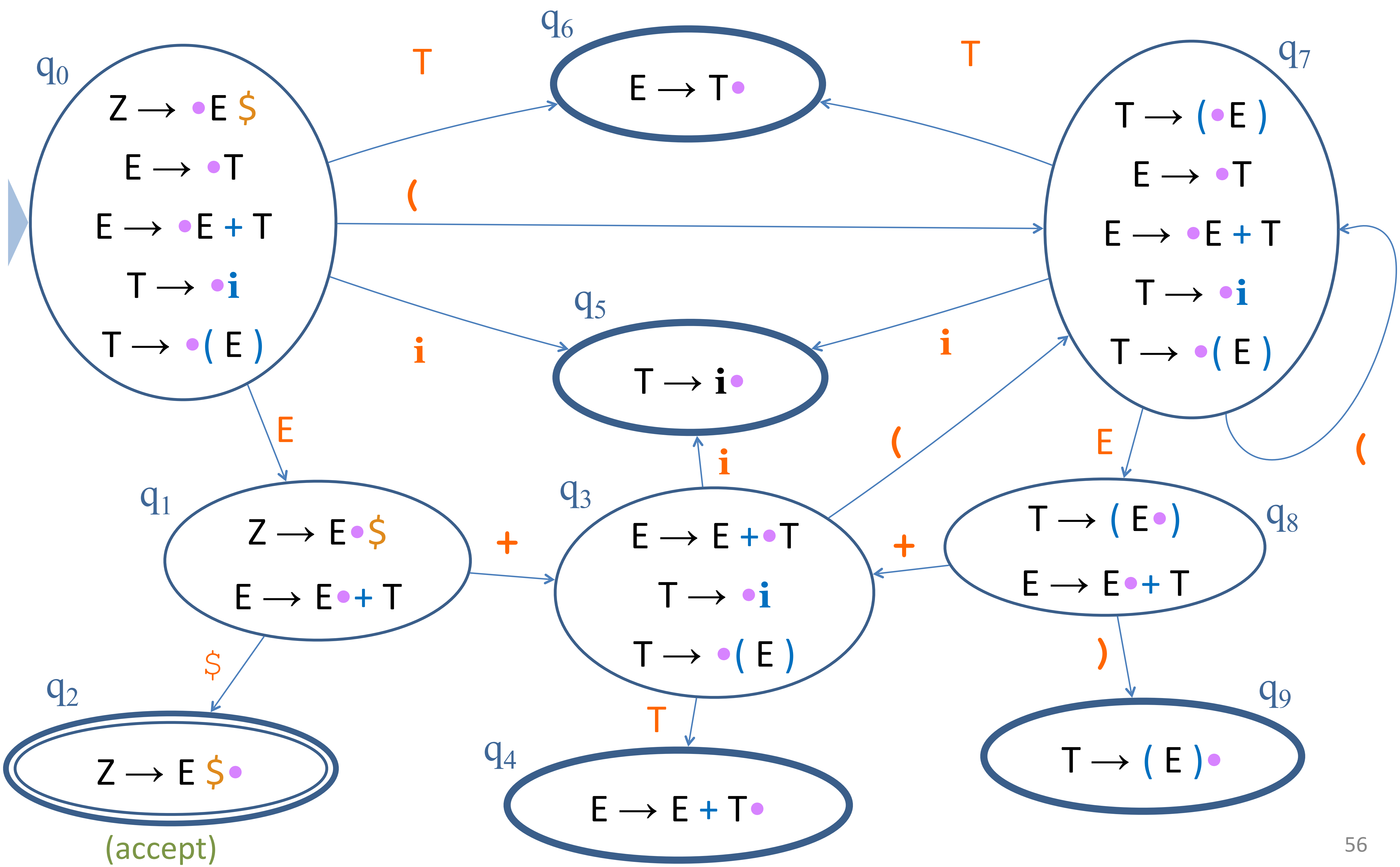
- Do we need to compute the state transitions during parsing?
  - Or can we do it ahead?



Computing State Transitions

- Do we need to compute the state transitions during parsing?
  - Or can we do it ahead of time?

LR(0) Automaton: All Possible Transitions



How to construct the LR(0) automaton?

(A bit more formal)

LR(0) Automaton Construction

- An LR(0) state consists of several LR(0) items
- For example, if we identified a string that is reduced to E, then we may be in one of the following LR items:

$$E \longrightarrow E \bullet + B \quad \text{or} \quad E \longrightarrow E \bullet * B$$

$$E \rightarrow E * B \mid E + B \mid B$$
$$B \rightarrow 0 \mid 1$$

- Therefore one state would be:

$$q = \{E \longrightarrow E \bullet + B\ ,\ E \longrightarrow E \bullet * B\}$$

- But if the current state includes $E \longrightarrow E + \bullet B$, then we must allow B to be derived too — **Closure**!

Construct the Closure

- Proposition: a closure set of LR(0) items has the following property — if the set contains an item of the form

  $$\boxed{A \longrightarrow \alpha \bullet B\ \beta}$$

  where B is a non terminal, then it must *also* contain an item

  $$\boxed{B \longrightarrow \bullet \delta}$$

  for *each rule* of the form $B \longrightarrow \delta$ in the grammar.

- Building the closure set for a given item set is a recursive fixpoint computation, as δ may also begin with a variable

Closure: an example

$$E \rightarrow E * B \mid E + B \mid B$$
$$B \rightarrow 0 \mid 1$$

grammar

$$C = \{\, E \rightarrow E + \bullet B \,\}$$

set C of LR items

- The closure of the set C is

$$\text{clos}(C) = \{\; E \rightarrow E + \bullet B,\; B \rightarrow \bullet 0,\; B \rightarrow \bullet 1 \;\}$$

- This will become another parser LR(0) state

Closure: an example

$$E \rightarrow E * B \mid E + B \mid B$$
$$B \rightarrow 0 \mid 1$$

grammar

$$C = \{ E \rightarrow \bullet E + B \}$$

set C of LR items

- The closure of the set C is

  clos(C) = {

Extended Grammar

- Goal: simple termination condition
  - Assume that the initial variable only appears in a single rule. This guarantees that the last reduction can be (easily) detected
  - Any grammar can be (easily) extended to have such a structure

Example: the grammar

(1) $E \longrightarrow E * B$
(2) $E \longrightarrow E + B$
(3) $E \longrightarrow B$
(4) $B \longrightarrow 0$
(5) $B \longrightarrow 1$

Can be extended into

Initial rule

(0) $S \longrightarrow E\ \$$

end-of-input marker

(1) $E \longrightarrow E * B$
(2) $E \longrightarrow E + B$
(3) $E \longrightarrow B$
(4) $B \longrightarrow 0$
(5) $B \longrightarrow 1$

The Initial State

- To build the automaton, we go through all possible states that can be seen during reduction
  - Each state represents a (closure) set of LR(0) items

- The initial state q_0 is the closure of the initial rule
  - In our example, the initial rule is $S \rightarrow \bullet E\ \$$, so:

$S \rightarrow E\ \$$
$E \rightarrow E * B \mid E + B \mid B$
$B \rightarrow 0 \mid 1$

$$q_0 = clos(\{S \rightarrow \bullet E\ \$\ \}) =$$

$$\boxed{\begin{array}{l} \{\ S \rightarrow \bullet E\ \$,\ \ E \rightarrow \bullet E * B\ ,\ E \rightarrow \bullet E + B\ , \\ \qquad\qquad E \rightarrow \bullet B\ ,\ \ B \rightarrow \bullet 0\ ,\ B \rightarrow \bullet 1\ \} \end{array}}$$

- Next, we build all possible states that can be reached by following a *single symbol* (token or variable)

The Next States

- For every symbol (terminal <u>or</u> variable) X, and every possible state (closure set) *q*,
  1. Find all items in the set of q in which the dot is before an X.
     (We denote this set by ***q*|X**)
  2. Move the dot ahead of the X in all items in *q*|X
  3. Find the closure of the obtained set:
     this is the state into which we move from *q* upon seeing X

$$\alpha \bullet X \bullet \beta$$

- Formally, the next set is defined on a set C and next symbol X:
  - $C|X = \{ (N \longrightarrow \alpha \bullet X\beta) \in C \}$
  - $\text{step}(C,X) = \{ N \longrightarrow \alpha X \bullet \beta \mid (N \longrightarrow \alpha \bullet X\beta) \in C\}$
  - $\text{nextSet}(C,X) = \text{clos}(\text{step}(C,X))$

The Next States

$S \longrightarrow E\ \$$

$E \longrightarrow E * B \mid E + B \mid B$

$B \longrightarrow 0 \mid 1$

Recall that in our example

$$q_0 = \mathrm{clos}(\{S \longrightarrow \bullet E\}) =$$

$$\{S \longrightarrow \bullet E\ \$\ ,\ E \longrightarrow \bullet E * B\ ,\ E \longrightarrow \bullet E + B\ ,$$
$$E \longrightarrow \bullet B\ ,\ B \longrightarrow \bullet 0\ ,\ B \longrightarrow \bullet 1\}$$

$\xrightarrow{0} q_1$

$\xrightarrow{1} q_2$

$\xrightarrow{E} q_3$

$\xrightarrow{B} q_4$

Let us check which states are reachable from it.

nextSet(q_0, $\cdot$) in the example

$S \longrightarrow E\ \$$

$E \longrightarrow E * B \mid E + B \mid B$

$B \longrightarrow 0 \mid 1$

$S \longrightarrow \bullet E\ \$$

$E \longrightarrow \bullet E * B$

$E \longrightarrow \bullet E + B$

$E \longrightarrow \bullet B$

$B \longrightarrow \bullet 0 \quad B \longrightarrow \bullet 1$

$q_0 \xrightarrow{0} q_1$

$q_0 \xrightarrow{1} q_2$

$q_0 \xrightarrow{E} q_3$

$q_0 \xrightarrow{B} q_4$

$q_0 | 0 = \{B \longrightarrow \bullet 0\}$

q_1

$q_1 = clos(\{B \longrightarrow 0 \bullet\}) =$

$\{B \longrightarrow 0 \bullet\}$

$q_0 | 1 = \{B \longrightarrow \bullet 1\}$

q_2

$q_2 = clos(\{B \longrightarrow 1 \bullet\}) =$

$\{B \longrightarrow 1 \bullet\}$

$q_0 | E = \{S \longrightarrow \bullet E\ \$\ , E \longrightarrow \bullet E * B\ , E \longrightarrow \bullet E + B\}$

q_3

$q_3 = clos\ (\{S \longrightarrow E \bullet \$, E \longrightarrow E \bullet * B\ , E \longrightarrow E \bullet + B\})$

$= \{S \longrightarrow E \bullet \$, E \longrightarrow E \bullet * B\ , E \longrightarrow E \bullet + B\}$

$q_0 | B = \{ E \longrightarrow \bullet B \}$

q_4

$q_4 = clos\ (\{ E \longrightarrow B \bullet\}) =$

$\{E \longrightarrow B \bullet\}$

From these new states there are more reachable states

$S \rightarrow E\ \$$

$E \rightarrow E * B \mid E + B \mid B$

$B \rightarrow 0 \mid 1$

- From q_1, q_2, q_4, there are no steps because the dot is at the end of every item in their sets.

- From state q_3 we can reach the following three states —

final state

$S \rightarrow E \bullet \$$

$E \rightarrow E \bullet * B$

$E \rightarrow E \bullet + B$

q_3

$q_3 \xrightarrow{*} q_5$, $q_3 \xrightarrow{+} q_6$, $q_3 \xrightarrow{\$} \text{accept}$

<u>q_5</u>

$q_3 | * = \{ E \rightarrow E \bullet * B \}$

$q_5 = \text{clos}(\{ E \rightarrow E * \bullet B \}) = \{ E \rightarrow E * \bullet B,\ B \rightarrow \bullet 0,\ B \rightarrow \bullet 1 \}$

<u>q_6</u>

$q_3 | + = \{ E \rightarrow E \bullet + B \}$

$q_6 = \text{clos}(\{ E \rightarrow E + \bullet B \}) = \{ E \rightarrow E + \bullet B,\ B \rightarrow \bullet 0,\ B \rightarrow \bullet 1 \}$

<u>accept</u>

$q_3 | \$ = \{ S \rightarrow E \bullet \$ \}$

$q_{accept} = \text{clos}(\{ S \rightarrow E\ \$ \bullet \}) = \{ S \rightarrow E\ \$ \bullet \}$

Finally

$S \to E\ \$$
$E \to E * B \mid E + B \mid B$
$B \to 0 \mid 1$

- From q_5 we can proceed with X=0, or X=1, or X=B.
- For X=0 we reach q_1 again; for X=1 we reach q_2.
- For X=B we get q_7:

$E \to E * \bullet B$
$B \to \bullet 0$
$B \to \bullet 1$

$q_5 \xrightarrow{B}$

$\underline{q_7}$
$\mathrm{clos}(\{ E \to E * B\bullet \}) = \{ E \to E * B\bullet \}$

- Similarly, from q_6 with X=B we get q_8:

$E \to E + \bullet B$
$B \to \bullet 0$
$B \to \bullet 1$

$q_6 \xrightarrow{B}$

$\underline{q_8}$
$\mathrm{clos}(\{ E \to E + B\bullet \}) = \{ E \to E + B\bullet \}$

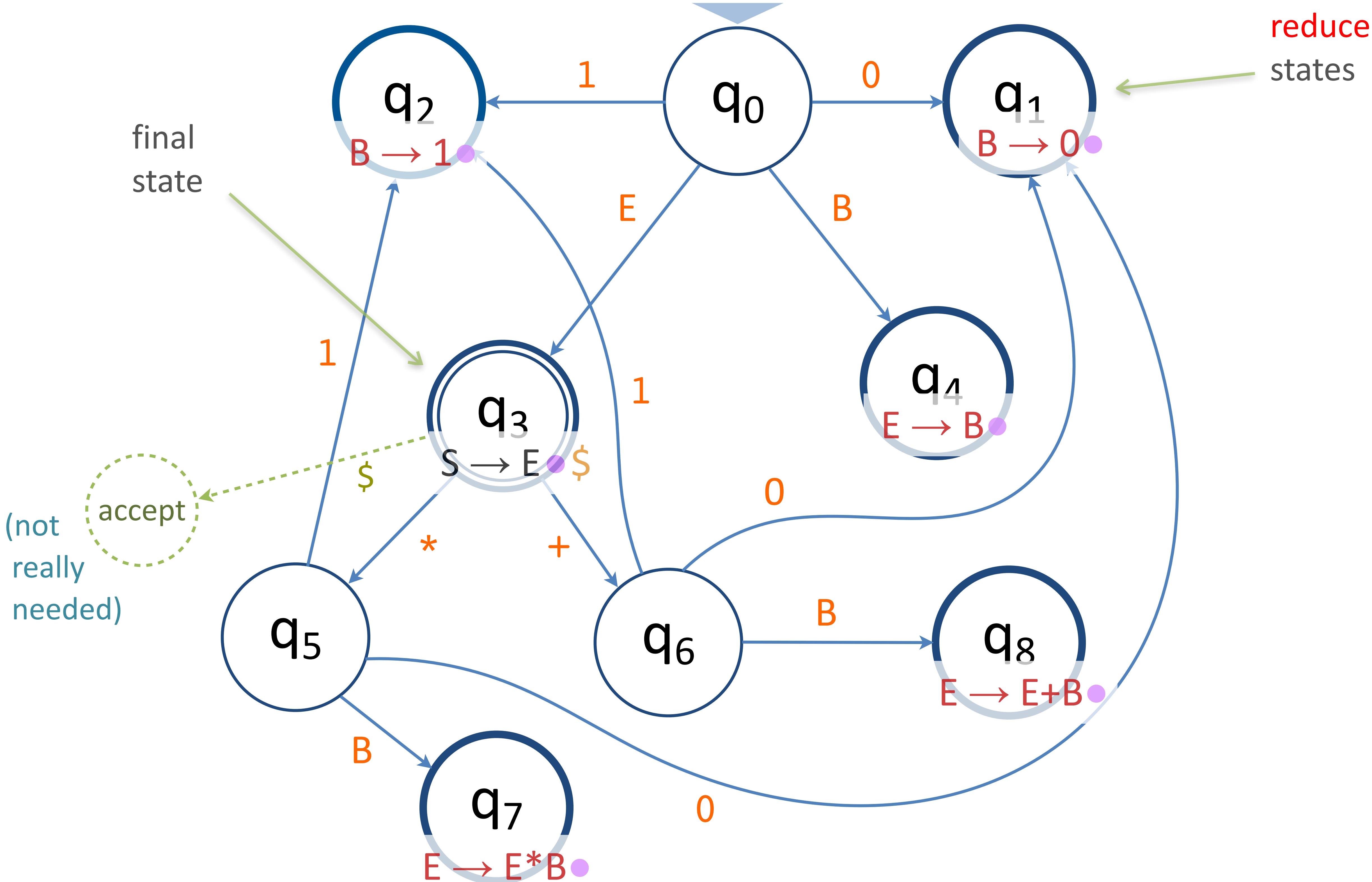
- These two states have no further steps. (Why?)

Automaton

$S \longrightarrow E\ \$$

$E \longrightarrow E * B \mid E + B \mid B$

$B \longrightarrow 0 \mid 1$



Another LR(0) Automaton

Z → E $
E → T | E + T
T → i | (E)