

Top-Down Parsing

Announcements

- Office hours schedule posted on Piazza.
 - Keith:
 - Monday/Tuesday, 2PM – 4PM in Gates 178.
 - Jinchao:
 - Wednesday/Thursday, 6PM – 8PM in Gates B26.
- Feel free to email us with questions!
- Sign up for Piazza (www.piazza.com).

Review from Last Time

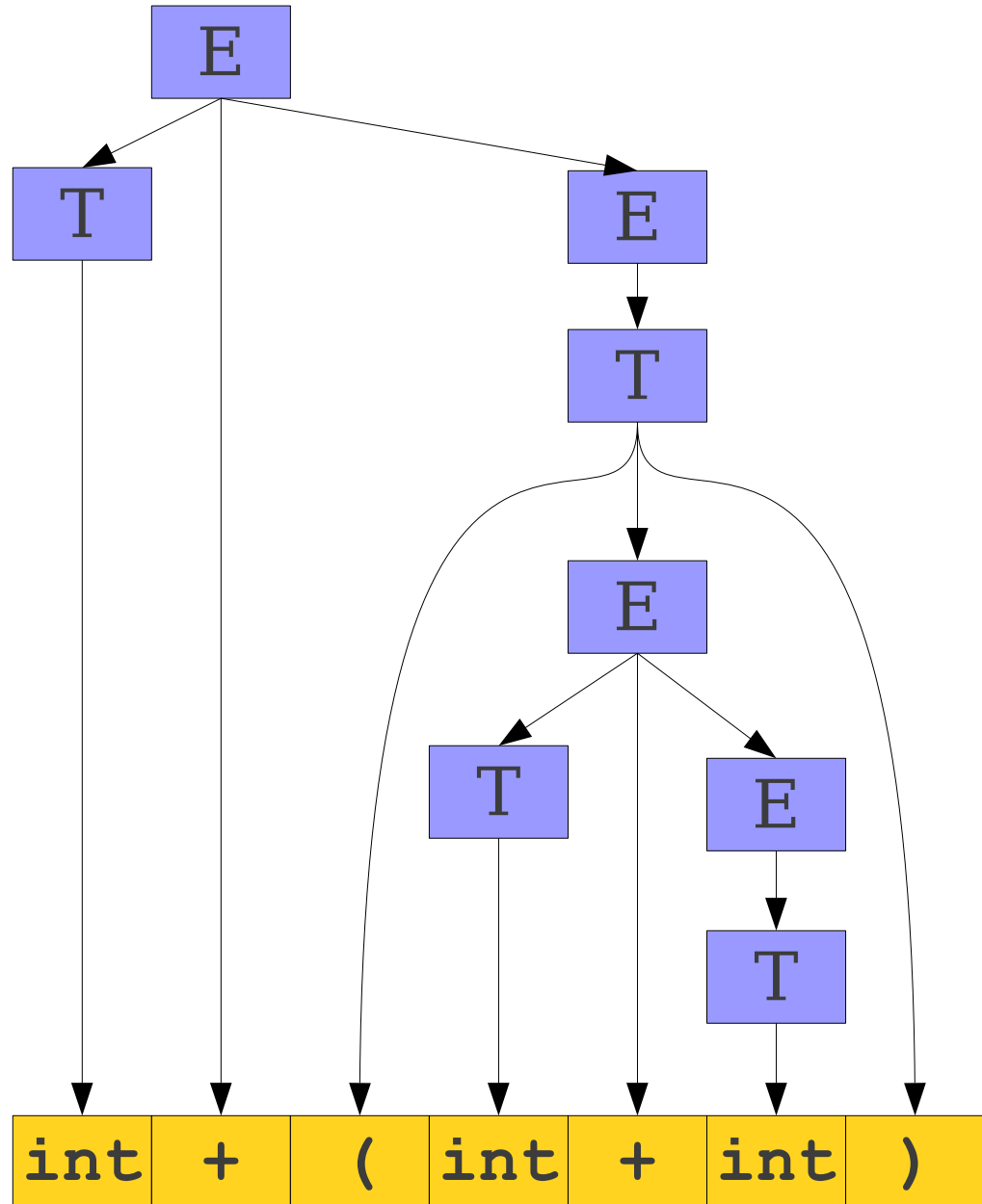
- Goal of syntax analysis: recover the intended structure of the program.
- Idea: Use a **context-free grammar** to describe the programming language.
- Given a sequence of tokens, look for a **parse tree** that generates those tokens.
- Recovering this syntax tree is called **parsing** and is the topic of this week (and part of next!)

Different Types of Parsing

- **Top-Down Parsing** (Today / Friday)
 - Beginning with the start symbol, try to guess the productions to apply to end up at the user's program.
- **Bottom-Up Parsing** (Friday / Monday)
 - Beginning with the user's program, try to apply productions in reverse to convert the program back into the start symbol.

Top-Down Parsing

E \rightarrow **T**
E \rightarrow **T** + **E**
T \rightarrow int
T \rightarrow (**E**)



Challenges in Top-Down Parsing

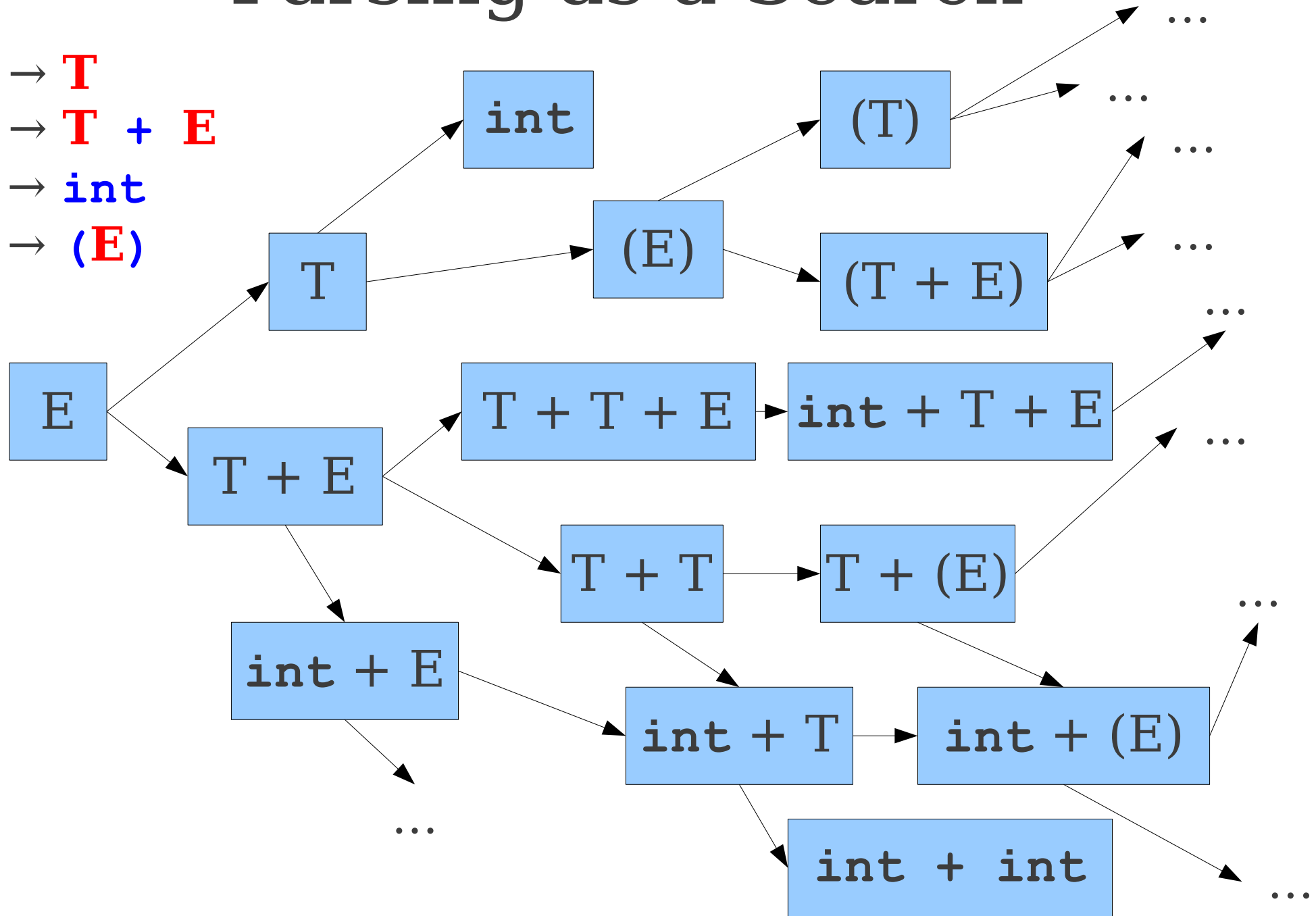
- Top-down parsing begins with virtually no information.
 - Begins with just the start symbol, which matches *every* program.
- How can we know which productions to apply?
- In general, we can't.
 - There are some grammars for which the best we can do is guess and backtrack if we're wrong.
 - If we have to guess, how do we do it?

Parsing as a Search

- An idea: **treat parsing as a graph search**.
- Each node is a **sentential form** (a string of terminals and nonterminals derivable from the start symbol).
- There is an edge from node α to node β iff $\alpha \Rightarrow \beta$.

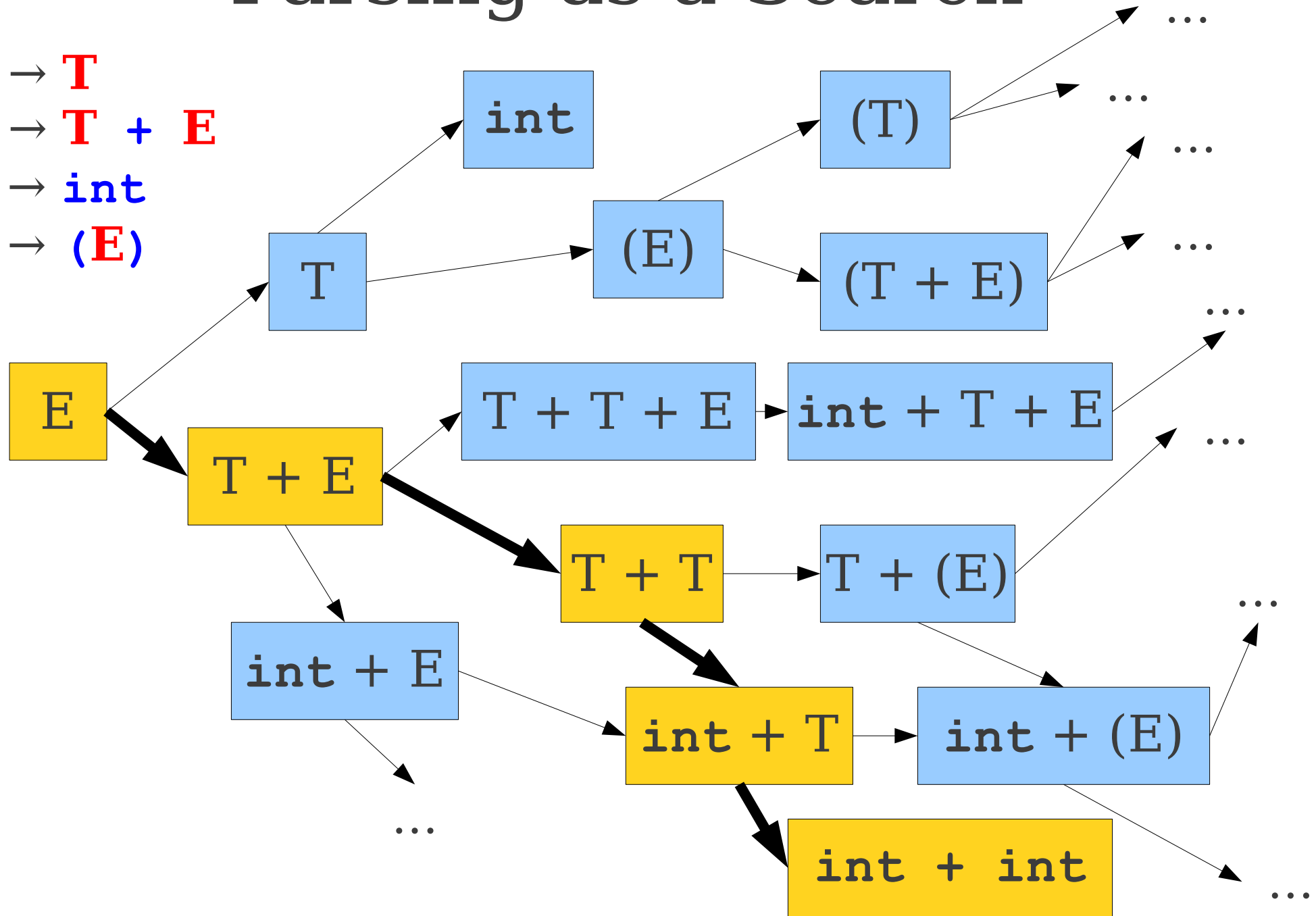
Parsing as a Search

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$



Parsing as a Search

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$



Our First Top-Down Algorithm

- **Breadth-First Search**
- Maintain a worklist of sentential forms, initially just the start symbol **S**.
- While the worklist isn't empty:
 - Remove an element from the worklist.
 - If it matches the target string, you're done.
 - Otherwise, for each possible string that can be derived in one step, add that string to the worklist.
- Can recover a parse tree by tracking what productions we applied at each step.

Breadth-First Search Parsing

Worklist

E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

`int + int`

Breadth-First Search Parsing



E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

`int + int`

Breadth-First Search Parsing

Worklist

E

E → **T**

E → **T** + **E**

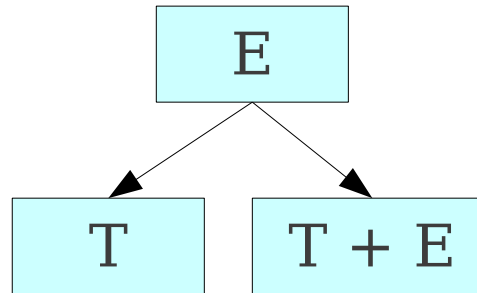
T → **int**

T → (**E**)

`int + int`

Breadth-First Search Parsing

Worklist



E → **T**

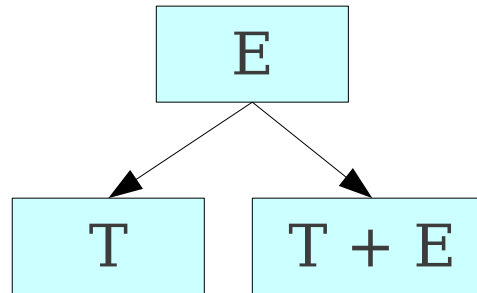
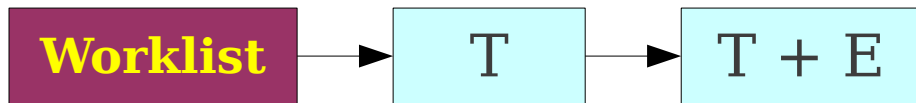
E → **T** + **E**

T → **int**

T → (**E**)

`int + int`

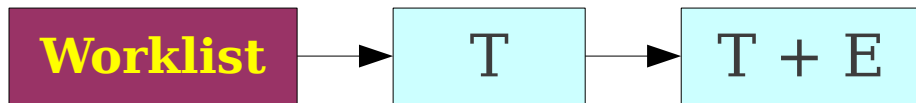
Breadth-First Search Parsing



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

`int + int`

Breadth-First Search Parsing



E → **T**

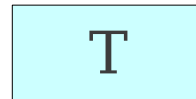
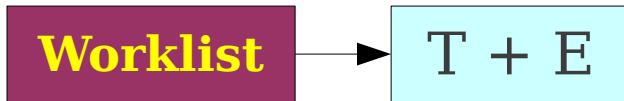
E → **T** + **E**

T → **int**

T → (**E**)

`int + int`

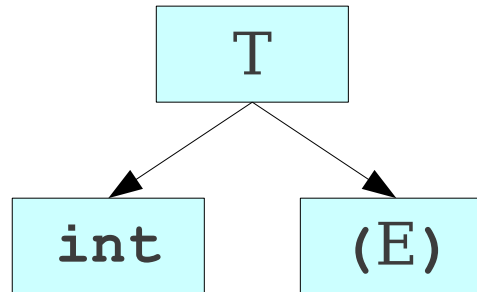
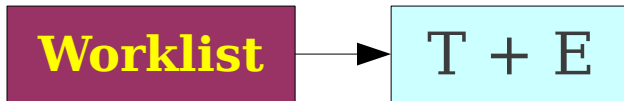
Breadth-First Search Parsing



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

`int + int`

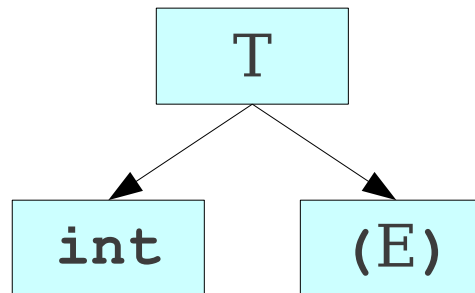
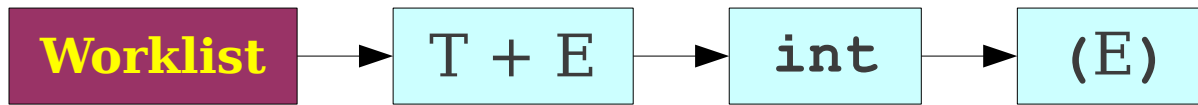
Breadth-First Search Parsing



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

int + int

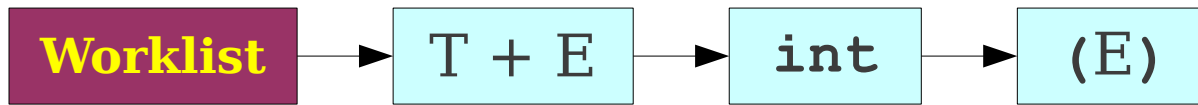
Breadth-First Search Parsing



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

`int + int`

Breadth-First Search Parsing



E → **T**

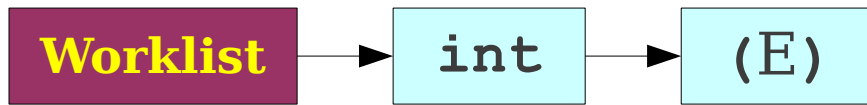
E → **T** + **E**

T → **int**

T → **(E)**

`int + int`

Breadth-First Search Parsing

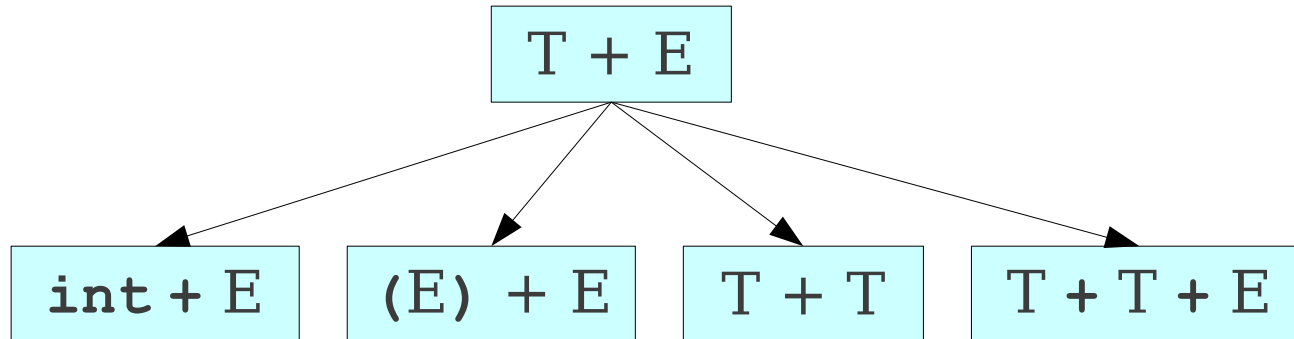
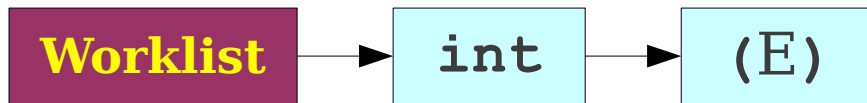


T + E

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

int + int

Breadth-First Search Parsing



E → **T**

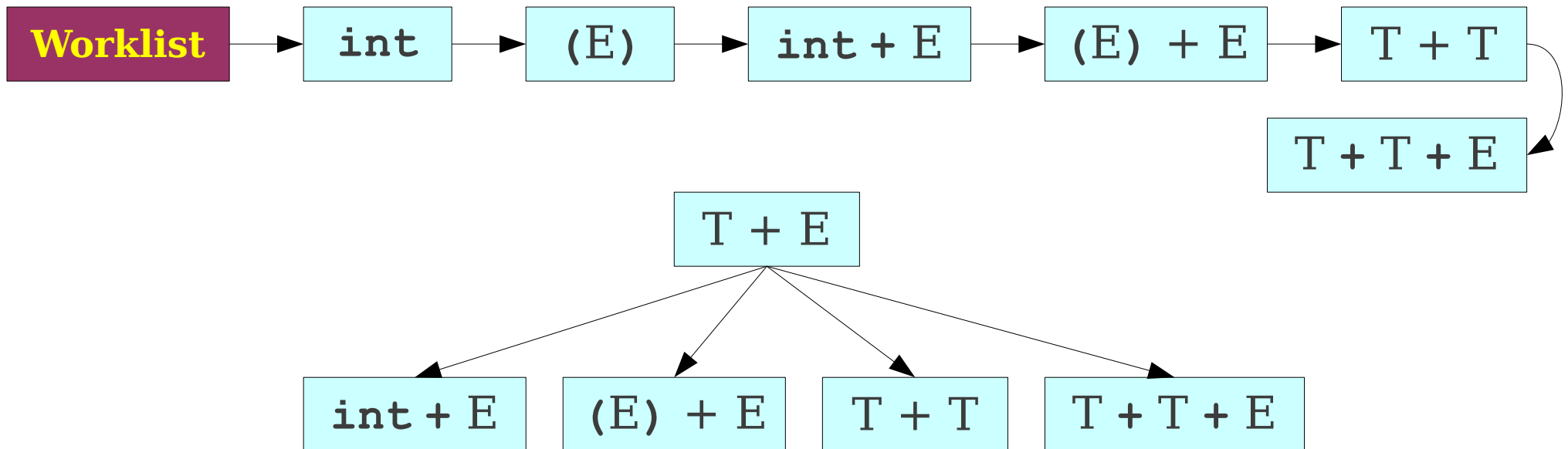
E → **T** + **E**

T → **int**

T → **(E)**

int + int

Breadth-First Search Parsing



E → **T**

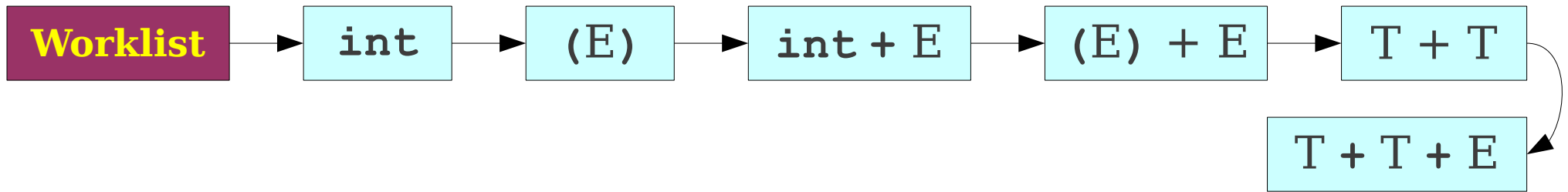
E → **T** + **E**

T → **int**

T → **(E)**

int + int

Breadth-First Search Parsing



E → **T**

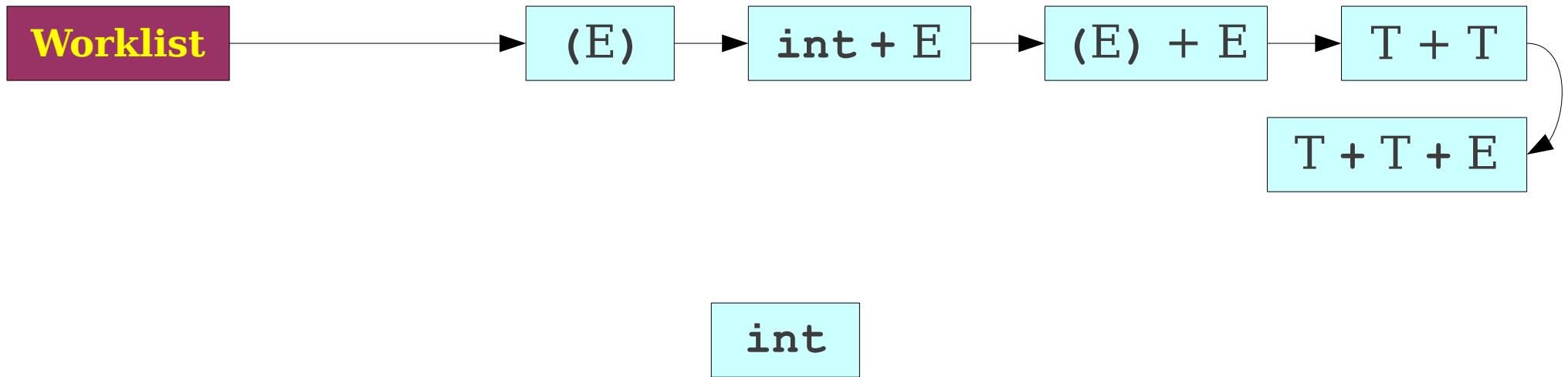
E → **T** + **E**

T → **int**

T → **(E)**

`int + int`

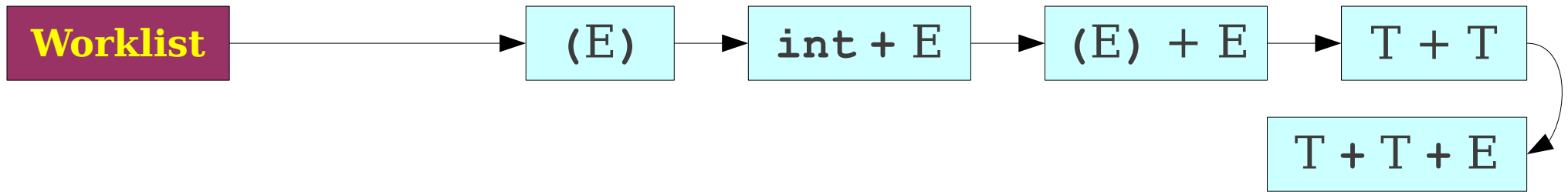
Breadth-First Search Parsing



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

int + int

Breadth-First Search Parsing



E → **T**

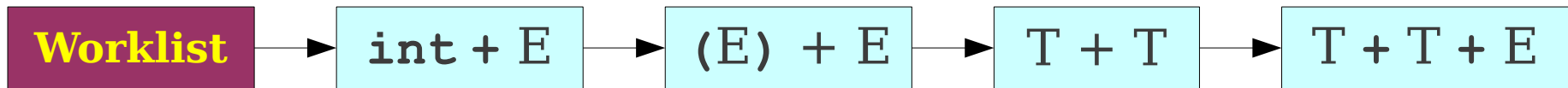
E → **T** + **E**

T → **int**

T → **(E)**

`int + int`

Breadth-First Search Parsing



(E)

E → **T**

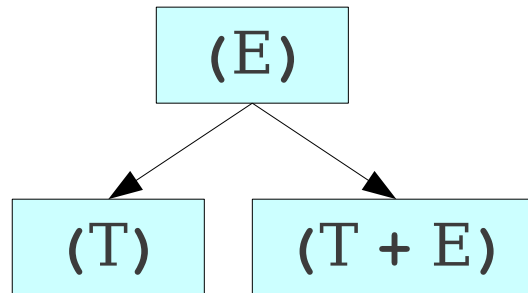
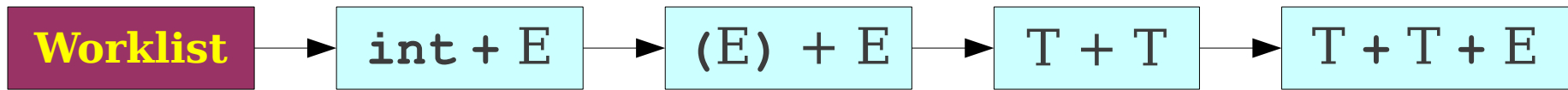
E → **T** + **E**

T → **int**

T → **(E)**

int + int

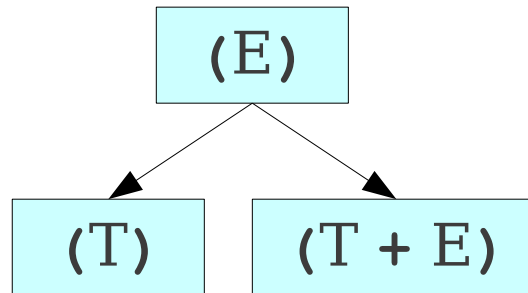
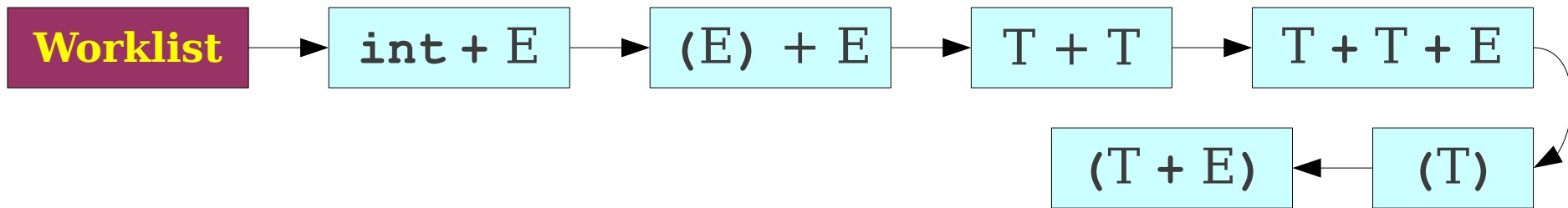
Breadth-First Search Parsing



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

int + int

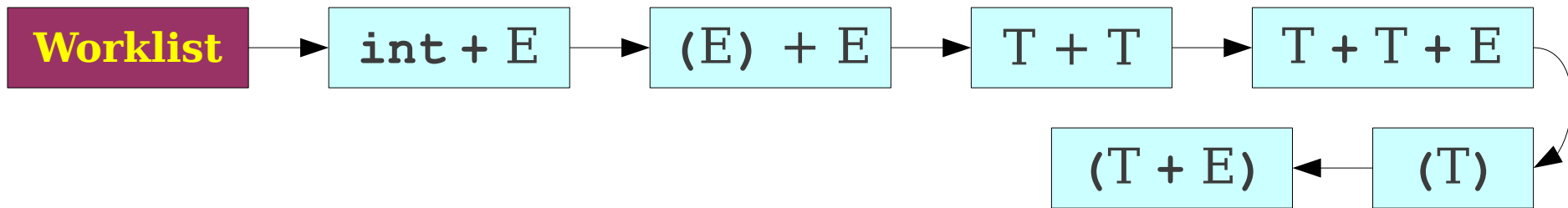
Breadth-First Search Parsing



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

`int + int`

Breadth-First Search Parsing



E → **T**

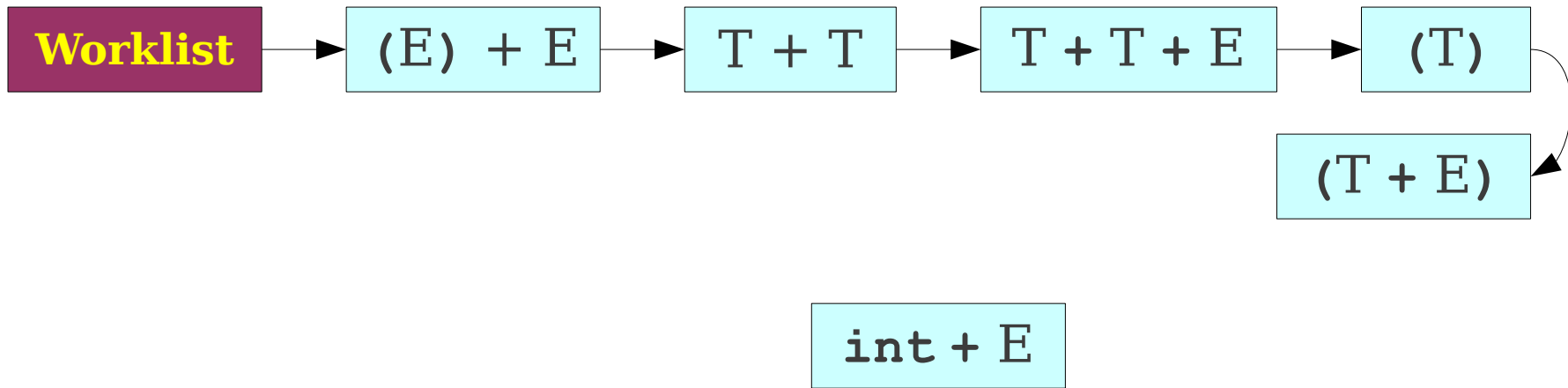
E → **T** + **E**

T → **int**

T → **(E)**

`int + int`

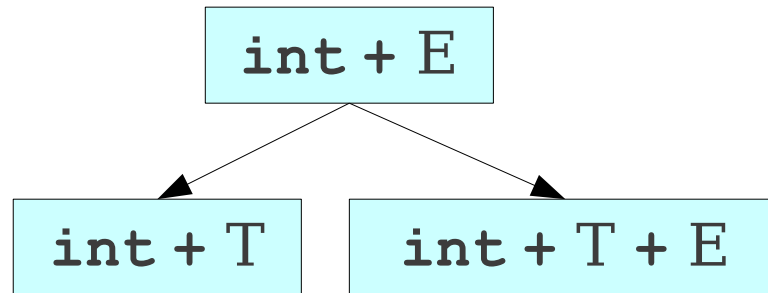
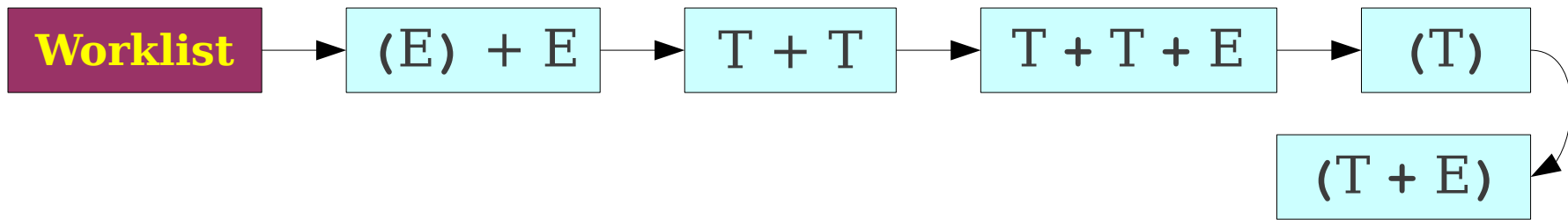
Breadth-First Search Parsing



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow int$
 $T \rightarrow (E)$

`int + int`

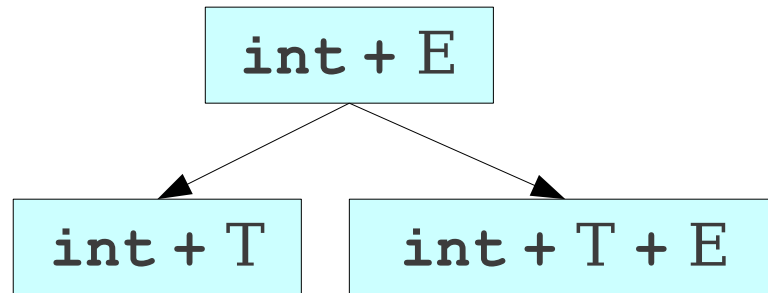
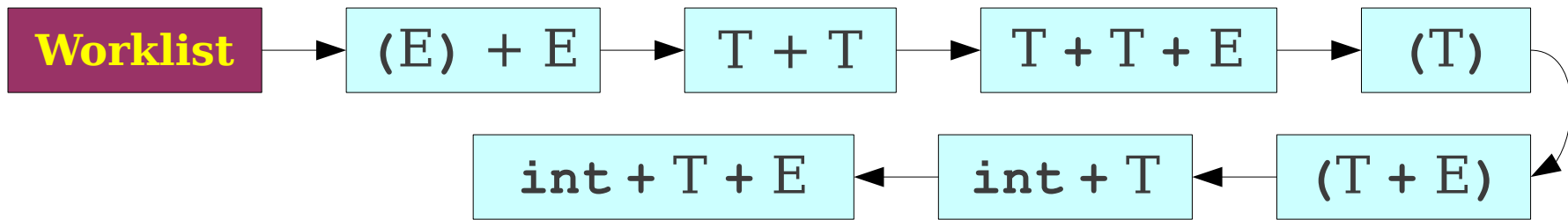
Breadth-First Search Parsing



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

`int + int`

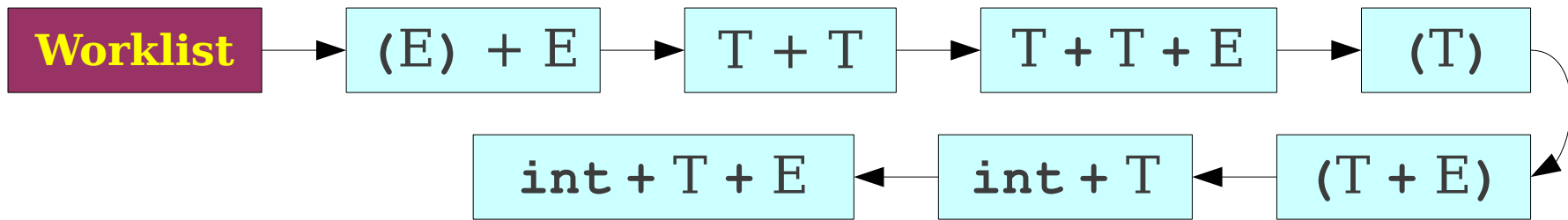
Breadth-First Search Parsing



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

`int + int`

Breadth-First Search Parsing



E → **T**

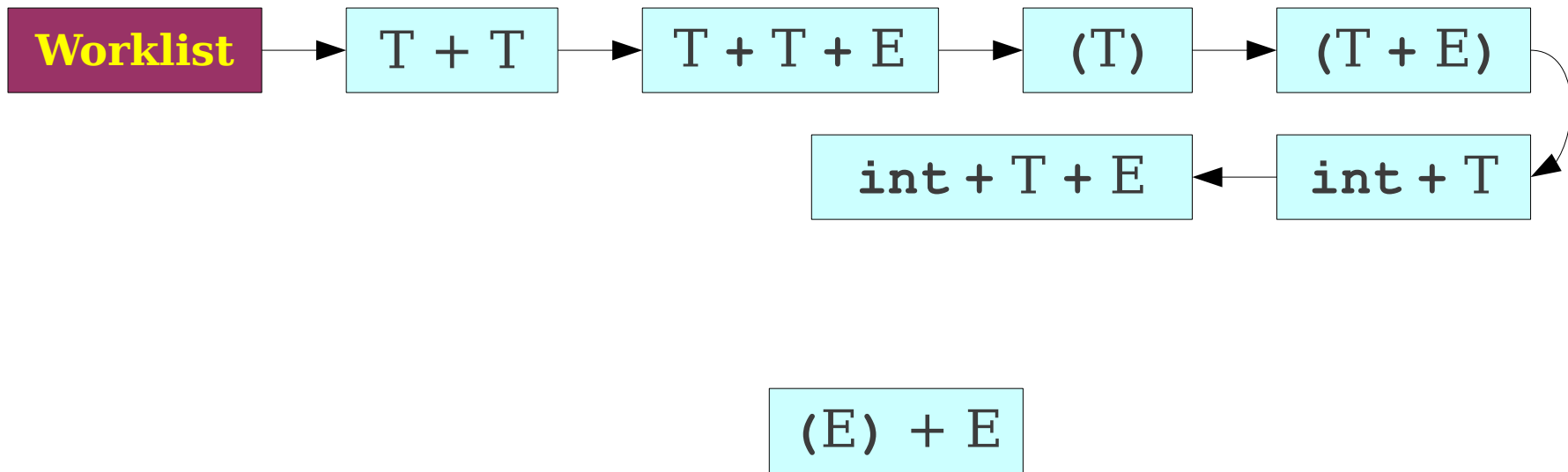
E → **T** + **E**

T → **int**

T → **(E)**

int + int

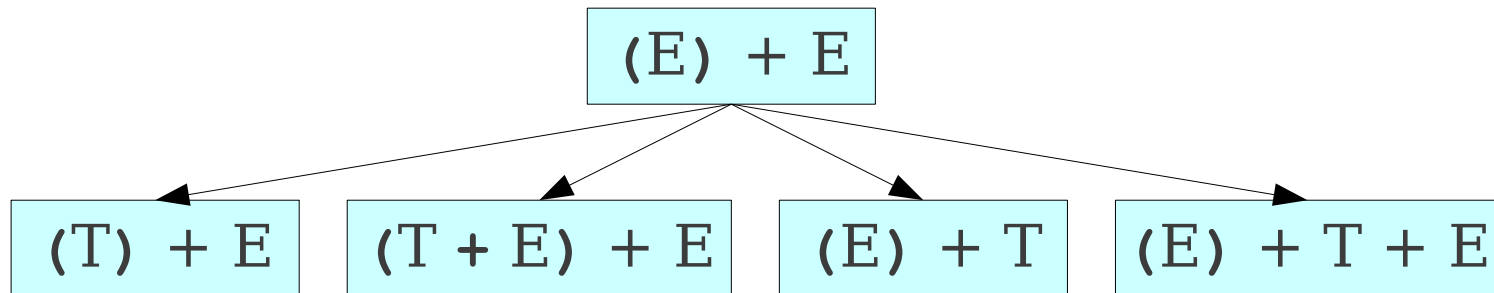
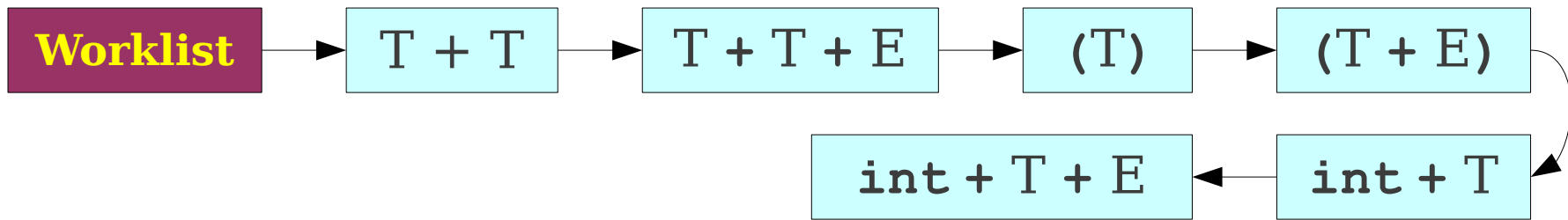
Breadth-First Search Parsing



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow int$
 $T \rightarrow (E)$

$int + int$

Breadth-First Search Parsing



E → **T**

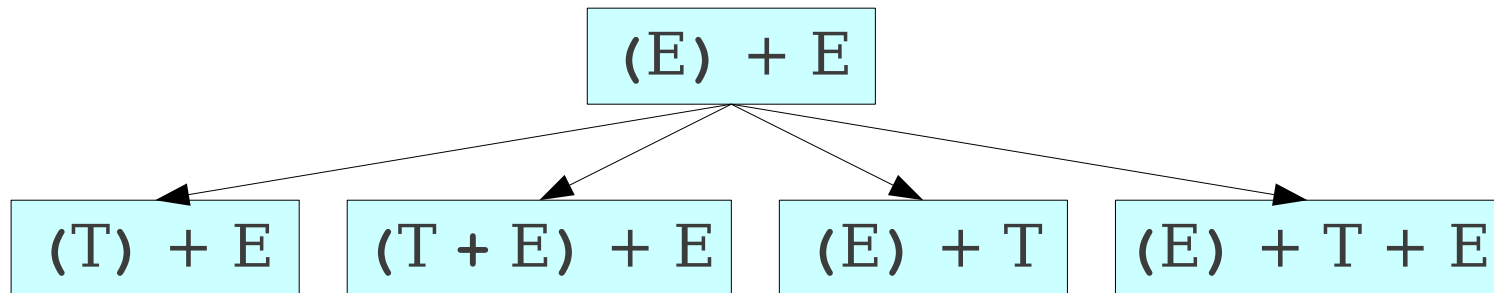
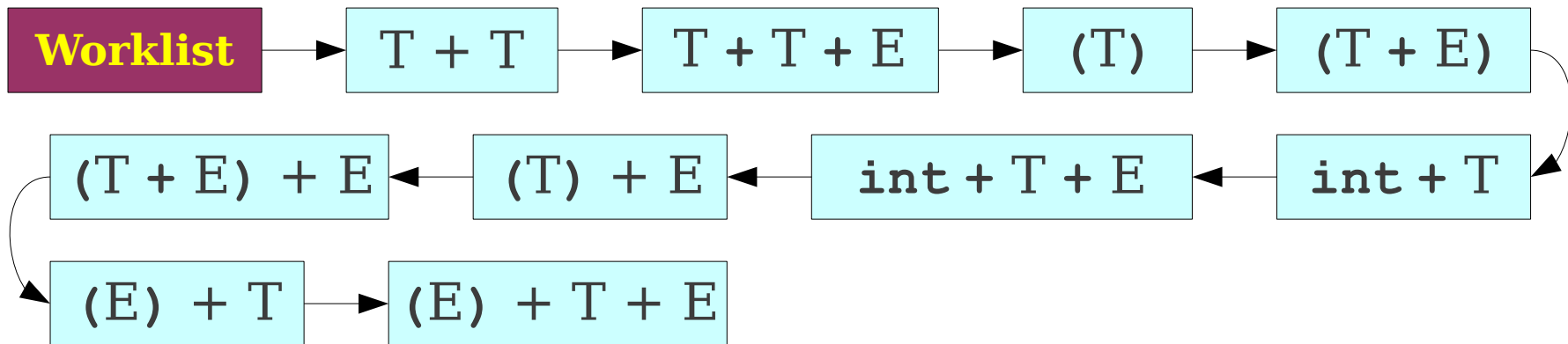
E → **T** + **E**

T → **int**

T → **(E)**

int + int

Breadth-First Search Parsing



$E \rightarrow T$

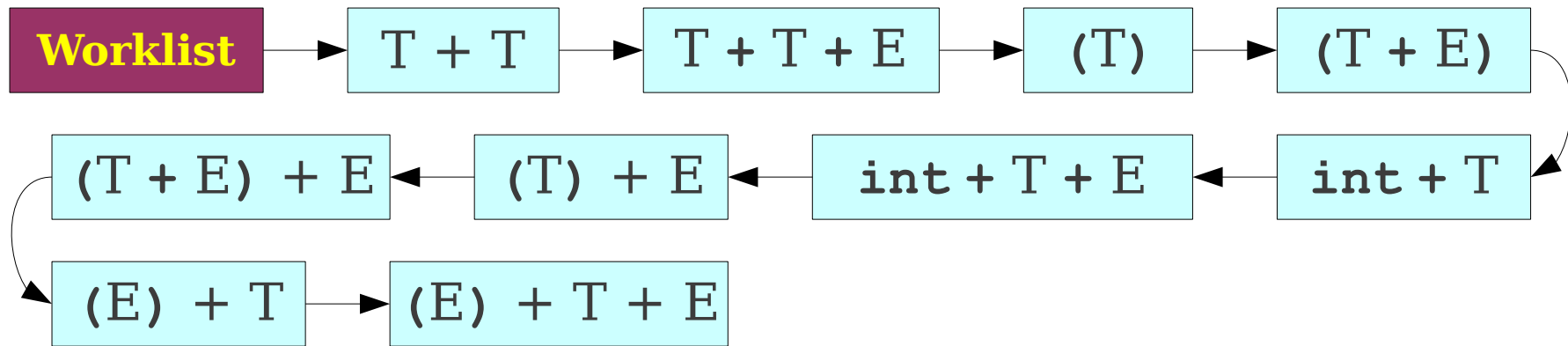
$E \rightarrow T + E$

$T \rightarrow \text{int}$

$T \rightarrow (E)$

$\text{int} + \text{int}$

Breadth-First Search Parsing



E → **T**

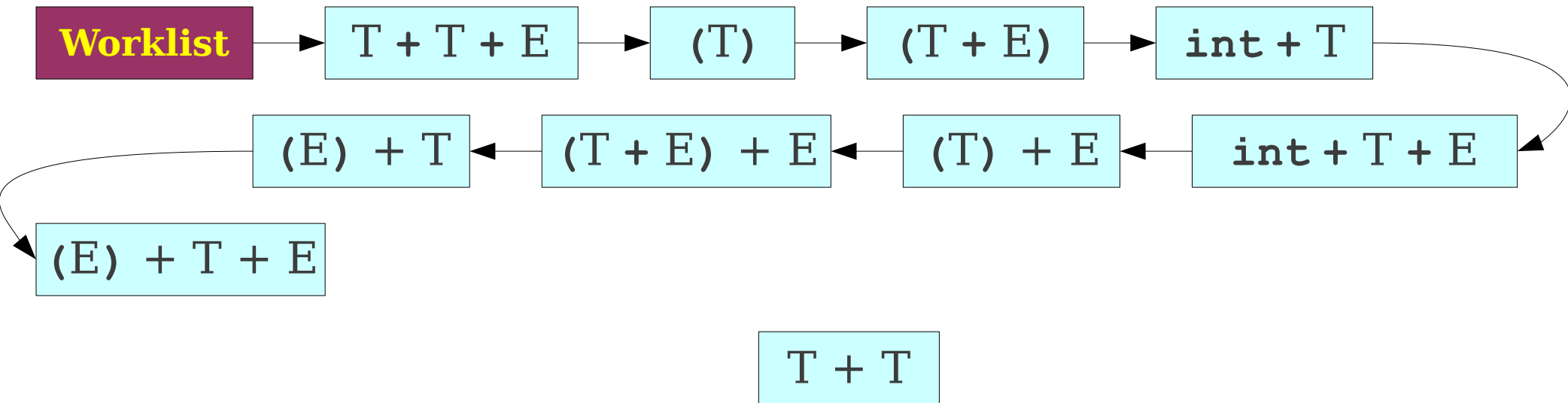
E \rightarrow **T** + **E**

T → **int**

T \rightarrow **(E)**

int + int

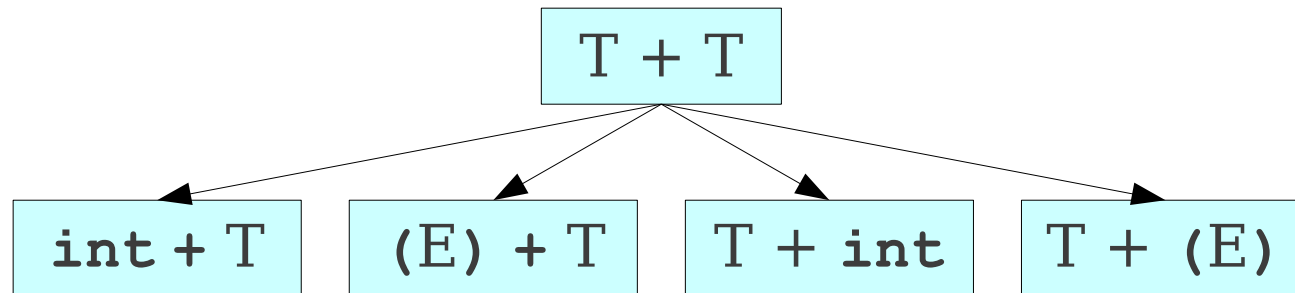
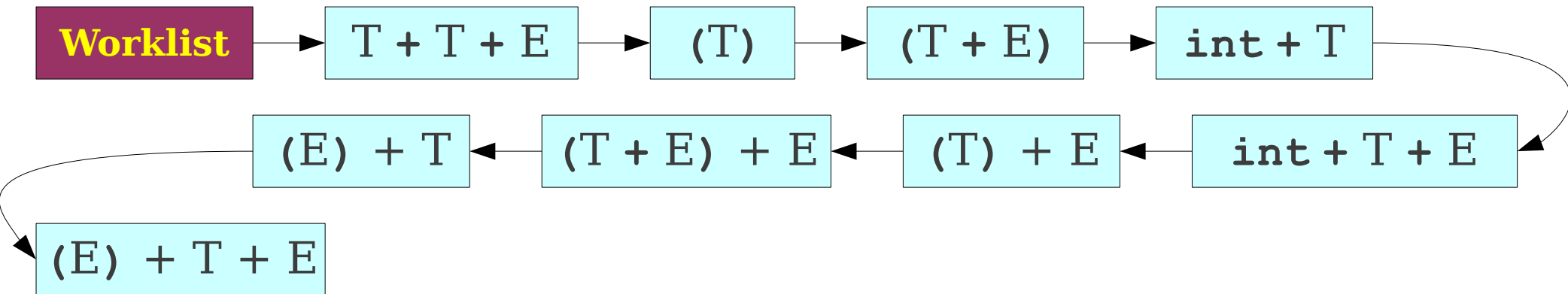
Breadth-First Search Parsing



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow int$
 $T \rightarrow (E)$

$int + int$

Breadth-First Search Parsing



E → **T**

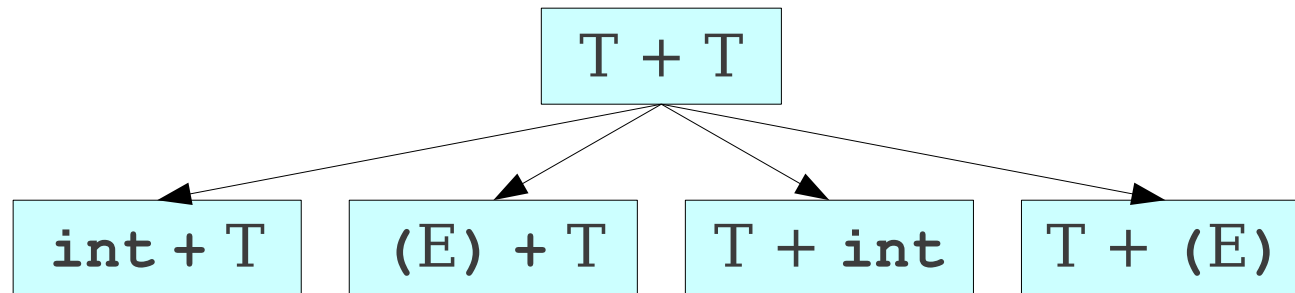
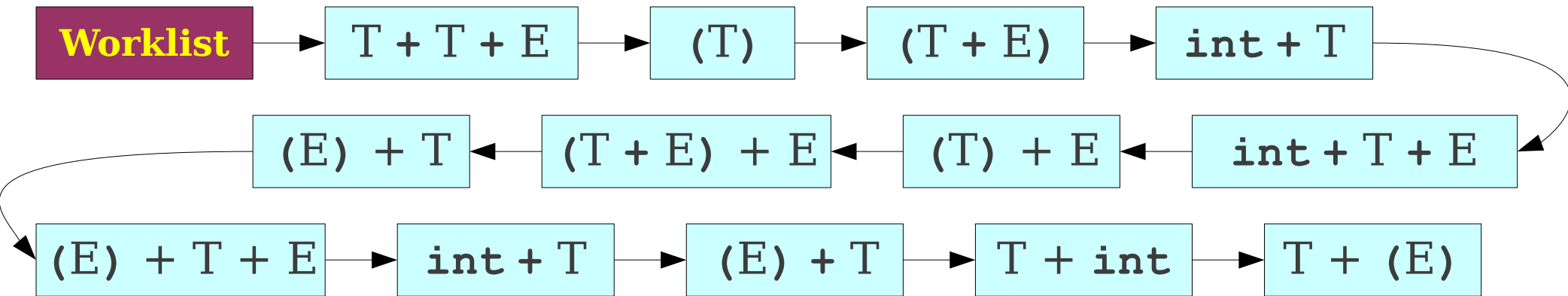
E → **T** + **E**

T → **int**

T → **(E)**

int + int

Breadth-First Search Parsing



$E \rightarrow T$

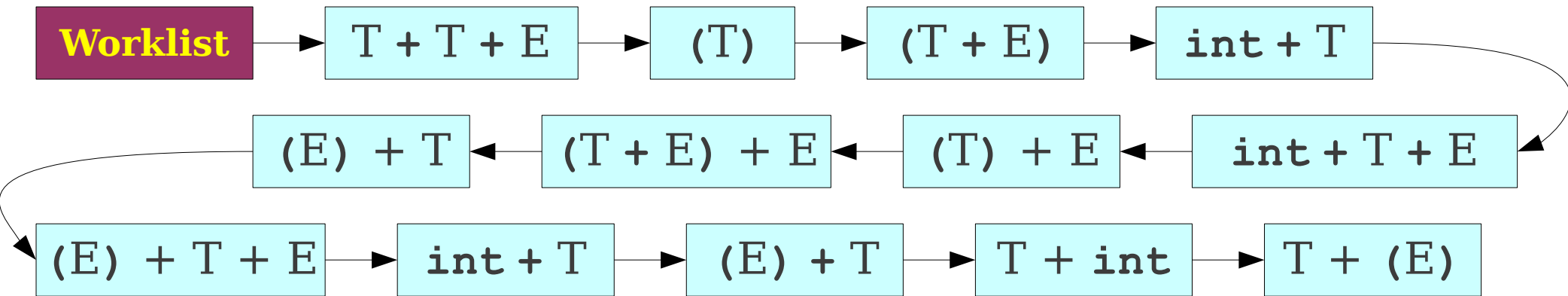
$E \rightarrow T + E$

$T \rightarrow int$

$T \rightarrow (E)$

$int + int$

Breadth-First Search Parsing



E → **T**

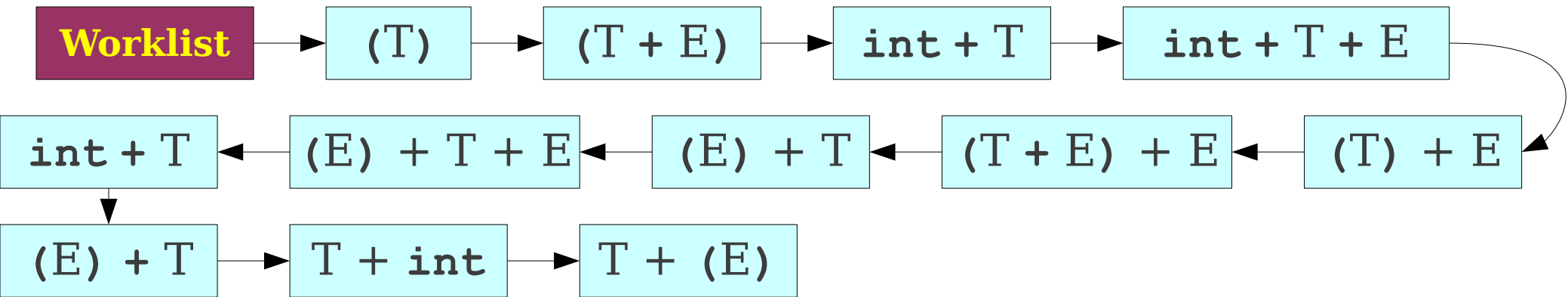
E → **T** + **E**

T → **int**

T → **(E)**

int + int

Breadth-First Search Parsing

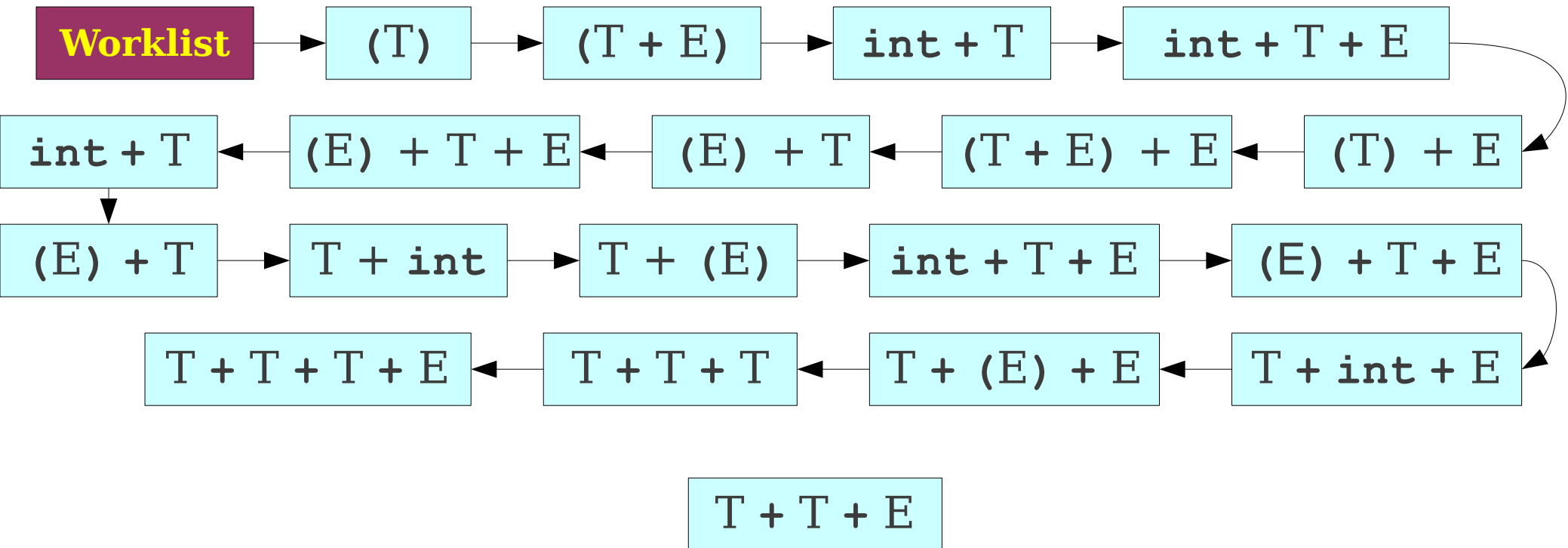


T + T + E

int + int

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

Breadth-First Search Parsing



E → **T**

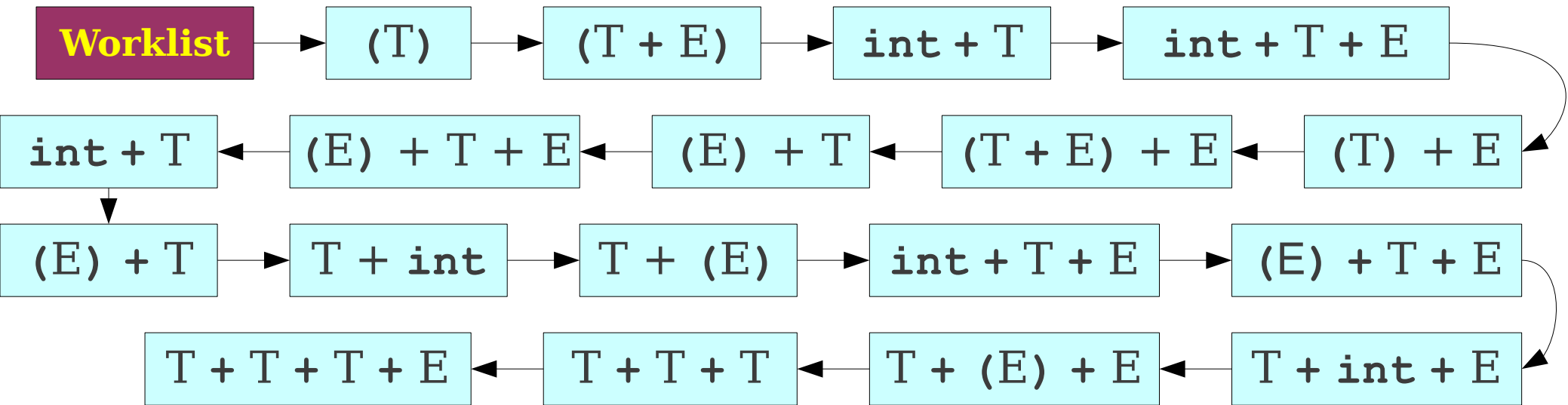
E → **T** + **E**

T → **int**

T → **(E)**

int + int

Breadth-First Search Parsing



E → **T**

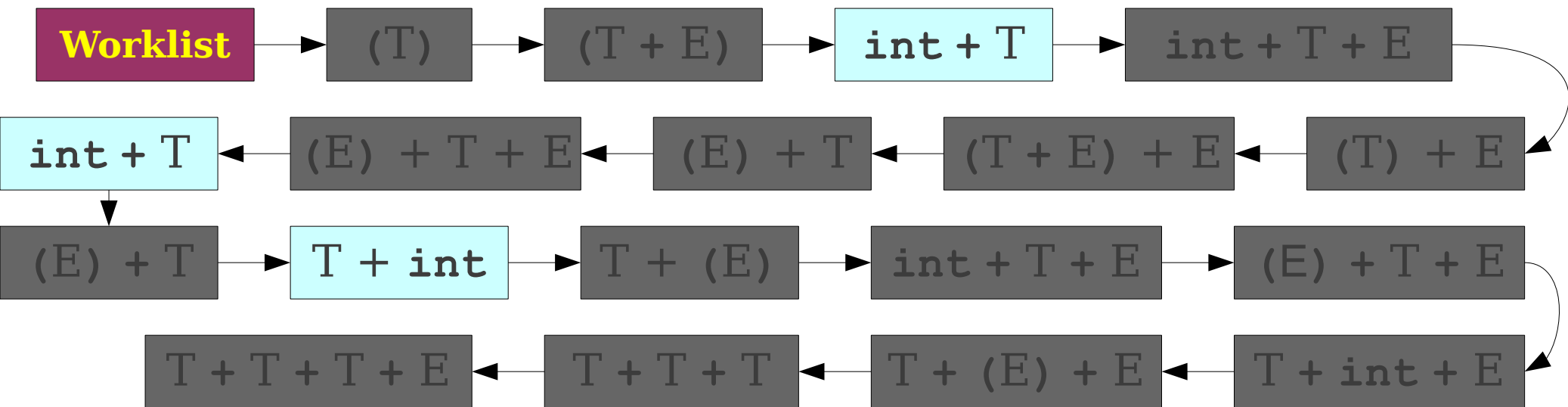
E → **T** + **E**

T → **int**

T → **(E)**

int + int

Breadth-First Search Parsing



E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

int + int

BFS is Slow

- Enormous time and memory usage:
 - Lots of **wasted effort**:
 - Generates a lot of sentential forms that couldn't possibly match.
 - But in general, extremely hard to tell whether a sentential form can match – that's the job of parsing!
 - High **branching factor**:
 - Each sentential form can expand in (potentially) many ways for each nonterminal it contains.

Reducing Wasted Effort

- Suppose we're trying to match a string y .
- Suppose we have a sentential form $\tau = \alpha\omega$, where α is a string of terminals and ω is a string of terminals and nonterminals.
- If α isn't a prefix of y , then no string derived from τ can ever match y .
- If we can find a way to try to get a prefix of terminals at the front of our sentential forms, then we can start pruning out impossible options.

Reducing the Branching Factor

- If a string has many nonterminals in it, the branching factor can be high.
 - Sum of the number of productions of each nonterminal involved.
- If we can restrict which productions we apply, we can keep the branching factor lower.

Leftmost Derivations

- Recall: A **leftmost derivation** is one where we always expand the leftmost symbol first.
- Updated algorithm:
 - Do a breadth-first search, **only considering leftmost derivations**.
 - Dramatically drops branching factor.
 - Increases likelihood that we get a prefix of nonterminals.
 - Prune sentential forms that can't possibly match.
 - Avoids wasted effort.

Leftmost BFS



E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

`int + int`

Leftmost BFS

Worklist

E

E → **T**

E → **T** + **E**

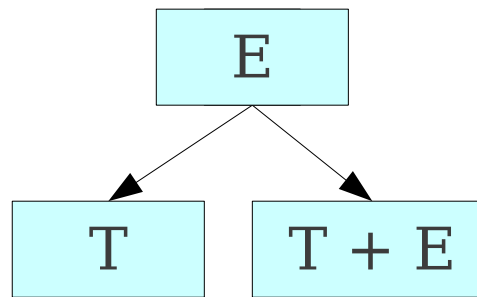
T → **int**

T → (**E**)

`int + int`

Leftmost BFS

Worklist



E → **T**

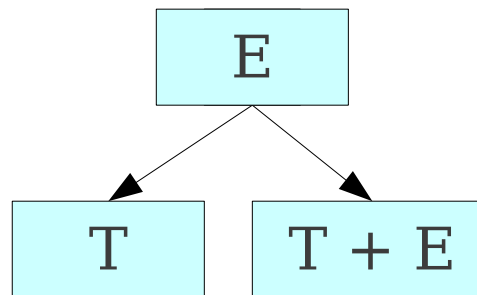
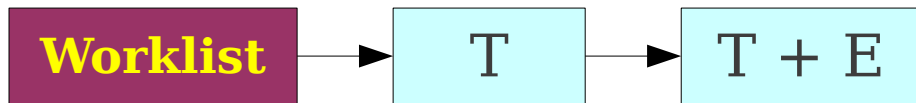
E → **T** + **E**

T → **int**

T → (**E**)

`int + int`

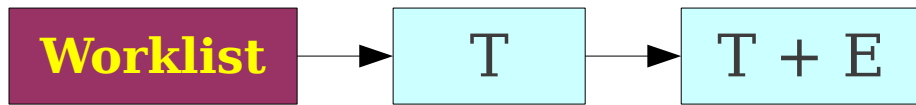
Leftmost BFS



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

`int + int`

Leftmost BFS



E → **T**

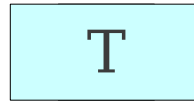
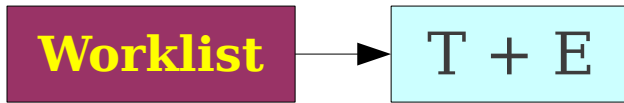
E → **T** + **E**

T → **int**

T → (**E**)

`int + int`

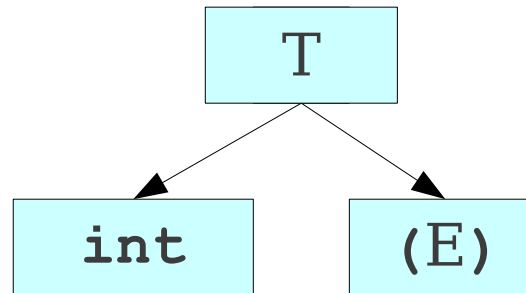
Leftmost BFS



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

`int + int`

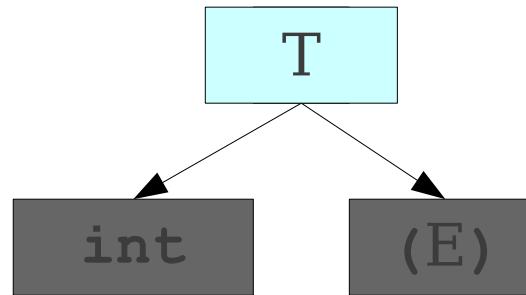
Leftmost BFS



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

`int + int`

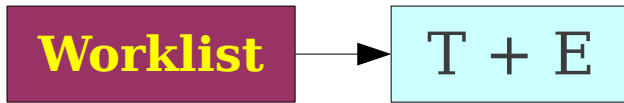
Leftmost BFS



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

`int + int`

Leftmost BFS



E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

`int + int`

Leftmost BFS

Worklist

T + E

E → **T**

E → **T** + **E**

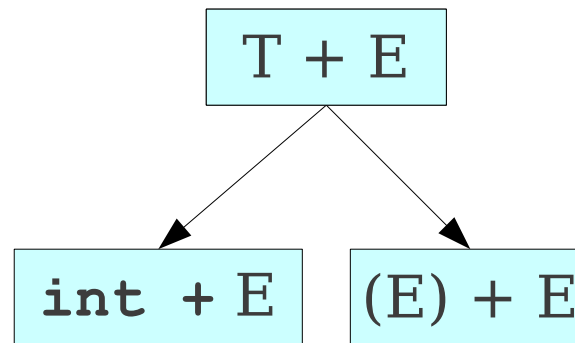
T → **int**

T → (**E**)

`int + int`

Leftmost BFS

Worklist



E → **T**

E → **T** + **E**

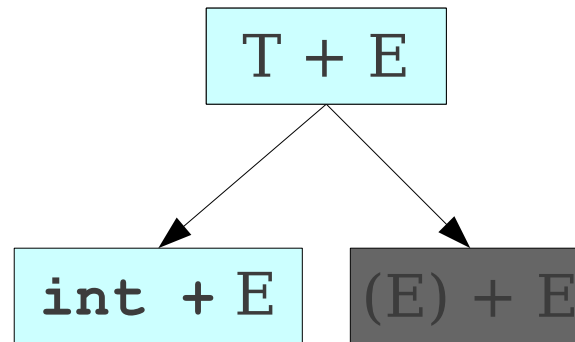
T → **int**

T → **(E)**

int + int

Leftmost BFS

Worklist



E → **T**

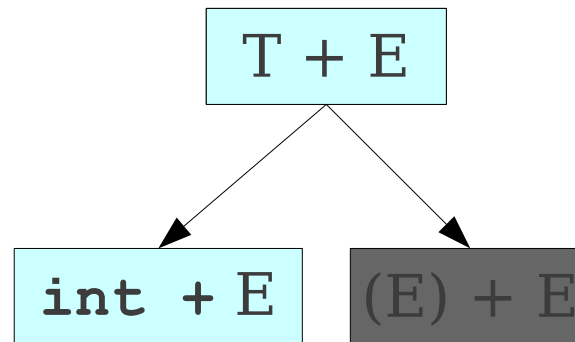
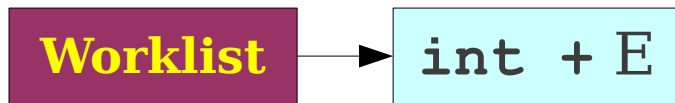
E → **T** + **E**

T → **int**

T → **(E)**

int + int

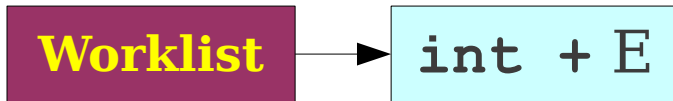
Leftmost BFS



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

`int + int`

Leftmost BFS



E → **T**

E → **T** + **E**

T → **int**

T → **(E)**

`int + int`

Leftmost BFS

Worklist

`int + E`

E → **T**

E → **T** + **E**

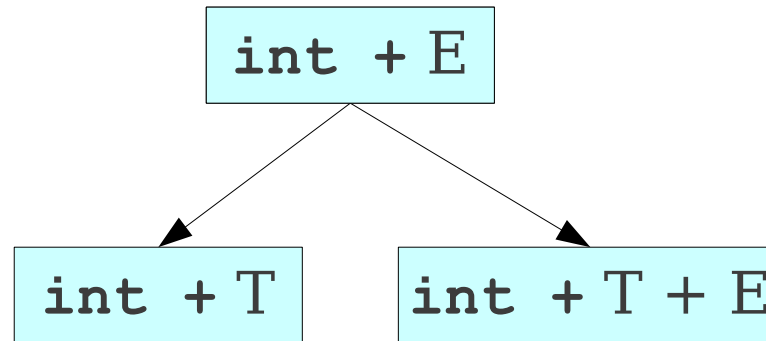
T → **int**

T → (**E**)

`int + int`

Leftmost BFS

Worklist



E → **T**

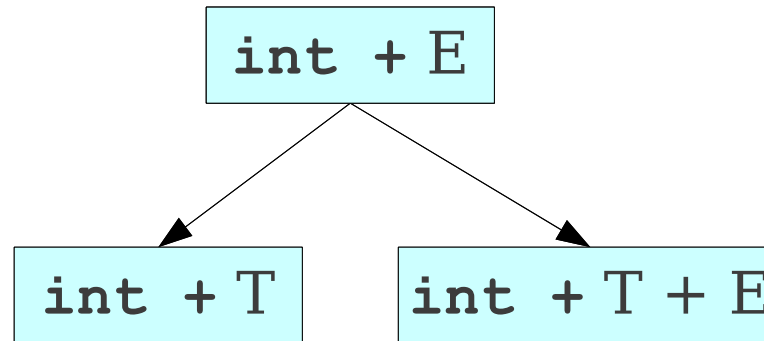
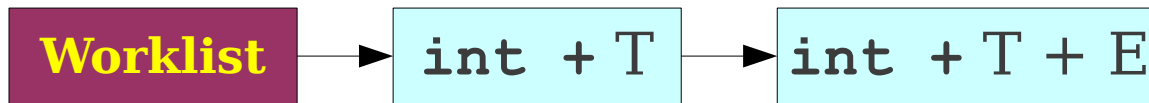
E → **T** + **E**

T → **int**

T → (**E**)

int + int

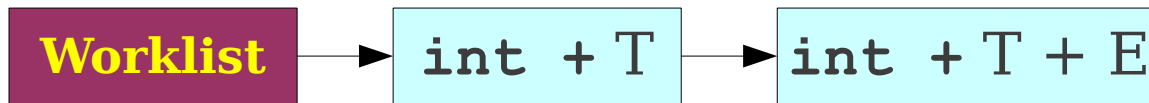
Leftmost BFS



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

$\text{int} + \text{int}$

Leftmost BFS



E → **T**

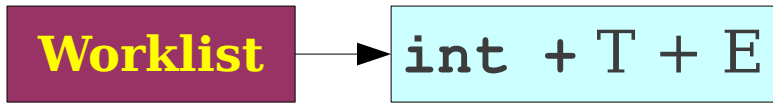
E → **T** + **E**

T → **int**

T → **(E)**

`int + int`

Leftmost BFS

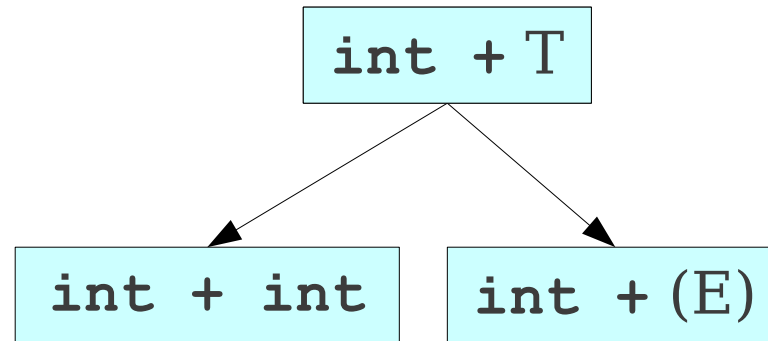
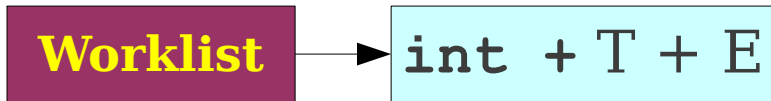


int + T

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

int + int

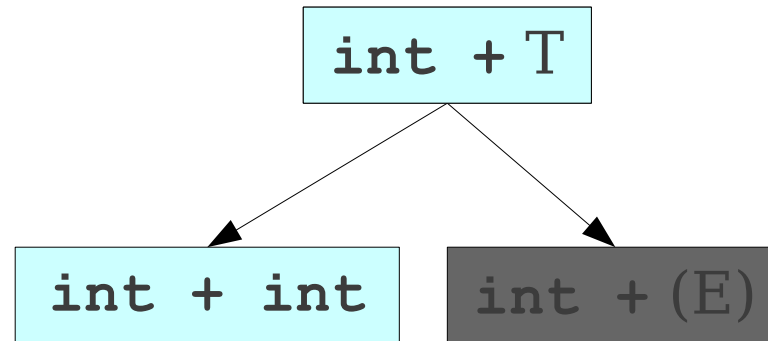
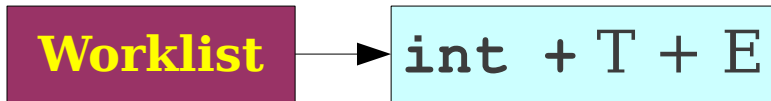
Leftmost BFS



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

int + int

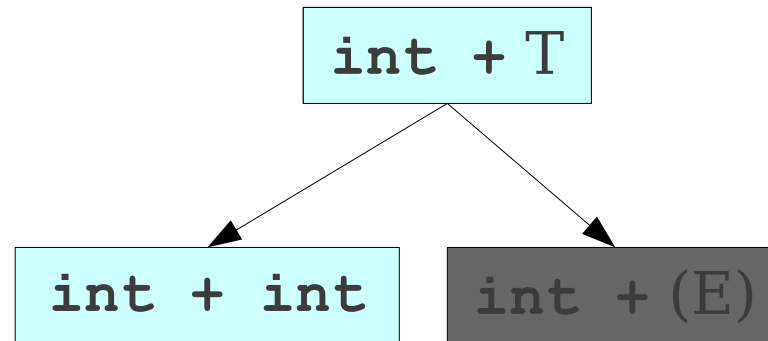
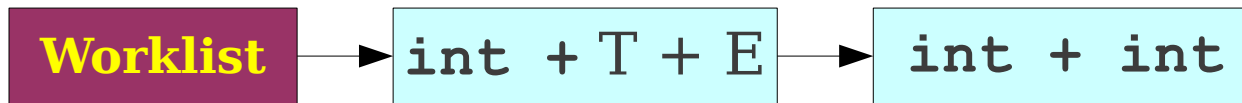
Leftmost BFS



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

int + int

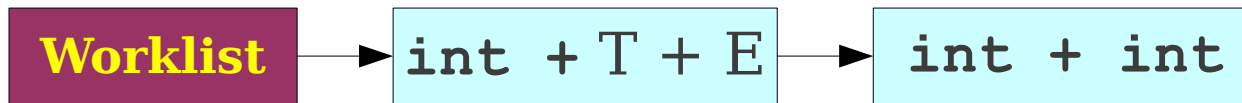
Leftmost BFS



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

`int + int`

Leftmost BFS



E → **T**

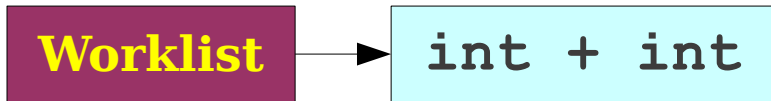
E → **T** + **E**

T → **int**

T → **(E)**

int + int

Leftmost BFS

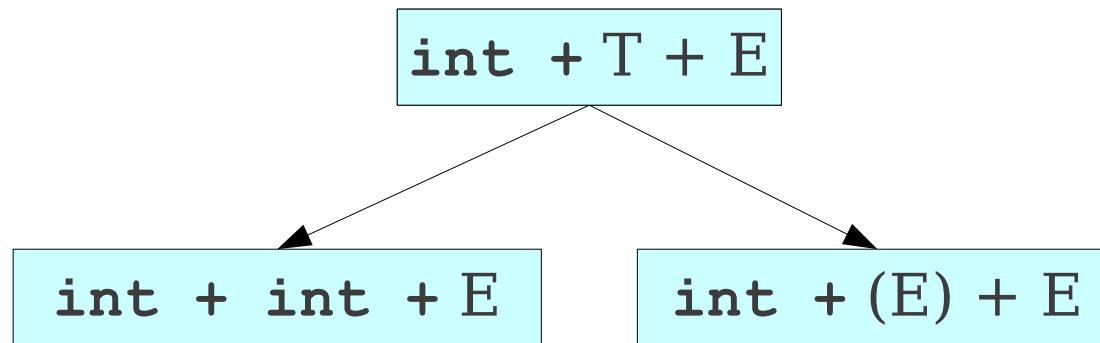
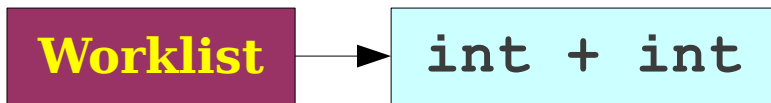


int + T + E

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

int + int

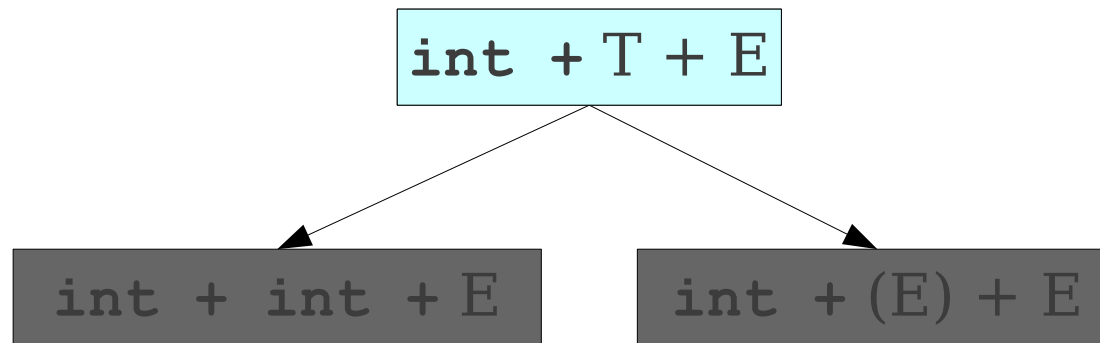
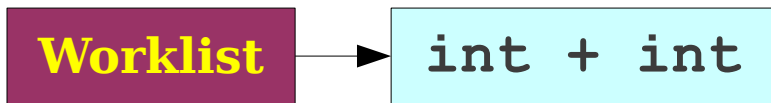
Leftmost BFS



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

`int + int`

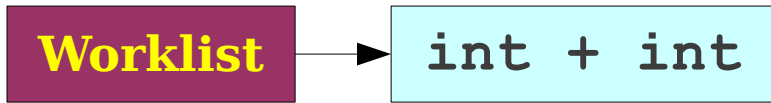
Leftmost BFS



$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

int + int

Leftmost BFS



E → **T**

E → **T** + **E**

T → **int**

T → **(E)**

`int + int`

Leftmost BFS

Worklist

int + int

E → **T**

E → **T** + **E**

T → int

T → (**E**)

int + int

Leftmost BFS

Worklist



int + int

E \rightarrow **T**

E → **T** + **E**

T → **int**

T \rightarrow **(E)**

int + int

Leftmost BFS

- Substantial improvement over naïve algorithm.
- Will always find a valid parse of a program if one exists.
- Can easily be modified to find if a program can't be parsed.
- But, there are still problems.

Leftmost BFS Has Problems

Worklist

A → **A****a** | **A****b** | **c**

Leftmost BFS Has Problems

Worklist

A → **A****a** | **A****b** | **c**

caaaaaaaaaa

Leftmost BFS Has Problems



A → **A***a* | **A***b* | *c*

caaaaaaaaaa

Leftmost BFS Has Problems

Worklist

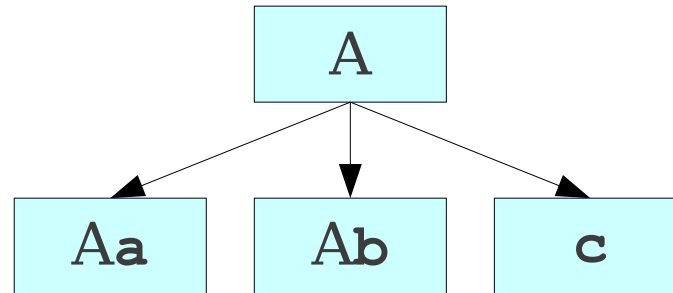
A

A → **A****a** | **A****b** | **c**

caaaaaaaaaa

Leftmost BFS Has Problems

Worklist

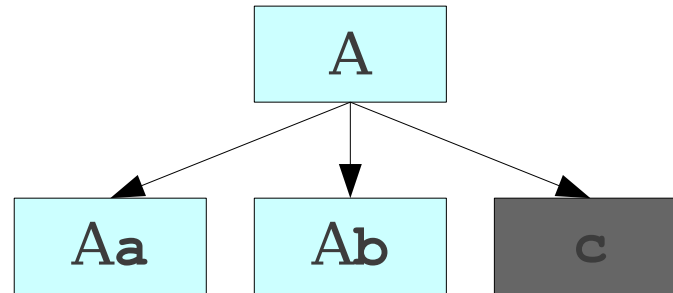


A → **Aa** | **Ab** | **c**

caaaaaaaaaa

Leftmost BFS Has Problems

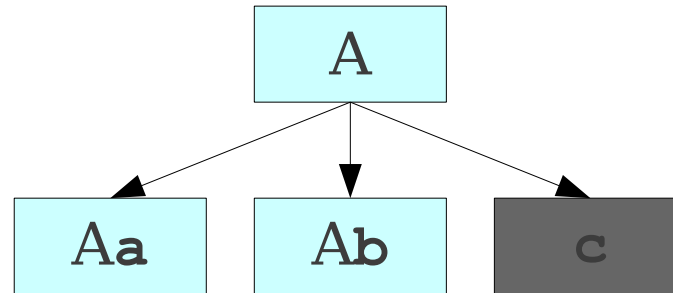
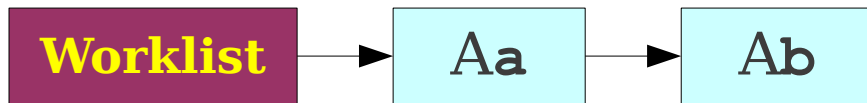
Worklist



A → **A****a** | **A****b** | **c**

caaaaaaaaaa

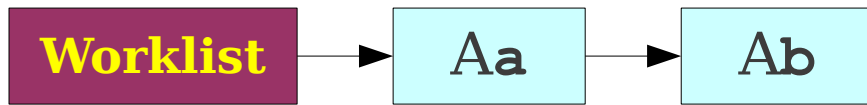
Leftmost BFS Has Problems



A → **A****a** | **A****b** | **c**

caaaaaaaaaa

Leftmost BFS Has Problems



A → **A****a** | **A****b** | **c**

caaaaaaaaaa

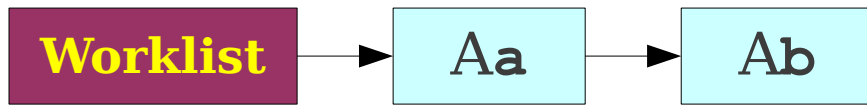
Leftmost BFS Has Problems



A → **A****a** | **A****b** | **c**

caaaaaaaaaa

Leftmost BFS Has Problems



A → **A****a** | **A****b** | **c**

caaaaaaaaaa

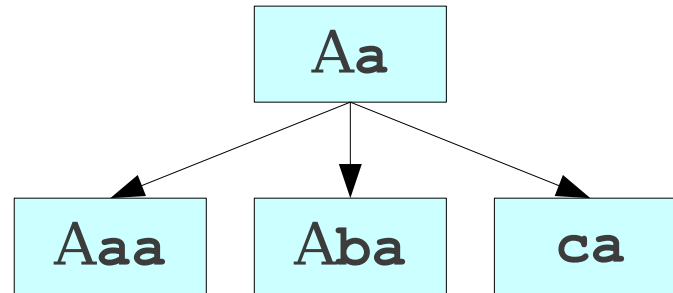
Leftmost BFS Has Problems



A → **Aa** | **Ab** | **c**

caaaaaaaaaa

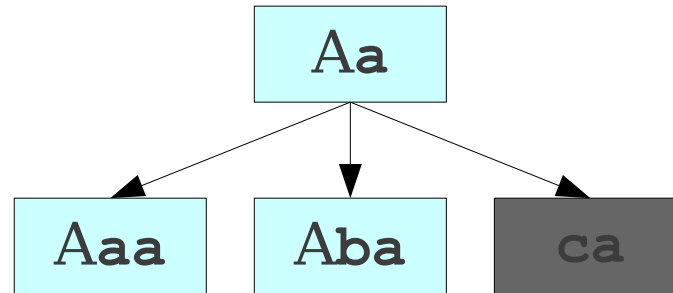
Leftmost BFS Has Problems



A → **Aa** | **Ab** | **c**

caaaaaaaaaa

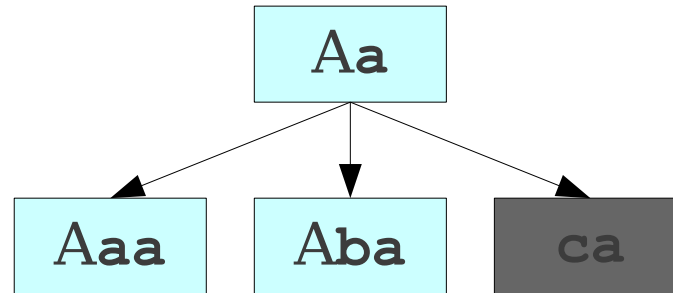
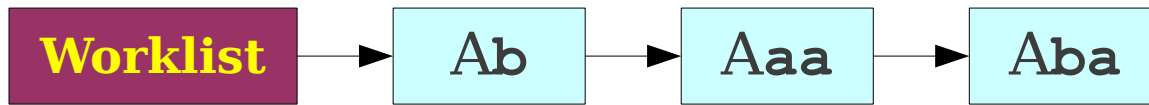
Leftmost BFS Has Problems



A → **Aa** | **Ab** | **c**

caaaaaaaaaa

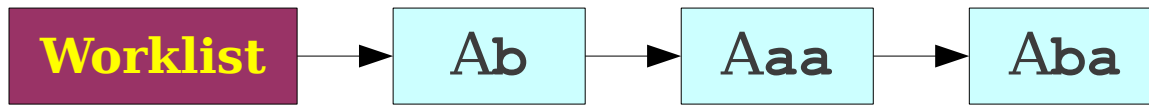
Leftmost BFS Has Problems



A → **Aa** | **Ab** | **c**

caaaaaaaaaa

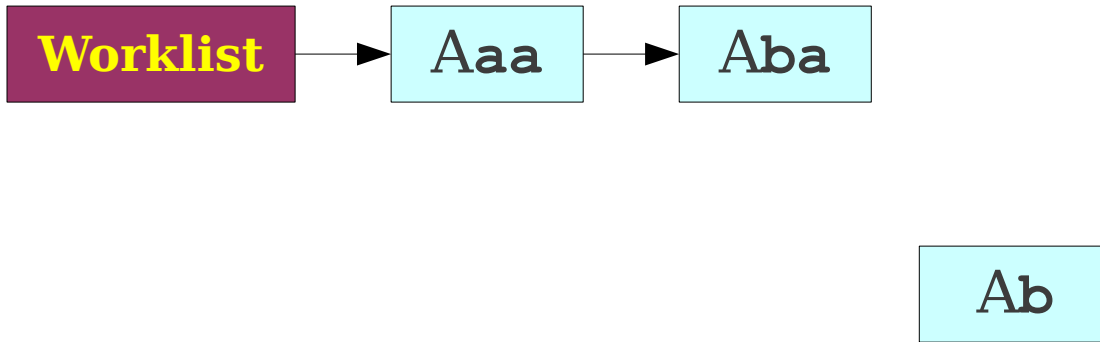
Leftmost BFS Has Problems



A → **Aa** | **Ab** | **c**

caaaaaaaaaa

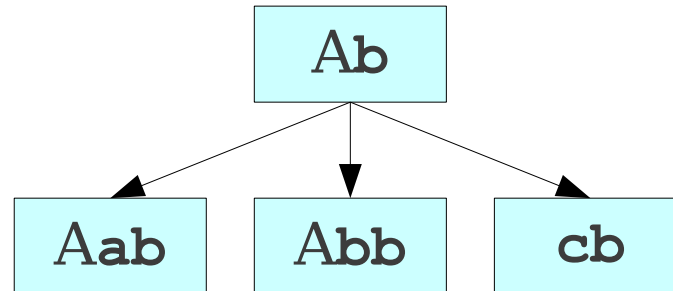
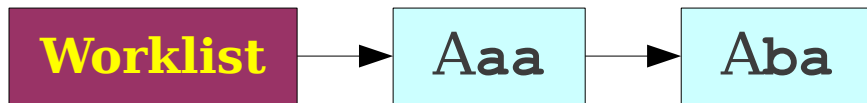
Leftmost BFS Has Problems



A → **Aa** | **Ab** | **c**

caaaaaaaaaa

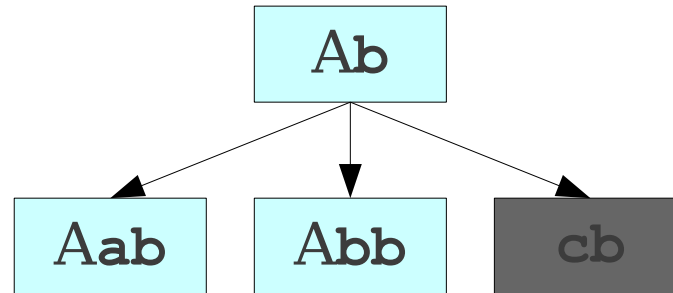
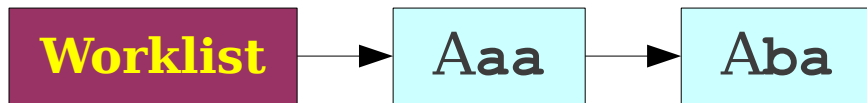
Leftmost BFS Has Problems



A → **Aa** | **Ab** | **c**

caaaaaaaaaa

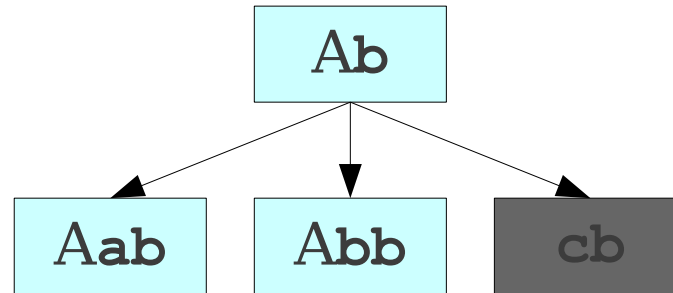
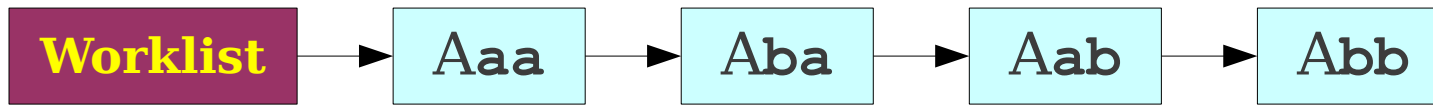
Leftmost BFS Has Problems



A → **Aa** | **Ab** | **c**

caaaaaaaaaa

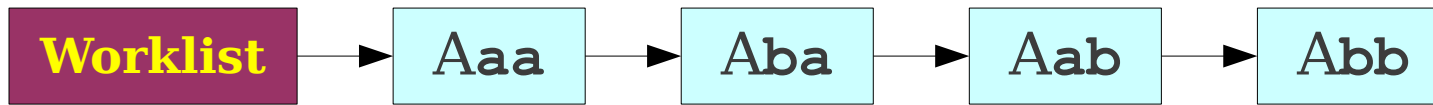
Leftmost BFS Has Problems



A → **Aa** | **Ab** | **c**

caaaaaaaaaa

Leftmost BFS Has Problems



A → **Aa** | **Ab** | **c**

caaaaaaaaaa

Leftmost BFS Has Problems



A → **Aa** | **Ab** | **c**

caaaaaaaaaa

Problems with Leftmost BFS

- Grammars like this can make parsing take exponential time.
- Also uses exponential memory.
- What if we search the graph with a different algorithm?

Leftmost DFS

- Idea: Use **depth-first** search.
- Advantages:
 - Lower memory usage: Only considers one branch at a time.
 - High performance: On many grammars, runs very quickly.
 - Easy to implement: Can be written as a set of mutually recursive functions.

Leftmost DFS

E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

Leftmost DFS

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

`int + int`

Leftmost DFS

E

E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

`int + int`

Leftmost DFS

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
T

`int + int`

Leftmost DFS

E → **T**
E → **T** + **E**
T → **int**
T → (**E**)

E
T
int

int + int

Leftmost DFS

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
T
int

`int + int`

Leftmost DFS

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
T

`int + int`

Leftmost DFS

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
T
(E)

`int + int`

Leftmost DFS

E → **T**
E → **T** + **E**
T → **int**
T → (**E**)

E
T
(E)

`int + int`

Leftmost DFS

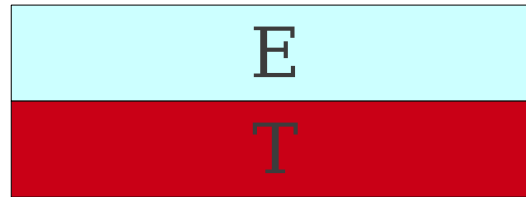
E → **T**
E → **T** + **E**
T → **int**
T → (**E**)

E
T

`int + int`

Leftmost DFS

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$



`int + int`

Leftmost DFS

E

E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

`int + int`

Leftmost DFS

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
T + E

`int + int`

Leftmost DFS

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
T + E
int + E

int + int

Leftmost DFS

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
T + E
int + E
int + T

int + int

Leftmost DFS

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
T + E
int + E
int + T
int + int

int + int

Leftmost DFS

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
T + E
int + E
int + T
int + int



int + int

Problems with Leftmost DFS

A → **A****a** | **c**

A
Aa
Aaa
Aaaa
Aaaaa



c

Left Recursion

- A nonterminal **A** is said to be **left-recursive** iff

$$\mathbf{A} \Rightarrow^* \mathbf{A}\omega$$

for some string ω .

- Leftmost DFS may fail on left-recursive grammars.
- Fortunately, in many cases it is possible to eliminate left recursion (see Handout 08 for details).

Summary of Leftmost BFS/DFS

- Leftmost BFS works on all grammars.
- Worst-case runtime is exponential.
- Worst-case memory usage is exponential.
- Rarely used in practice.
- Leftmost DFS works on grammars without left recursion.
- Worst-case runtime is exponential.
- Worst-case memory usage is linear.
- Often used in a limited form as **recursive descent**.

Predictive Parsing

Predictive Parsing

- The leftmost DFS/BFS algorithms are **backtracking** algorithms.
 - Guess which production to use, then back up if it doesn't work.
 - Try to match a prefix by sheer dumb luck.
- There is another class of parsing algorithms called **predictive** algorithms.
 - Based on remaining input, predict (*without backtracking*) which production to use.

Tradeoffs in Prediction

- Predictive parsers are *fast*.
 - Many predictive algorithms can be made to run in linear time.
 - Often can be table-driven for extra performance.
- Predictive parsers are *weak*.
 - Not all grammars can be accepted by predictive parsers.
- Trade *expressiveness* for *speed*.

Exploiting Lookahead

- Given just the start symbol, how do you know which productions to use to get to the input program?
- Idea: Use **lookahead tokens**.
- When trying to decide which production to use, look at some number of tokens of the input to help make the decision.

Implementing Predictive Parsing

- Predictive parsing is only possible if we can predict which production to use given some number of lookahead tokens.
- Increasing the number of lookahead tokens increases the number of grammars we can parse, but complicates the parser.
- Decreasing the number of lookahead tokens decreases the number of grammars we can parse, but simplifies the parser.

Predictive Parsing

E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

E

E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

E

E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

E
T + E

E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

int	+	(int	+	int)
------------	----------	----------	------------	----------	------------	----------

Predictive Parsing

E
T + E

E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

int	+	(int	+	int)
------------	----------	----------	------------	----------	------------	----------

Predictive Parsing

E
T + E

E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

int	+	(int	+	int)
------------	---	---	-----	---	-----	---

Predictive Parsing

E
T + E
int + E

E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

E
T + E
int + E

E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

int	+	(int	+	int)
------------	----------	----------	------------	----------	------------	----------

Predictive Parsing

E
T + E
int + E

E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

int	+	(int	+	int)
------------	----------	----------	------------	----------	------------	----------

Predictive Parsing

E
T + E
int + E

E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

E
T + E
int + E
int + T

E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

E
T + E
int + E
int + T

E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

E
T + E
int + E
int + T

E → **T**

E → **T** + **E**

T → **int**

T → (**E**)

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
$T + E$
$\text{int} + E$
$\text{int} + T$
$\text{int} + (E)$

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
$T + E$
$\text{int} + E$
$\text{int} + T$
$\text{int} + (E)$

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
$T + E$
$\text{int} + E$
$\text{int} + T$
$\text{int} + (E)$

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
$T + E$
$\text{int} + E$
$\text{int} + T$
$\text{int} + (E)$

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
$T + E$
$\text{int} + E$
$\text{int} + T$
$\text{int} + (E)$
$\text{int} + (T + E)$

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
$T + E$
$\text{int} + E$
$\text{int} + T$
$\text{int} + (E)$
$\text{int} + (T + E)$

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
$T + E$
$\text{int} + E$
$\text{int} + T$
$\text{int} + (E)$
$\text{int} + (T + E)$

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
$T + E$
$\text{int} + E$
$\text{int} + T$
$\text{int} + (E)$
$\text{int} + (T + E)$
$\text{int} + (\text{int} + E)$

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
$T + E$
$\text{int} + E$
$\text{int} + T$
$\text{int} + (E)$
$\text{int} + (T + E)$
$\text{int} + (\text{int} + E)$

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
$T + E$
$\text{int} + E$
$\text{int} + T$
$\text{int} + (E)$
$\text{int} + (T + E)$
$\text{int} + (\text{int} + E)$

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
$T + E$
$\text{int} + E$
$\text{int} + T$
$\text{int} + (E)$
$\text{int} + (T + E)$
$\text{int} + (\text{int} + E)$

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
$T + E$
$\text{int} + E$
$\text{int} + T$
$\text{int} + (E)$
$\text{int} + (T + E)$
$\text{int} + (\text{int} + E)$
$\text{int} + (\text{int} + T)$

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
$T + E$
$\text{int} + E$
$\text{int} + T$
$\text{int} + (E)$
$\text{int} + (T + E)$
$\text{int} + (\text{int} + E)$
$\text{int} + (\text{int} + T)$

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
$T + E$
$\text{int} + E$
$\text{int} + T$
$\text{int} + (E)$
$\text{int} + (T + E)$
$\text{int} + (\text{int} + E)$
$\text{int} + (\text{int} + T)$

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
$T + E$
$\text{int} + E$
$\text{int} + T$
$\text{int} + (E)$
$\text{int} + (T + E)$
$\text{int} + (\text{int} + E)$
$\text{int} + (\text{int} + T)$
$\text{int} + (\text{int} + \text{int})$

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
$T + E$
$\text{int} + E$
$\text{int} + T$
$\text{int} + (E)$
$\text{int} + (T + E)$
$\text{int} + (\text{int} + E)$
$\text{int} + (\text{int} + T)$
$\text{int} + (\text{int} + \text{int})$

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
$T + E$
$\text{int} + E$
$\text{int} + T$
$\text{int} + (E)$
$\text{int} + (T + E)$
$\text{int} + (\text{int} + E)$
$\text{int} + (\text{int} + T)$
$\text{int} + (\text{int} + \text{int})$

int	+	(int	+	int)
-----	---	---	-----	---	-----	---

Predictive Parsing

$E \rightarrow T$
 $E \rightarrow T + E$
 $T \rightarrow \text{int}$
 $T \rightarrow (E)$

E
$T + E$
$\text{int} + E$
$\text{int} + T$
$\text{int} + (E)$
$\text{int} + (T + E)$
$\text{int} + (\text{int} + E)$
$\text{int} + (\text{int} + T)$
$\text{int} + (\text{int} + \text{int})$

int	+	(int	+	int)
-----	---	---	-----	---	-----	---



A Simple Predictive Parser: **LL(1)**

- Top-down, predictive parsing:
 - **L**: Left-to-right scan of the tokens
 - **L**: Leftmost derivation.
 - **(1)**: One token of lookahead
- Construct a leftmost derivation for the sequence of tokens.
- When expanding a nonterminal, we predict the production to use by looking at the next token of the input. **The decision is forced.**

LL(1) Parse Tables

LL(1) Parse Tables

E \rightarrow **int**

E \rightarrow (**E Op E**)

Op \rightarrow **+**

Op \rightarrow *****

LL(1) Parse Tables

E \rightarrow **int**

E \rightarrow **(E Op E)**

Op \rightarrow **+**

Op \rightarrow *****

	int	()	+	*
E	int	(E Op E)			
Op				+	*

LL(1) Parsing

(int + (int * int))

(1) **E** → int

(2) **E** → (E **Op** E)

(3) **Op** → +

(4) **Op** → *

LL(1) Parsing

E	(int + (int * int))
---	---------------------

(1) **E** → **int**

(2) **E** → **(E Op E)**

(3) **Op** → **+**

(4) **Op** → *****

LL(1) Parsing

E	(int + (int * int))
---	---------------------

(1) **E** \rightarrow int

(2) **E** \rightarrow (**E Op E**)

(3) **Op** \rightarrow +

(4) **Op** \rightarrow *

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Parsing

E\$

(int + (int * int))\$

(1) **E** → int

(2) **E** → (E **Op** E)

(3) **Op** → +

(4) **Op** → *

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Parsing

E\$

(int + (int * int))\$

(1) **E** → int

(2) **E** → (**E Op E**)

(3) **Op** → +

(4) **Op** → *

	int	()	+	*
E	1	2			
Op				3	4

The **\$** symbol is the end-of-input marker and is used by the parser to detect when we have reached the end of the input. It is not a part of the grammar.

LL(1) Parsing

E\$

(int + (int * int))\$

(1) **E** → int

(2) **E** → (E **Op** E)

(3) **Op** → +

(4) **Op** → *

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Parsing

E\$

(int + (int * int))\$

(1) **E** → int

(2) **E** → (**E Op E**)

(3) **Op** → +

(4) **Op** → *

	int	()	+	*
E	1	2			
Op				3	4

The first symbol of our guess is a nonterminal. We then look at our parsing table to see what production to use.

This is called a **predict** step.

LL(1) Parsing

E\$

(int + (int * int))\$

(1) **E** → int

(2) **E** → (E **Op** E)

(3) **Op** → +

(4) **Op** → *

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Parsing

- (1) **E** \rightarrow **int**
- (2) **E** \rightarrow **(E Op E)**
- (3) **Op** \rightarrow **+**
- (4) **Op** \rightarrow *****

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Parsing

- (1) **E** \rightarrow **int**
- (2) **E** \rightarrow **(E Op E)**
- (3) **Op** \rightarrow **+**
- (4) **Op** \rightarrow *****

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$

The first symbol of our guess is now a terminal symbol. We thus match it against the first symbol of the string to parse.

This is called a **match** step.

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Parsing

(1) **E** \rightarrow **int**

(2) **E** \rightarrow (**E Op E**)

(3) **Op** \rightarrow **+**

(4) **Op** \rightarrow *****

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Parsing

(1) **E** \rightarrow **int**

(2) **E** \rightarrow (**E Op E**)

(3) **Op** \rightarrow **+**

(4) **Op** \rightarrow *****

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$
Op E)\$	+ (int * int))\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$
Op E)\$	+ (int * int))\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$
Op E)\$	+ (int * int))\$
+ E)\$	+ (int * int))\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$
Op E)\$	+ (int * int))\$
+ E)\$	+ (int * int))\$
E)\$	(int * int))\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$
Op E)\$	+ (int * int))\$
+ E)\$	+ (int * int))\$
E)\$	(int * int))\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$
Op E)\$	+ (int * int))\$
+ E)\$	+ (int * int))\$
E)\$	(int * int))\$
(E Op E))\$	(int * int))\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

	int	()	+	*
E	1	2			
Op				3	4

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$
Op E)\$	+ (int * int))\$
+ E)\$	+ (int * int))\$
E)\$	(int * int))\$
(E Op E))\$	(int * int))\$
E Op E))\$	int * int))\$

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

	int	()	+	*
E	1	2			
Op				3	4

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$
Op E)\$	+ (int * int))\$
+ E)\$	+ (int * int))\$
E)\$	(int * int))\$
(E Op E))\$	(int * int))\$
E Op E))\$	int * int))\$

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

	int	()	+	*
E	1	2			
Op				3	4

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$
Op E)\$	+ (int * int))\$
+ E)\$	+ (int * int))\$
E)\$	(int * int))\$
(E Op E))\$	(int * int))\$
E Op E))\$	int * int))\$
int Op E))\$	int * int))\$

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

	int	()	+	*
E	1	2			
Op				3	4

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$
Op E)\$	+ (int * int))\$
+ E)\$	+ (int * int))\$
E)\$	(int * int))\$
(E Op E))\$	(int * int))\$
E Op E))\$	int * int))\$
int Op E))\$	int * int))\$
Op E))\$	* int))\$

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

	int	()	+	*
E	1	2			
Op				3	4

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$
Op E)\$	+ (int * int))\$
+ E)\$	+ (int * int))\$
E)\$	(int * int))\$
(E Op E))\$	(int * int))\$
E Op E))\$	int * int))\$
int Op E))\$	int * int))\$
Op E))\$	* int))\$

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

	int	()	+	*
E	1	2			
Op				3	4

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$
Op E)\$	+ (int * int))\$
+ E)\$	+ (int * int))\$
E)\$	(int * int))\$
(E Op E))\$	(int * int))\$
E Op E))\$	int * int))\$
int Op E))\$	int * int))\$
Op E))\$	* int))\$
* E))\$	* int))\$

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

	int	()	+	*
E	1	2			
Op				3	4

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$
Op E)\$	+ (int * int))\$
+ E)\$	+ (int * int))\$
E)\$	(int * int))\$
(E Op E))\$	(int * int))\$
E Op E))\$	int * int))\$
int Op E))\$	int * int))\$
Op E))\$	* int))\$
* E))\$	* int))\$
E))\$	int))\$

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

	int	()	+	*
E	1	2			
Op				3	4

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$
Op E)\$	+ (int * int))\$
+ E)\$	+ (int * int))\$
E)\$	(int * int))\$
(E Op E))\$	(int * int))\$
E Op E))\$	int * int))\$
int Op E))\$	int * int))\$
Op E))\$	* int))\$
* E))\$	* int))\$
E))\$	int))\$

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

	int	()	+	*
E	1	2			
Op				3	4

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$
Op E)\$	+ (int * int))\$
+ E)\$	+ (int * int))\$
E)\$	(int * int))\$
(E Op E))\$	(int * int))\$
E Op E))\$	int * int))\$
int Op E))\$	int * int))\$
Op E))\$	* int))\$
* E))\$	* int))\$
E))\$	int))\$
int))\$	int))\$

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

	int	()	+	*
E	1	2			
Op				3	4

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$
Op E)\$	+ (int * int))\$
+ E)\$	+ (int * int))\$
E)\$	(int * int))\$
(E Op E))\$	(int * int))\$
E Op E))\$	int * int))\$
int Op E))\$	int * int))\$
Op E))\$	* int))\$
* E))\$	* int))\$
E))\$	int))\$
int))\$	int))\$
)\$)\$

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

	int	()	+	*
E	1	2			
Op				3	4

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$
Op E)\$	+ (int * int))\$
+ E)\$	+ (int * int))\$
E)\$	(int * int))\$
(E Op E))\$	(int * int))\$
E Op E))\$	int * int))\$
int Op E))\$	int * int))\$
Op E))\$	* int))\$
* E))\$	* int))\$
E))\$	int))\$
int))\$	int))\$
)\$)\$
)\$)\$

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

	int	()	+	*
E	1	2			
Op				3	4

E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$
Op E)\$	+ (int * int))\$
+ E)\$	+ (int * int))\$
E)\$	(int * int))\$
(E Op E))\$	(int * int))\$
E Op E))\$	int * int))\$
int Op E))\$	int * int))\$
Op E))\$	* int))\$
* E))\$	* int))\$
E))\$	int))\$
int))\$	int))\$
)\$)\$
)\$)\$
\$	\$

LL(1) Parsing

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

	int	()
E	1	2	
Op			



E\$	(int + (int * int))\$
(E Op E)\$	(int + (int * int))\$
E Op E)\$	int + (int * int))\$
int Op E)\$	int + (int * int))\$
Op E)\$	+ (int * int))\$
	+ (int * int))\$
	(int * int))\$
	(int * int))\$
	int * int))\$
	int * int))\$
	* int))\$
	* int))\$
	int))\$
int))\$	int))\$
)\$)\$
)\$)\$
\$	\$

LL(1) Error Detection

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

int + int\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Error Detection

(1) **E** \rightarrow **int**

(2) **E** \rightarrow (**E Op E**)

(3) **Op** \rightarrow **+**

(4) **Op** \rightarrow *****

E\$	int + int\$
-----	-------------

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Error Detection

(1) **E** \rightarrow **int**

(2) **E** \rightarrow (**E Op E**)

(3) **Op** \rightarrow **+**

(4) **Op** \rightarrow *****

E\$	int + int\$
-----	-------------

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Error Detection

(1) **E** \rightarrow **int**

(2) **E** \rightarrow (**E Op E**)

(3) **Op** \rightarrow **+**

(4) **Op** \rightarrow *****

E\$	int + int\$
int \$	int + int\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Error Detection

(1) **E** \rightarrow **int**

(2) **E** \rightarrow (**E Op E**)

(3) **Op** \rightarrow **+**

(4) **Op** \rightarrow *****

E\$	int + int\$
int \$	int + int\$
\$	+ int\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Error Detection

(1) **E** \rightarrow **int**

(2) **E** \rightarrow (**E Op E**)

(3) **Op** \rightarrow **+**

(4) **Op** \rightarrow *****

E\$	int + int\$
int \$	int + int\$
\$	+ int\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Error Detection

(1) **E** \rightarrow **int**

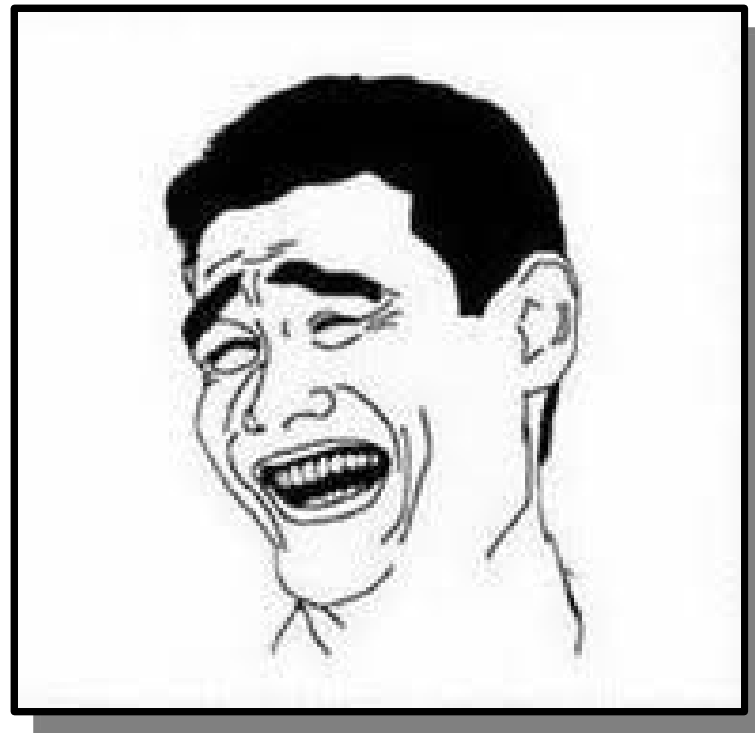
(2) **E** \rightarrow (**E Op E**)

(3) **Op** \rightarrow **+**

(4) **Op** \rightarrow *****

E\$	int + int\$
int \$	int + int\$
\$	+ int\$

	int	()	+	*
E	1	2			
Op				3	4



LL(1) Error Detection, Part II

(1) **E** \rightarrow **int**

(2) **E** \rightarrow (**E Op E**)

(3) **Op** \rightarrow **+**

(4) **Op** \rightarrow *****

(int (int))\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Error Detection, Part II

(1) **E** \rightarrow **int**

(2) **E** \rightarrow (**E Op E**)

(3) **Op** \rightarrow **+**

(4) **Op** \rightarrow *****

E\$	(int (int))\$
-----	---------------

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Error Detection, Part II

(1) **E** \rightarrow **int**

(2) **E** \rightarrow (**E Op E**)

(3) **Op** \rightarrow **+**

(4) **Op** \rightarrow *****

E\$	(int (int))\$
-----	---------------

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Error Detection, Part II

(1) **E** \rightarrow **int**

(2) **E** \rightarrow (**E Op E**)

(3) **Op** \rightarrow **+**

(4) **Op** \rightarrow *****

E\$	(int (int))\$
(E Op E) \$	(int (int))\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Error Detection, Part II

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

E\$	(int (int))\$
(E Op E)\$	(int (int))\$
E Op E)\$	int (int))\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Error Detection, Part II

(1) **E** \rightarrow **int**

(2) **E** \rightarrow **(E Op E)**

(3) **Op** \rightarrow **+**

(4) **Op** \rightarrow *****

E\$	(int (int))\$
(E Op E)\$	(int (int))\$
E Op E)\$	int (int))\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Error Detection, Part II

(1) **E** \rightarrow **int**

(2) **E** \rightarrow (**E Op E**)

(3) **Op** \rightarrow **+**

(4) **Op** \rightarrow *****

E\$	(int (int))\$
(E Op E)\$	(int (int))\$
E Op E)\$	int (int))\$
int Op E)\$	int (int))\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Error Detection, Part II

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

E\$	(int (int))\$
(E Op E)\$	(int (int))\$
E Op E)\$	int (int))\$
int Op E)\$	int (int))\$
Op E)\$	(int))\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Error Detection, Part II

(1) **E** → **int**

(2) **E** → (**E Op E**)

(3) **Op** → **+**

(4) **Op** → *****

E\$	(int (int))\$
(E Op E)\$	(int (int))\$
E Op E)\$	int (int))\$
int Op E)\$	int (int))\$
Op E)\$	(int))\$

	int	()	+	*
E	1	2			
Op				3	4

LL(1) Error Detection, Part II

(1) **E** \rightarrow **int**

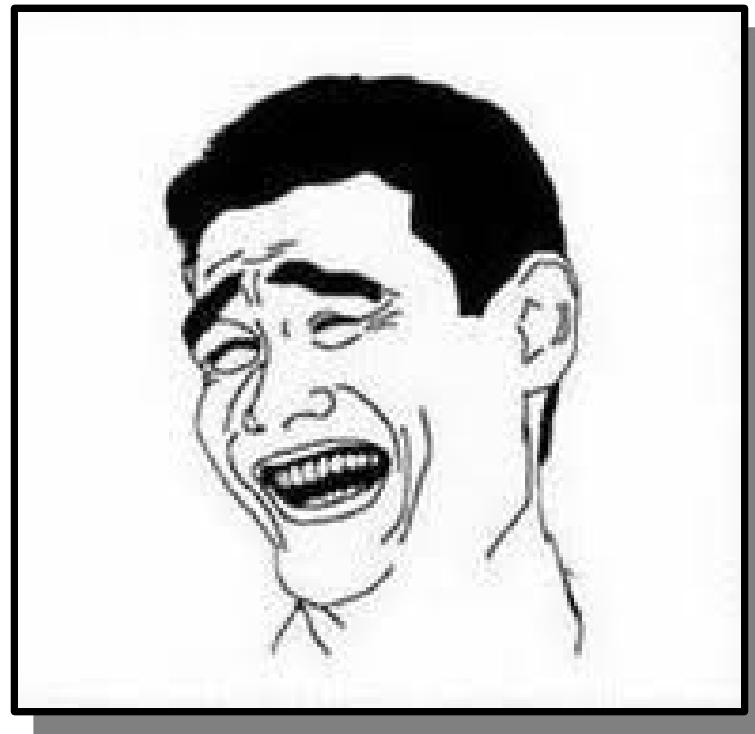
(2) **E** \rightarrow (**E Op E**)

(3) **Op** \rightarrow **+**

(4) **Op** \rightarrow *****

E\$	(int (int))\$
(E Op E)\$	(int (int))\$
E Op E)\$	int (int))\$
int Op E)\$	int (int))\$
Op E)\$	(int))\$

	int	()	+	*
E	1	2			
Op				3	4



The LL(1) Algorithm

- Suppose a grammar has start symbol **S** and LL(1) parsing table T. We want to parse string ω
- Initialize a stack containing **S** and $\$$.
- Repeat until the stack is empty:
 - Let the next character of ω be **t**.
 - If the top of the stack is a terminal **r**:
 - If **r** and **t** don't match, report an error.
 - Otherwise consume the character **t** and pop **r** from the stack.
 - Otherwise, the top of the stack is a nonterminal **A**:
 - If $T[\mathbf{A}, \mathbf{t}]$ is undefined, report an error.
 - Replace the top of the stack with $T[\mathbf{A}, \mathbf{t}]$.

A Simple LL(1) Grammar

STMT → if **EXPR** then **STMT**
 | while **EXPR** do **STMT**
 | **EXPR** ;

EXPR → **TERM** -> id
 | zero? **TERM**
 | not **EXPR**
 | ++ id
 | -- id

TERM → id
 | constant

A Simple LL(1) Grammar

STMT → **if** **EXPR** **then** **STMT**
 | **while** **EXPR** **do** **STMT**
 | **EXPR ;**

EXPR → **TERM** → **id** **id** → **id**;
 | **zero?** **TERM** **while** **not** **zero?** **id**
 | **not** **EXPR** **do** **--id**;
 | **++ id** **if** **not** **zero?** **id** **then**
 | **-- id** **if** **not** **zero?** **id** **then**

TERM → **id** **constant** → **id**;
 | **constant**

Constructing LL(1) Parse Tables

STMT	→	if EXPR then STMT	(1)
		while EXPR do STMT	(2)
		EXPR ;	(3)

EXPR	→	TERM -> id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

$$\begin{array}{lll} \text{TERM} & \rightarrow & \text{id} & (9) \\ & | & \text{constant} & (10) \end{array}$$
[illegible]

Constructing LL(1) Parse Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR ;** (3)

EXPR	→	TERM -> id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

TERM \rightarrow **id** (9)
 \mid **constant** (10)

[illegible]

Constructing LL(1) Parse Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR ;** (3)

EXPR	→	TERM -> id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

TERM \rightarrow **id** (9)
 \mid **constant** (10)

[illegible]

Constructing LL(1) Parse Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR ;** (3)

EXPR	→	TERM -> id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

TERM
 |

→ **id**
constant

(9)
(10)

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT												
EXPR												
TERM										9	10	

Constructing LL(1) Parse Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR ;** (3)

EXPR	→	TERM → id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

$$\begin{array}{lll} \text{TERM} & \rightarrow & \text{id} & (9) \\ & | & \text{constant} & (10) \end{array}$$
[illegible]

Constructing LL(1) Parse Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR ;** (3)

EXPR	→	TERM → id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

TERM \rightarrow **id** (9)
 \mid **constant** (10)

[illegible]

Constructing LL(1) Parse Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR ;** (3)

EXPR	→	TERM → id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

TERM \rightarrow **id** (9)
 \mid **constant** (10)

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT												
EXPR					5	6	7	8		4	4	
TERM										9	10	

Constructing LL(1) Parse Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR ;** (3)

EXPR	→	TERM -> id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

$$\begin{array}{lll} \text{TERM} & \rightarrow & \text{id} & (9) \\ & | & \text{constant} & (10) \end{array}$$

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT												
EXPR					5	6	7	8		4	4	
TERM										9	10	

Constructing LL(1) Parse Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR ;** (3)

EXPR	→	TERM -> id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

$$\begin{array}{lll} \text{TERM} & \rightarrow & \text{id} & (9) \\ & | & \text{constant} & (10) \end{array}$$
[illegible]

Constructing LL(1) Parse Tables

$$\text{STMT} \rightarrow \text{if } \text{EXPR} \text{ then STMT} \quad (1)$$

while **EXPR** **do** **STMT** (2)

$$\text{EXPR} ; \quad (3)$$
$$\mathbf{EXPR} \rightarrow \mathbf{TERM} \rightarrow \text{id} \quad (4)$$
$$\text{zero? TERM} \quad (5)$$
$$\text{not } \mathbf{EXPR} \quad (6)$$
$$++ \text{ id} \quad (7)$$
$$--id \quad (8)$$
$$\text{TERM} \rightarrow \text{id} \quad (9)$$
$$\text{constant} \quad (10)$$
[illegible]

Constructing LL(1) Parse Tables

$$\text{STMT} \rightarrow \text{if } \text{EXPR} \text{ then } \text{STMT} \quad (1)$$

while **EXPR** **do** **STMT** **(2)**

$$| \text{EXPR} ; \quad (3)$$
$$\mathbf{EXPR} \rightarrow \mathbf{TERM} \rightarrow \text{id} \quad (4)$$
$$\text{zero? TERM} \quad (5)$$

$$\text{not } \text{EXPR} \quad (6)$$

$$| \quad ++ \text{ id} \quad (7)$$
$$| \quad -- id \qquad \qquad \qquad (8)$$
$$\text{TERM} \rightarrow \text{id} \quad (9)$$
$$| \text{constant} \quad (10)$$
[illegible]

Constructing LL(1) Parse Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR ;** (3)

EXPR	→	TERM -> id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

TERM \rightarrow **id** (9)
 \mid **constant** (10)

[illegible]

Constructing LL(1) Parse Tables

STMT	→	if EXPR then STMT	(1)
		while EXPR do STMT	(2)
		EXPR ;	(3)

EXPR	→	TERM -> id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

TERM \rightarrow **id** (9)
 \mid **constant** (10)

[illegible]

Constructing LL(1) Parse Tables

STMT	→	if EXPR then STMT	(1)
		while EXPR do STMT	(2)
		EXPR ;	(3)

EXPR	→	TERM -> id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

$$\begin{array}{lll} \text{TERM} & \rightarrow & \text{id} & (9) \\ & | & \text{constant} & (10) \end{array}$$
[illegible]

Can we find an algorithm for constructing LL(1) parse tables?

Filling in Table Entries

- Intuition: The next character should uniquely identify a production, so we should pick a production that ultimately starts with that character.
- $T[\mathbf{A}, \mathbf{t}]$ should be a production $\mathbf{A} \rightarrow \omega$ iff ω derives something starting with \mathbf{t} .
- More rigorously:
$$T[\mathbf{A}, \mathbf{t}] = \mathbf{B}\omega \text{ iff } \mathbf{A} \rightarrow \omega \text{ and } \omega \Rightarrow^* \mathbf{t}\omega'$$

In what follows, assume that our grammar does not contain any ε -productions.

(We'll relax this restriction later.)

FIRST Sets

- We want to tell if a particular nonterminal **A** derives a string starting with a particular nonterminal **t**.

- We can formalize this with **FIRST sets**.

$$\text{FIRST}(\mathbf{A}) = \{ \mathbf{t} \mid \mathbf{A} \Rightarrow^* \mathbf{t}\omega \text{ for some } \omega \}$$

- Intuitively, $\text{FIRST}(\mathbf{A})$ is the set of terminals that can be at the start of a string produced by **A**.
- If we can compute FIRST sets for all nonterminals in a grammar, we can efficiently construct the LL(1) parsing table. Details soon.

Computing FIRST Sets

- Initially, for all nonterminals **A**, set
$$\text{FIRST}(\mathbf{A}) = \{ \mathbf{t} \mid \mathbf{A} \rightarrow \mathbf{t}\omega \text{ for some } \omega \}$$
- Then, repeat the following until no changes occur: For each nonterminal **A**, for each production $\mathbf{A} \rightarrow \mathbf{B}\omega$, set
$$\text{FIRST}(\mathbf{A}) = \text{FIRST}(\mathbf{A}) \cup \text{FIRST}(\mathbf{B})$$
- This is known as a **fixed-point iteration** or a **transitive closure algorithm**.

Iterative FIRST Computations

STMT → **if** **EXPR** **then** **STMT**
| **while** **EXPR** **do** **STMT**
| **EXPR** ;

EXPR → **TERM** -> **id**
| **zero?** **TERM**
| **not** **EXPR**
| ++ **id**
| -- **id**

TERM → **id**
| **constant**

Iterative FIRST Computations

STMT → **if** **EXPR** **then** **STMT**
| **while** **EXPR** **do** **STMT**
| **EXPR ;**

EXPR → **TERM** **->** **id**
| **zero?** **TERM**
| **not** **EXPR**
| **++ id**
| **-- id**

TERM → **id**
| **constant**

STMT	EXPR	TERM

Iterative FIRST Computations

```

STMT  →  if EXPR then STMT
          |  while EXPR do STMT
          |  EXPR ;

```

```

EXPR  →  TERM -> id
        |
        |
        |  zero? TERM
        |  not EXPR
        |  ++ id
        |  -- id

```

TERM \rightarrow id
 |
 constant

STMT	EXPR	TERM
if while		

Iterative FIRST Computations

STMT → **if** **EXPR** **then** **STMT**
| **while** **EXPR** **do** **STMT**
| **EXPR** ;

EXPR → **TERM** -> **id**
| **zero?** **TERM**
| **not** **EXPR**
| ++ **id**
| -- **id**

TERM → **id**
| **constant**

STMT	EXPR	TERM
if while	zero? not ++ --	

Iterative FIRST Computations

STMT → **if** **EXPR** **then** **STMT**
| **while** **EXPR** **do** **STMT**
| **EXPR** ;

EXPR → **TERM** -> **id**
| **zero?** **TERM**
| **not** **EXPR**
| ++ **id**
| -- **id**

TERM → **id**
| **constant**

STMT	EXPR	TERM
if while	zero? not ++ --	id constant

Iterative FIRST Computations

STMT → if EXPR then STMT
| while EXPR do STMT
| **EXPR ;**

EXPR → TERM -> id
| zero? TERM
| not EXPR
| ++ id
| -- id

TERM → id
| constant

STMT	EXPR	TERM
if while	zero? not ++ --	id constant

Iterative FIRST Computations

STMT → if EXPR then STMT
| while EXPR do STMT
| **EXPR ;**

EXPR → TERM -> id
| zero? TERM
| not EXPR
| ++ id
| -- id

TERM → id
| constant

STMT	EXPR	TERM
if while zero? not ++ --	zero? not ++ --	id constant

Iterative FIRST Computations

STMT → **if** **EXPR** **then** **STMT**
| **while** **EXPR** **do** **STMT**
| **EXPR ;**

EXPR → **TERM** -> **id**
| **zero?** **TERM**
| **not** **EXPR**
| **++ id**
| **-- id**

TERM → **id**
| **constant**

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++		
--		

Iterative FIRST Computations

STMT → if EXPR then STMT
| while EXPR do STMT
| EXPR ;

EXPR → **TERM** -> id
| zero? TERM
| not EXPR
| ++ id
| -- id

TERM → id
| constant

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++		
--		

Iterative FIRST Computations

STMT → if EXPR then STMT
| while EXPR do STMT
| EXPR ;

EXPR → **TERM** -> id
| zero? TERM
| not EXPR
| ++ id
| -- id

TERM → id
| constant

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	

Iterative FIRST Computations

STMT → **if** **EXPR** **then** **STMT**
| **while** **EXPR** **do** **STMT**
| **EXPR** ;

EXPR → **TERM** -> **id**
| **zero?** **TERM**
| **not** **EXPR**
| ++ **id**
| -- **id**

TERM → **id**
| **constant**

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	

Iterative FIRST Computations

STMT → if EXPR then STMT
| while EXPR do STMT
| **EXPR ;**

EXPR → TERM -> id
| zero? TERM
| not EXPR
| ++ id
| -- id

TERM → id
| constant

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	

Iterative FIRST Computations

STMT → if EXPR then STMT
 | while EXPR do STMT
 | **EXPR ;**

EXPR → TERM -> id
 | zero? TERM
 | not EXPR
 | ++ id
 | -- id

TERM → id
 | constant

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

Iterative FIRST Computations

STMT → **if** **EXPR** **then** **STMT**
| **while** **EXPR** **do** **STMT**
| **EXPR** ;

EXPR → **TERM** -> **id**
| **zero?** **TERM**
| **not** **EXPR**
| ++ **id**
| -- **id**

TERM → **id**
| **constant**

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

From FIRST Sets to LL(1) Tables

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
		while EXPR do STMT	(2)
		EXPR ;	(3)
EXPR	→	TERM -> id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)
TERM	→	id	(9)
		constant	(10)

From FIRST Sets to LL(1) Tables

$$\begin{array}{lll} \text{STMT} & \rightarrow & \text{if } \text{EXPR} \text{ then } \text{STMT} & (1) \\ & | & \text{while } \text{EXPR} \text{ do } \text{STMT} & (2) \\ & | & \text{EXPR} ; & (3) \end{array}$$

- EXPR** \rightarrow **TERM** \rightarrow id (4)
- | zero? **TERM** (5)
- | not **EXPR** (6)
- | ++ id (7)
- | -- id (8)

$$\begin{array}{ll} \text{TERM} \rightarrow & \text{id} \quad (9) \\ & | \quad \text{constant} \quad (10) \end{array}$$
[illegible]

From FIRST Sets to LL(1) Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 $\quad \quad \quad$ **while** **EXPR** **do** **STMT** (2)
 $\quad \quad \quad$ **EXPR** ; (3)

- EXPR** \rightarrow **TERM** \rightarrow id (4)
- | zero? **TERM** (5)
- | not **EXPR** (6)
- | ++ id (7)
- | -- id (8)

TERM

\rightarrow

id

constant

(9)

(10)

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

$$\text{STMT} \rightarrow \text{if } \text{EXPR} \text{ then } \text{STMT} \quad (1)$$

while **EXPR** **do** **STMT** (2)

$$\text{EXPR} ; \quad (3)$$
$$\mathbf{EXPR} \rightarrow \mathbf{TERM} \rightarrow \text{id} \quad (4)$$
$$\text{zero? TERM} \quad (5)$$
$$\text{not } \text{EXPR} \quad (6)$$
$$++ \text{ id} \quad (7)$$
$$--id \quad (8)$$
$$\text{TERM} \rightarrow \text{id} \quad (9)$$

constant **(10)**

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

$$\text{STMT} \rightarrow \text{if } \text{EXPR} \text{ then } \text{STMT} \quad (1)$$

while **EXPR** **do** **STMT** (2)

$$\text{EXPR} ; \quad (3)$$
$$\mathbf{EXPR} \rightarrow \mathbf{TERM} \rightarrow \text{id} \quad (4)$$
$$\text{zero? TERM} \quad (5)$$
$$\text{not } \mathbf{EXPR} \quad (6)$$
$$++ \text{ id} \quad (7)$$
$$--id \quad (8)$$
$$\text{TERM} \rightarrow \text{id} \quad (9)$$

constant **(10)**

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

$$\text{STMT} \rightarrow \text{if } \text{EXPR} \text{ then } \text{STMT} \quad (1)$$
$$\text{while } \mathbf{EXPR} \text{ do } \mathbf{STMT} \quad (2)$$

$$\text{EXPR} ; \quad (3)$$

$$\mathbf{EXPR} \rightarrow \mathbf{TERM} \rightarrow \text{id} \quad (4)$$
$$\text{zero? TERM} \quad (5)$$
$$\text{not } \mathbf{EXPR} \quad (6)$$
$$++ \text{ id} \quad (7)$$
$$--id \quad (8)$$
$$\text{TERM} \rightarrow \text{id} \quad (9)$$
$$\text{constant} \quad (10)$$

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

$$\text{STMT} \rightarrow \text{if } \text{EXPR} \text{ then } \text{STMT} \quad (1)$$
$$\text{while } \mathbf{EXPR} \text{ do } \mathbf{STMT} \quad (2)$$

$$\text{EXPR} ; \quad (3)$$

$$\mathbf{EXPR} \rightarrow \mathbf{TERM} \rightarrow \text{id} \quad (4)$$
$$\text{zero? TERM} \quad (5)$$
$$\text{not } \mathbf{EXPR} \quad (6)$$
$$++ \text{ id} \quad (7)$$
$$| \quad -- id \qquad\qquad\qquad (8)$$
$$\text{TERM} \rightarrow \text{id} \quad (9)$$
$$\text{constant} \quad (10)$$

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

$$\text{STMT} \rightarrow \text{if } \text{EXPR} \text{ then } \text{STMT} \quad (1)$$

while **EXPR** **do** **STMT** (2)

$$\text{EXPR} ; \quad (3)$$
$$\mathbf{EXPR} \rightarrow \mathbf{TERM} \rightarrow \text{id} \quad (4)$$
$$\text{zero? TERM} \quad (5)$$
$$\text{not } \mathbf{EXPR} \quad (6)$$
$$++ \text{ id} \quad (7)$$
$$--id \quad (8)$$
$$\text{TERM} \rightarrow \text{id} \quad (9)$$
$$\text{constant} \quad (10)$$

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
		while EXPR do STMT	(2)
		EXPR ;	(3)

EXPR	→	TERM → id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

TERM →

|

id

constant

(9)

(10)

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
		while EXPR do STMT	(2)
		EXPR ;	(3)

- EXPR** \rightarrow **TERM** \rightarrow id (4)
- | zero? **TERM** (5)
- | not **EXPR** (6)
- | ++ id (7)
- | -- id (8)

$$\begin{array}{ll} \text{TERM} \rightarrow & \text{id} \quad (9) \\ & | \quad \text{constant} \quad (10) \end{array}$$

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR ;** (3)

EXPR	→	TERM → id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

$$\begin{array}{ll} \text{TERM} \rightarrow & \text{id} \quad (9) \\ & | \quad \text{constant} \quad (10) \end{array}$$

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR ;** (3)

EXPR	→	TERM → id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

$$\begin{array}{ll} \text{TERM} \rightarrow & \text{id} \quad (9) \\ & | \quad \text{constant} \quad (10) \end{array}$$

STMT	EXPR	TERM
if	zero?	id constant
while	not	
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR ;** (3)

EXPR	→	TERM -> id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

$$\begin{array}{ll} \text{TERM} \rightarrow & \text{id} \quad (9) \\ & | \quad \text{constant} \quad (10) \end{array}$$

STMT	EXPR	TERM
if	zero?	id constant
while	not	
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR** ; (3)

EXPR	\rightarrow	TERM	\rightarrow	id	(4)
		zero?		TERM	(5)
		not		EXPR	(6)
		++		id	(7)
		--		id	(8)

$$\begin{array}{ll} \text{TERM} \rightarrow & \text{id} \quad (9) \\ & | \quad \text{constant} \quad (10) \end{array}$$

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR** ; (3)

EXPR	\rightarrow	TERM	\rightarrow	id	(4)
		zero?		TERM	(5)
		not		EXPR	(6)
		++		id	(7)
		--		id	(8)

$$\begin{array}{ll} \text{TERM} \rightarrow & \text{id} \quad (9) \\ & | \quad \text{constant} \quad (10) \end{array}$$

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

$$\begin{array}{lll} \text{STMT} & \rightarrow & \text{if } \text{EXPR} \text{ then } \text{STMT} & (1) \\ & | & \text{while } \text{EXPR} \text{ do } \text{STMT} & (2) \\ & | & \text{EXPR} ; & (3) \end{array}$$

EXPR	→	TERM → id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

TERM

\rightarrow

id

constant

(9)

(10)

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR** ; (3)

EXPR	→	TERM → id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

$$\begin{array}{ll} \text{TERM} \rightarrow & \text{id} \quad (9) \\ & | \quad \text{constant} \quad (10) \end{array}$$

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR** ; (3)

EXPR	→	TERM → id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

TERM

\rightarrow

id

constant

(9)

(10)

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR** ; (3)

EXPR	→	TERM → id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

$$\begin{array}{ll} \text{TERM} \rightarrow & \text{id} \quad (9) \\ & | \quad \text{constant} \quad (10) \end{array}$$

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR** ; (3)

EXPR	→	TERM -> id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

$$\begin{array}{ll} \text{TERM} \rightarrow & \text{id} \quad (9) \\ & | \quad \text{constant} \quad (10) \end{array}$$

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR** ; (3)

EXPR	→	TERM -> id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

$$\begin{array}{ll} \text{TERM} \rightarrow & \text{id} \quad (9) \\ & | \quad \text{constant} \quad (10) \end{array}$$

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR ;** (3)

EXPR	→	TERM → id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

TERM	\rightarrow	id	(9)
		constant	(10)

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

$$\begin{array}{lll} \text{STMT} & \rightarrow & \text{if } \text{EXPR} \text{ then } \text{STMT} & (1) \\ & | & \text{while } \text{EXPR} \text{ do } \text{STMT} & (2) \\ & | & \text{EXPR} ; & (3) \end{array}$$

- EXPR** \rightarrow **TERM** \rightarrow id (4)
- | zero? **TERM** (5)
- | not **EXPR** (6)
- | ++ id (7)
- | -- id (8)

TERM	\rightarrow	id	(9)
		constant	(10)

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR** ; (3)

EXPR	→	TERM → id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

TERM	\rightarrow	id	(9)
		constant	(10)

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 $\quad \quad \quad$ **while** **EXPR** **do** **STMT** (2)
 $\quad \quad \quad$ **EXPR** ; (3)

- EXPR** \rightarrow **TERM** \rightarrow id (4)
- | zero? **TERM** (5)
- | not **EXPR** (6)
- | ++ id (7)
- | -- id (8)

TERM	\rightarrow	id	(9)
		constant	(10)

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR ;** (3)

EXPR	\rightarrow	TERM \rightarrow id	(4)
		zero? TERM	(5)
		not EXPR	(6)
		++ id	(7)
		-- id	(8)

TERM \rightarrow

|

id

constant

(9)

(10)

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

[illegible]

From FIRST Sets to LL(1) Tables

STMT \rightarrow **if** **EXPR** **then** **STMT** (1)
 | **while** **EXPR** **do** **STMT** (2)
 | **EXPR** ; (3)

```

EXPR  →  TERM -> id
          |
          |  zero? TERM
          |  not EXPR
          |  ++ id
          |  -- id

```

TERM \rightarrow id
 |
 constant

STMT

EXPR

TERM

if

zero?

```
id
constant
```

not

++

— —

id

constant

[illegible]

ϵ -Free LL(1) Parse Tables

- The following algorithm constructs an LL(1) parse table for a grammar with no ϵ -productions.
- Compute the FIRST sets for all nonterminals in the grammar.
- For each production $A \rightarrow t\omega$, set $T[A, t] = t\omega$.
- For each production $A \rightarrow B\omega$, set $T[A, t] = B\omega$ for each $t \in \text{FIRST}(B)$.

Expanding our Grammar

STMT	→	if EXPR then STMT	(1)	id → id;
		while EXPR do STMT	(2)	
		EXPR ;	(3)	while not zero? id do --id;
EXPR	→	TERM -> id	(4)	if not zero? id then
		zero? TERM	(5)	if not zero? id then
		not EXPR	(6)	constant → id;
		++ id	(7)	
		-- id	(8)	
TERM	→	id	(9)	
		constant	(10)	

Expanding our Grammar

STMT	→	if EXPR then STMT	(1)	id → id ;
		while EXPR do STMT	(2)	
		EXPR ;	(3)	while not zero? id do --id;
EXPR	→	TERM -> id	(4)	if not zero? id then
		zero? TERM	(5)	if not zero? id then
		not EXPR	(6)	constant → id ;
		++ id	(7)	
		-- id	(8)	
TERM	→	id	(9)	
		constant	(10)	
BLOCK	→	STMT	(11)	
		{ STMTS }	(12)	
STMTS	→	STMT STMTS	(13)	
		ϵ	(14)	

Expanding our Grammar

STMT	→	if EXPR then BLOCK	(1)	id → id;
		while EXPR do BLOCK	(2)	
		EXPR ;	(3)	while not zero? id do --id;
EXPR	→	TERM -> id	(4)	if not zero? id then
		zero? TERM	(5)	if not zero? id then
		not EXPR	(6)	constant → id;
		++ id	(7)	
		-- id	(8)	
TERM	→	id	(9)	
		constant	(10)	
BLOCK	→	STMT	(11)	
		{ STMTS }	(12)	
STMTS	→	STMT STMTS	(13)	
		ε	(14)	

Expanding our Grammar

STMT	→	if EXPR then BLOCK	(1)	id → id;
		while EXPR do BLOCK	(2)	
		EXPR ;	(3)	while not zero? id do --id;
EXPR	→	TERM -> id	(4)	if not zero? id then
		zero? TERM	(5)	if not zero? id then
		not EXPR	(6)	constant → id;
		++ id	(7)	
		-- id	(8)	if zero? id then
TERM	→	id	(9)	while zero? id do {
		constant	(10)	constant → id;
BLOCK	→	STMT	(11)	constant → id;
		{ STMTS }	(12)	}
STMTS	→	STMT STMTS	(13)	
		ε	(14)	

LL(1) with ϵ -Productions

- Computation of FIRST is different.
 - What if the first nonterminal in a production can produce ϵ ?
- Building the table is different.
 - What action do you take if the correct production produces the empty string?

FIRST Sets with ϵ

FIRST Sets with ϵ

Num \rightarrow **Sign Digits**

Sign \rightarrow **+** | **-** | **ϵ**

Digits \rightarrow **Digit More**

More \rightarrow **Digits** | **ϵ**

Digit \rightarrow **0** | **1** | **2** | ... | **9**

FIRST Sets with ϵ

Num \rightarrow **Sign Digits**

Sign \rightarrow **+** | **-** | **ϵ**

Digits \rightarrow **Digit More**

More \rightarrow **Digits** | **ϵ**

Digit \rightarrow **0** | **1** | **2** | ... | **9**

Num	Sign	Digit	Digits	More

FIRST Sets with ϵ

Num \rightarrow **Sign Digits**
Sign \rightarrow **+** | **-** | **ϵ**
Digits \rightarrow **Digit More**
More \rightarrow **Digits** | **ϵ**
Digit \rightarrow **0** | **1** | **2** | ... | **9**

Num	Sign	Digit	Digits	More
	+ -	0 5 1 6 2 7 3 8 4 9		

FIRST Sets with ϵ

Num → **Sign Digits**

Sign → + | - | ϵ

Digits → **Digit More**

More → **Digits** | ϵ

Digit → 0 | 1 | 2 | ... | 9

Num	Sign	Digit	Digits	More
	+ -	0 5 1 6 2 7 3 8 4 9		

FIRST Sets with ϵ

Num → **Sign Digits**

Sign → + | - | ϵ

Digits → **Digit More**

More → **Digits** | ϵ

Digit → 0 | 1 | 2 | ... | 9

Num	Sign	Digit	Digits	More
+ -	+ -	0 5 1 6 2 7 3 8 4 9		

FIRST Sets with ϵ

Num \rightarrow **Sign Digits**

Sign \rightarrow **+** | **-** | **ϵ**

Digits \rightarrow **Digit More**

More \rightarrow **Digits** | **ϵ**

Digit \rightarrow **0** | **1** | **2** | ... | **9**

Num	Sign	Digit	Digits	More
+ -	+ -	0 5 1 6 2 7 3 8 4 9		

FIRST Sets with ϵ

Num \rightarrow Sign Digits

Sign $\rightarrow + \mid - \mid \epsilon$

Digits \rightarrow **Digit More**

More \rightarrow Digits $\mid \epsilon$

Digit $\rightarrow 0 \mid 1 \mid 2 \mid \dots \mid 9$

Num	Sign	Digit	Digits	More
$+$ $-$	$+$ $-$	0 5 1 6 2 7 3 8 4 9		

FIRST Sets with ϵ

Num \rightarrow Sign Digits
Sign $\rightarrow + \mid - \mid \epsilon$
Digits \rightarrow **Digit More**
More \rightarrow Digits $\mid \epsilon$
Digit $\rightarrow 0 \mid 1 \mid 2 \mid \dots \mid 9$

Num	Sign	Digit	Digits	More
+ -	+ -	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	

FIRST Sets with ϵ

Num \rightarrow **Sign Digits**
Sign \rightarrow **+** | **-** | **ϵ**
Digits \rightarrow **Digit More**
More \rightarrow **Digits** | **ϵ**
Digit \rightarrow **0** | **1** | **2** | ... | **9**

Num	Sign	Digit	Digits	More
+ -	+ -	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	

FIRST Sets with ϵ

Num \rightarrow Sign Digits
Sign $\rightarrow + \mid - \mid \epsilon$
Digits \rightarrow Digit More
More \rightarrow **Digits** $\mid \epsilon$
Digit $\rightarrow 0 \mid 1 \mid 2 \mid \dots \mid 9$

Num	Sign	Digit	Digits	More
$+$ $-$	$+$ $-$	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	

FIRST Sets with ϵ

Num \rightarrow Sign Digits
Sign $\rightarrow + \mid - \mid \epsilon$
Digits \rightarrow Digit More
More \rightarrow **Digits** $\mid \epsilon$
Digit $\rightarrow 0 \mid 1 \mid 2 \mid \dots \mid 9$

Num	Sign	Digit	Digits	More
+ -	+ -	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9

FIRST Sets with ϵ

Num \rightarrow **Sign Digits**
Sign \rightarrow **+** | **-** | **ϵ**
Digits \rightarrow **Digit More**
More \rightarrow **Digits** | **ϵ**
Digit \rightarrow **0** | **1** | **2** | ... | **9**

Num	Sign	Digit	Digits	More
+ -	+ -	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9

FIRST Sets with ϵ

Num \rightarrow **Sign Digits**
Sign \rightarrow **+** | **-** | **ϵ**
Digits \rightarrow **Digit More**
More \rightarrow **Digits** | **ϵ**
Digit \rightarrow **0** | **1** | **2** | ... | **9**

Num	Sign	Digit	Digits	More
+ -	+ - ϵ	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ϵ

FIRST Sets with ϵ

Num → **Sign Digits**
Sign → + | - | ϵ
Digits → **Digit More**
More → **Digits** | ϵ
Digit → 0 | 1 | 2 | ... | 9

Num	Sign	Digit	Digits	More
+ -	+ - ϵ	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ϵ

FIRST Sets with ϵ

Num → **Sign Digits**
Sign → + | - | ϵ
Digits → **Digit More**
More → **Digits** | ϵ
Digit → 0 | 1 | 2 | ... | 9

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

FIRST Sets with ϵ

Num \rightarrow **Sign Digits**
Sign \rightarrow **+** | **-** | **ϵ**
Digits \rightarrow **Digit More**
More \rightarrow **Digits** | **ϵ**
Digit \rightarrow **0** | **1** | **2** | ... | **9**

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

FIRST and ε

- When computing FIRST sets in a grammar with ε -productions, we often have to “look through” nonterminals.
- Rationale: Might have a derivation like this:

$$\mathbf{A} \Rightarrow \mathbf{B}t \Rightarrow t$$

- So $t \in \text{FIRST}(\mathbf{A})$.

FIRST Computation with ϵ

- Initially, for all nonterminals A , set

$$\text{FIRST}(A) = \{ t \mid A \rightarrow t\omega \text{ for some } \omega \}$$

- For all nonterminals A where $A \rightarrow \epsilon$ is a production, add ϵ to $\text{FIRST}(A)$.
- Repeat the following until no changes occur:
 - For each production $A \rightarrow \alpha$, where α is a string of nonterminals whose FIRST sets contain ϵ , set $\text{FIRST}(A) = \text{FIRST}(A) \cup \{ \epsilon \}$.
 - For each production $A \rightarrow \alpha t \omega$, where α is a string of nonterminals whose FIRST sets contain ϵ , set $\text{FIRST}(A) = \text{FIRST}(A) \cup \{ t \}$.
 - For each production $A \rightarrow \alpha B \omega$, where α is string of nonterminals whose FIRST sets contain ϵ , set $\text{FIRST}(A) = \text{FIRST}(A) \cup (\text{FIRST}(B) - \{ \epsilon \})$.

A Notational Diversion

- Once we have computed the correct FIRST sets for each nonterminal, we can generalize our definition of FIRST sets to strings.
- Define $\text{FIRST}^*(\omega)$ as follows:
 - $\text{FIRST}^*(\epsilon) = \{ \epsilon \}$
 - $\text{FIRST}^*(t\omega) = \{ t \}$
 - If $\epsilon \notin \text{FIRST}(\mathbf{A})$:
 - $\text{FIRST}^*(\mathbf{A}\omega) = \text{FIRST}(\mathbf{A})$
 - If $\epsilon \in \text{FIRST}(\mathbf{A})$:
 - $\text{FIRST}^*(\mathbf{A}\omega) = (\text{FIRST}(\mathbf{A}) - \{ \epsilon \}) \cup \text{FIRST}^*(\omega)$

FIRST Computation with ε

- Initially, for all nonterminals A , set
$$\text{FIRST}(A) = \{ t \mid A \rightarrow t\omega \text{ for some } \omega \}$$
- For all nonterminals A where $A \rightarrow \varepsilon$ is a production, add ε to $\text{FIRST}(A)$.
- Repeat the following until no changes occur:
 - For each production $A \rightarrow \alpha$, set
$$\text{FIRST}(A) = \text{FIRST}(A) \cup \text{FIRST}^*(\alpha)$$

LL(1) Tables with ε

Msg \rightarrow **Hi End**

Hi \rightarrow **hello** | **heya** | **yo**

End \rightarrow **world!** | ε

LL(1) Tables with ϵ

Msg \rightarrow **Hi End**

Hi \rightarrow **hello** | **heya** | **yo**

End \rightarrow **world!** | ϵ

	hello	heya	yo	world!
Msg				
Hi				
End				

LL(1) Tables with ϵ

Msg \rightarrow **Hi End**

Hi \rightarrow **hello** | **heya** | **yo**

End \rightarrow **world!** | ϵ

Msg	Hi	End

	hello	heya	yo	world!
Msg				
Hi				
End				

LL(1) Tables with ϵ

Msg → **Hi End**

Hi → **hello** | **heya** | **yo**

End → **world!** | ϵ

Msg	Hi	End
	hello heya yo	

	hello	heya	yo	world!
Msg				
Hi				
End				

LL(1) Tables with ϵ

Msg → **Hi End**

Hi → **hello** | **heya** | **yo**

End → **world!** | ϵ

Msg	Hi	End
	hello heya yo	world ϵ

	hello	heya	yo	world!
Msg				
Hi				
End				

LL(1) Tables with ϵ

Msg \rightarrow **Hi End**

Hi \rightarrow hello | heyA | yo

End \rightarrow world! | ϵ

Msg	Hi	End
	hello heyA yo	world ϵ

	hello	heyA	yo	world!
Msg				
Hi				
End				

LL(1) Tables with ϵ

Msg \rightarrow **Hi End**

Hi \rightarrow hello | heyA | yo

End \rightarrow world! | ϵ

Msg	Hi	End
hello heyA yo	hello heyA yo	world ϵ

	hello	heyA	yo	world!
Msg				
Hi				
End				

LL(1) Tables with ϵ

Msg \rightarrow **Hi End**

Hi \rightarrow **hello** | **heya** | **yo**

End \rightarrow **world!** | ϵ

Msg	Hi	End
hello heya yo	hello heya yo	world ϵ

	hello	heya	yo	world!
Msg				
Hi				
End				

LL(1) Tables with ϵ

Msg \rightarrow **Hi End**

Hi \rightarrow **hello** | **heya** | **yo**

End \rightarrow **world!** | ϵ

Msg	Hi	End
hello heya yo	hello heya yo	world ϵ

	hello	heya	yo	world!
Msg				
Hi				
End				

LL(1) Tables with ϵ

Msg \rightarrow **Hi End**

Hi \rightarrow **hello** | **heya** | **yo**

End \rightarrow **world!** | ϵ

Msg	Hi	End
hello heya yo	hello heya yo	world ϵ

	hello	heya	yo	world!
Msg	Hi End	Hi End	Hi End	
Hi				
End				

LL(1) Tables with ϵ

Msg \rightarrow **Hi End**

Hi \rightarrow **hello** | **heya** | **yo**

End \rightarrow **world!** | ϵ

Msg	Hi	End
hello heya yo	hello heya yo	world ϵ

	hello	heya	yo	world!
Msg	Hi End	Hi End	Hi End	
Hi				
End				

LL(1) Tables with ϵ

Msg \rightarrow **Hi End**

Hi \rightarrow **hello** | **heya** | **yo**

End \rightarrow **world!** | ϵ

Msg	Hi	End
hello heya yo	hello heya yo	world ϵ

	hello	heya	yo	world!
Msg	Hi End	Hi End	Hi End	
Hi	hello	heya	yo	
End				

LL(1) Tables with ϵ

Msg \rightarrow **Hi End**

Hi \rightarrow **hello** | **heya** | **yo**

End \rightarrow **world!** | ϵ

Msg	Hi	End
hello heya yo	hello heya yo	world ϵ

	hello	heya	yo	world!
Msg	Hi End	Hi End	Hi End	
Hi	hello	heya	yo	
End				

LL(1) Tables with ϵ

Msg \rightarrow **Hi End**

Hi \rightarrow **hello** | **heya** | **yo**

End \rightarrow **world!** | ϵ

Msg	Hi	End
hello heya yo	hello heya yo	world ϵ

	hello	heya	yo	world!
Msg	Hi End	Hi End	Hi End	
Hi	hello	heya	yo	
End				world!

LL(1) Tables with ϵ

Msg \rightarrow **Hi End**

Hi \rightarrow **hello** | **heya** | **yo**

End \rightarrow **world!** | ϵ

Msg	Hi	End
hello heya yo	hello heya yo	world ϵ

	hello	heya	yo	world!
Msg	Hi End	Hi End	Hi End	
Hi	hello	heya	yo	
End				world!

LL(1) Tables with ϵ

	hello	heya	yo	world!
Msg	Hi End	Hi End	Hi End	
Hi	hello	heya	yo	
End				world!

LL(1) Tables with ϵ

Msg \$	hello \$
--------	----------

	hello	heya	yo	world!
Msg	Hi End	Hi End	Hi End	
Hi	hello	heya	yo	
End				world!

LL(1) Tables with ϵ

Msg \$	hello \$
--------	----------

	hello	heya	yo	world!
Msg	Hi End	Hi End	Hi End	
Hi	hello	heya	yo	
End				world!

LL(1) Tables with ϵ

Msg \$	hello \$
Hi End \$	hello \$

	hello	heya	yo	world!
Msg	Hi End	Hi End	Hi End	
Hi	hello	heya	yo	
End				world!

LL(1) Tables with ϵ

Msg \$	hello \$
Hi End \$	hello \$

	hello	heya	yo	world!
Msg	Hi End	Hi End	Hi End	
Hi	hello	heya	yo	
End				world!

LL(1) Tables with ϵ

Msg \$	hello \$
Hi End \$	hello \$
hello End \$	hello \$

	hello	heya	yo	world!
Msg	Hi End	Hi End	Hi End	
Hi	hello	heya	yo	
End				world!

LL(1) Tables with ϵ

Msg \$	hello \$
Hi End \$	hello \$
hello End \$	hello \$
End \$	\$

	hello	heya	yo	world!
Msg	Hi End	Hi End	Hi End	
Hi	hello	heya	yo	
End				world!

LL(1) Tables with ϵ

Msg \$	hello \$
Hi End \$	hello \$
hello End \$	hello \$
End \$	\$



	hello			world!
Msg	Hi End			
Hi	hello	heya	yo	
End				world!

ϵ is Complicated

- When constructing LL(1) tables with ϵ -productions, we need to have an extra column for $\$$.

Msg \rightarrow **Hi End**

Hi \rightarrow **hello** | **heya** | **yo**

End \rightarrow **world!** | ϵ

ϵ is Complicated

- When constructing LL(1) tables with ϵ -productions, we need to have an extra column for $\$$.

Msg \rightarrow **Hi End**

Hi \rightarrow **hello** | **heya** | **yo**

End \rightarrow **world!** | ϵ

	hello	heya	yo	world!	\$
Msg	Hi End	Hi End	Hi End		
Hi	hello	heya	yo		
End				world!	

ϵ is Complicated

- When constructing LL(1) tables with ϵ -productions, we need to have an extra column for $\$$.

Msg \rightarrow **Hi End**

Hi \rightarrow **hello** | **heya** | **yo**

End \rightarrow **world!** | **ϵ**

	hello	heya	yo	world!	\$
Msg	Hi End	Hi End	Hi End		
Hi	hello	heya	yo		
End				world!	ϵ

LL(1) Tables with ε

LL(1) Tables with ϵ

Msg \$	hello \$
Hi End \$	hello \$
hello End \$	hello \$
End \$	\$

LL(1) Tables with ϵ

Msg \$	hello \$
Hi End \$	hello \$
hello End \$	hello \$
End \$	\$

	hello	heya	yo	world!	\$
Msg	Hi End	Hi End	Hi End		
Hi	hello	heya	yo		
End				world!	ϵ

LL(1) Tables with ϵ

Msg \$	hello \$
Hi End \$	hello \$
hello End \$	hello \$
End \$	\$

	hello	heya	yo	world!	\$
Msg	Hi End	Hi End	Hi End		
Hi	hello	heya	yo		
End				world!	ϵ

LL(1) Tables with ϵ

Msg \$	hello \$
Hi End \$	hello \$
hello End \$	hello \$
End \$	\$
\$	\$

	hello	heya	yo	world!	\$
Msg	Hi End	Hi End	Hi End		
Hi	hello	heya	yo		
End				world!	ϵ

It Gets Trickier

Num → **Sign Digits**

Sign → **+** | **-** | **ϵ**

Digits → **Digit More**

More → **Digits** | **ϵ**

Digit → **0** | **1** | ... | **9**

It Gets Trickier

Num → **Sign Digits**

Sign → **+** | **-** | **ϵ**

Digits → **Digit More**

More → **Digits** | **ϵ**

Digit → **0** | **1** | ... | **9**

	+	-	#	\$
Num				
Sign				
Digits				
More				
Digit				

It Gets Trickier

- Num** → **Sign Digits**
- Sign** → **+ | - | ϵ**
- Digits** → **Digit More**
- More** → **Digits | ϵ**
- Digit** → **0 | 1 | ... | 9**

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num				
Sign				
Digits				
More				
Digit				

It Gets Trickier

Num → **Sign Digits**

Sign → + | - | ϵ

Digits → **Digit More**

More → **Digits** | ϵ

Digit → 0 | 1 | ... | 9

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num				
Sign				
Digits				
More				
Digit				

It Gets Trickier

Num → **Sign Digits**

Sign → + | - | ϵ

Digits → **Digit More**

More → **Digits** | ϵ

Digit → 0 | 1 | ... | 9

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num	Sign Digits	Sign Digits		
Sign				
Digits				
More				
Digit				

It Gets Trickier

Num → **Sign Digits**

Sign → + | - | ϵ

Digits → **Digit More**

More → **Digits** | ϵ

Digit → 0 | 1 | ... | 9

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num	Sign Digits	Sign Digits		
Sign				
Digits				
More				
Digit				

It Gets Trickier

Num → **Sign Digits**

Sign → + | - | ϵ

Digits → **Digit More**

More → **Digits** | ϵ

Digit → 0 | 1 | ... | 9

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num	Sign Digits	Sign Digits		
Sign	+	-		
Digits				
More				
Digit				

It Gets Trickier

Num → **Sign Digits**
Sign → + | - | ϵ
Digits → **Digit More**
More → **Digits** | ϵ
Digit → 0 | 1 | ... | 9

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num	Sign Digits	Sign Digits		
Sign	+	-		
Digits				
More				
Digit				

It Gets Trickier

Num → **Sign Digits**
Sign → + | - | ϵ
Digits → **Digit More**
More → **Digits** | ϵ
Digit → 0 | 1 | ... | 9

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num	Sign Digits	Sign Digits		
Sign	+	-		
Digits			Digits More	
More				
Digit				

It Gets Trickier

Num → **Sign Digits**
Sign → **+ | - | ϵ**
Digits → **Digit More**
More → **Digits | ϵ**
Digit → **0 | 1 | ... | 9**

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num	Sign Digits	Sign Digits		
Sign	+	-		
Digits			Digits More	
More				
Digit				

It Gets Trickier

Num → **Sign Digits**
Sign → + | - | ϵ
Digits → **Digit More**
More → **Digits** | ϵ
Digit → 0 | 1 | ... | 9

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num	Sign Digits	Sign Digits		
Sign	+	-		
Digits			Digits More	
More			Digits	
Digit				

It Gets Trickier

- Num** → **Sign Digits**
- Sign** → **+ | - | ϵ**
- Digits** → **Digit More**
- More** → **Digits | ϵ**
- Digit** → **0 | 1 | ... | 9**

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num	Sign Digits	Sign Digits		
Sign	+	-		
Digits			Digits More	
More			Digits	
Digit				

It Gets Trickier

- Num** → **Sign Digits**
- Sign** → **+ | - | ϵ**
- Digits** → **Digit More**
- More** → **Digits | ϵ**
- Digit** → **0 | 1 | ... | 9**

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num	Sign Digits	Sign Digits		
Sign	+	-		
Digits			Digits More	
More			Digits	
Digit			#	

It Gets Trickier

- Num** → **Sign Digits**
- Sign** → + | - | ϵ
- Digits** → **Digit More**
- More** → **Digits** | ϵ
- Digit** → 0 | 1 | ... | 9

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num	Sign Digits	Sign Digits		
Sign	+	-		
Digits			Digits More	
More			Digits	
Digit			#	

It Gets Trickier

Num → **Sign Digits**

Sign → + | - | ϵ

Digits → **Digit More**

More → **Digits** | ϵ

Digit → 0 | 1 | ... | 9

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num	Sign Digits	Sign Digits		
Sign	+	-		
Digits			Digits More	
More			Digits	
Digit			#	

It Gets Trickier

Num → **Sign Digits**

Sign → + | - | ϵ

Digits → **Digit More**

More → **Digits** | ϵ

Digit → 0 | 1 | ... | 9

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num	Sign Digits	Sign Digits	Sign Digits	
Sign	+	-		
Digits			Digits More	
More			Digits	
Digit			#	

It Gets Trickier

Num → **Sign Digits**

Sign → + | - | ϵ

Digits → **Digit More**

More → **Digits** | ϵ

Digit → 0 | 1 | ... | 9

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num	Sign Digits	Sign Digits	Sign Digits	
Sign	+	-		
Digits			Digits More	
More			Digits	
Digit			#	

It Gets Trickier

Num → **Sign Digits**

Sign → + | - | ϵ

Digits → **Digit More**

More → **Digits** | ϵ

Digit → 0 | 1 | ... | 9

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num	Sign Digits	Sign Digits	Sign Digits	
Sign	+	-	ϵ	
Digits			Digits More	
More			Digits	
Digit			#	

It Gets Trickier

Num → **Sign Digits**
Sign → + | - | ϵ
Digits → **Digit More**
More → **Digits** | ϵ
Digit → 0 | 1 | ... | 9

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num	Sign Digits	Sign Digits	Sign Digits	
Sign	+	-	ϵ	
Digits			Digits More	
More			Digits	
Digit			#	

It Gets Trickier

Num → **Sign Digits**
Sign → + | - | ϵ
Digits → **Digit More**
More → **Digits** | ϵ
Digit → 0 | 1 | ... | 9

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num	Sign Digits	Sign Digits	Sign Digits	
Sign	+	-	ϵ	
Digits			Digits More	
More			Digits	
Digit			#	

It Gets Trickier

Num → **Sign Digits**

Sign → + | - | ϵ

Digits → **Digit More**

More → **Digits** | ϵ

Digit → 0 | 1 | ... | 9

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num	Sign Digits	Sign Digits	Sign Digits	
Sign	+	-	ϵ	
Digits			Digits More	
More			Digits	ϵ
Digit			#	

It Gets Trickier

Num → **Sign Digits**

Sign → + | - | ϵ

Digits → **Digit More**

More → **Digits** | ϵ

Digit → 0 | 1 | ... | 9

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num	Sign Digits	Sign Digits	Sign Digits	
Sign	+	-	ϵ	
Digits			Digits More	
More			Digits	ϵ
Digit			#	

It Gets Trickier

Num → **Sign Digits**
Sign → + | - | ϵ
Digits → **Digit More**
More → **Digits** | ϵ
Digit → 0 | 1 | ... | 9

Num		Sign		Digit		Digits		More	
+	-	+	-	0	5	0	5	0	5
0	5	ϵ		1	6	1	6	1	6
1	6			2	7	2	7	2	7
2	7			3	8	3	8	3	8
3	8			4	9	4	9	4	9
4	9							ϵ	

	+	-	#	\$
Num	Sign Digits	Sign Digits	Sign Digits	
Sign	+	-	ϵ	
Digits			Digits More	
More			Digits	ϵ
Digit			#	

FOLLOW Sets

- With ϵ -productions in the grammar, we may have to “look past” the current nonterminal to what can come after it.
- The **FOLLOW set** represents the set of terminals that might come after a given nonterminal.
- Formally:

$$\text{FOLLOW}(\mathbf{A}) = \{ \mathbf{t} \mid \mathbf{S} \Rightarrow^* \alpha \mathbf{A} \mathbf{t} \omega \text{ for some } \alpha, \omega \}$$

where \mathbf{S} is the start symbol of the grammar.

- Informally, every nonterminal that can ever come after \mathbf{A} in a derivation.