Parsing

Sujit Kumar Chakrabarti

IIITB

Parsing

- Process of determining if the input belongs to the language of the grammar
- Builds a *parse tree*

Parsing

Example

12:10:45

Specification: A number followed by a colon followed by a number followed by a colon followed by a number.
Regular expression: num COLON num COLON num

Parsing

Example

12:10:45
11:09:22
...

Specification: A sequence of a number followed by a colon followed by a number followed by a colon followed by a number.
Regular expression: (num COLON num COLON num)+

Parsing

Example

()
(() ())
((())) ()

Specification: A language of balanced parentheses
Regular expression: ?

Parsing

Example

()
(() ())
((())) ()

Specification: A language of balanced parentheses
Regular expression: ?

Parsing algorithm:

```
procedure BALANCED-PARENTHESES(buffer)
    level ← 0
    while buffer has more characters do
        c ← NEXTCHAR(buffer)
        if c = LPAREN then
            level ← level + 1
        else if c = RPAREN then
            level ← level − 1
        if level < 0 then
            return false
    if level = 0 then
        return true
    else
        return false
```

Parsing

Examples of structures that can't be expressed using regular expressions

```
(* ... (* ... (* ... *)*)*)
```

```
let ... in
let ... in
let in e
```

```
if(...) {
  if(...) {
    ...
  }
}
```

Context Free Grammar[1]

Example: Balanced parentheses

[1] Backus Naur Form (BNF)

Context Free Grammar[1]

Example: Balanced parentheses

$$
\begin{array}{lcl}
S & \rightarrow & \epsilon \\
S & \rightarrow & (S) \\
S & \rightarrow & S\ S
\end{array}
$$

- **Components of a grammar:** Rules/productions, terminals, non-terminals, start symbol
- **Meta-language:** language in which the grammar is written; terminals, non-terminals are grammar-symbols/tokens in the meta-language.
- **Notational variances:** `::=` instead of $\rightarrow$
- Could be rewritten as:

$$
\begin{array}{lcl}
S & \rightarrow & \epsilon \\
 & | & (S) \\
 & | & S\ S
\end{array}
$$

[1] Backus Naur Form (BNF)

Parsing

Grammar

Example: Grammar for arithmetic expressions

Parsing

Grammar

Example: Grammar for arithmetic expressions

$$
\begin{array}{lcl}
E & \rightarrow & E + E \\
E & \rightarrow & E * E \\
E & \rightarrow & (E) \\
E & \rightarrow & 0 \mid 1 \ldots \mid 9
\end{array}
$$

Grammar

Example: Grammar for function call

Grammar

Example: Grammar for function call

$$
\begin{array}{lcl}
fcall & \rightarrow & \textbf{id}\ (\ \text{arglist}\ ) \\
arglist & \rightarrow & arg* \\
arg & \rightarrow & exp \\
exp & \rightarrow & \ldots \mid \textbf{SL} \mid \ldots
\end{array}
$$

Note:

- This grammar contains multiple non-terminals.
- $arglist \rightarrow arg*$ shorthand for

$$
\begin{array}{lcl}
arglist & \rightarrow & \epsilon \\
arglist & \rightarrow & arg\ arglist
\end{array}
$$

Parsing

Derivation – Verifying $i \in L$

$((())) \ ()$

Parsing

Derivation – Verifying $i \in L$

$((()))\ ()$

- S
- $S\ S$
- $(S)(S)$
- $((S))(\epsilon)$
- $(((S)))()$
- $(((\epsilon)))()$
- $((()))()$

Parsing

Derivation

Parsing

Derivation

Example: Derivations for arithmetic expressions

- 5
- 1 + 2
- 1 + 2 * 3

Parsing

Parse Tree

```
printf("Hello World!");
```

Parsing

Parse Tree

printf("Hello World!");

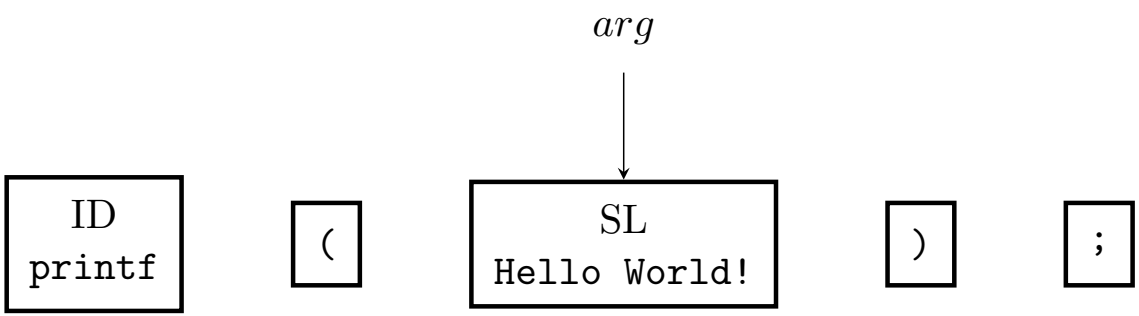
After lexical analysis:

Parsing

Parse Tree

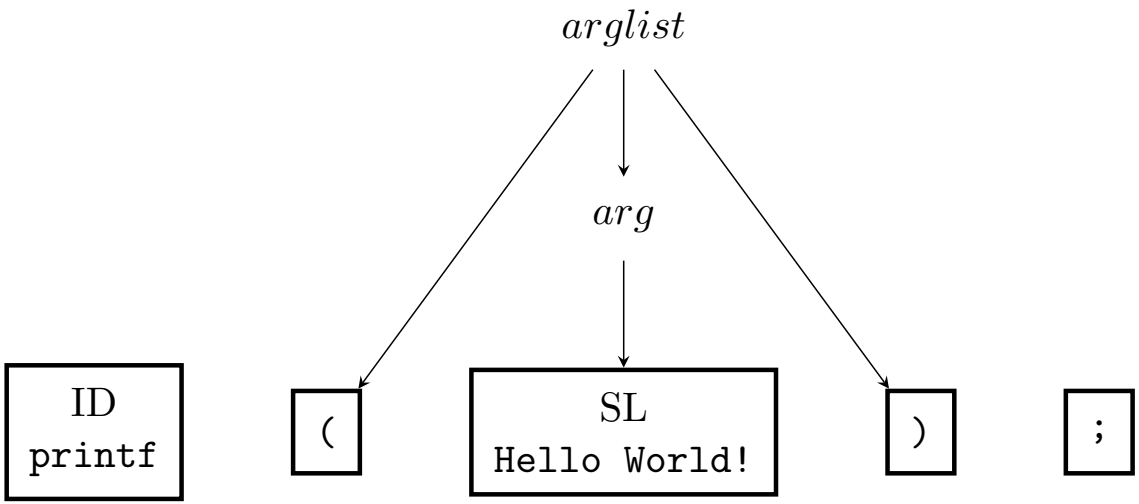
Parsing

Parse Tree



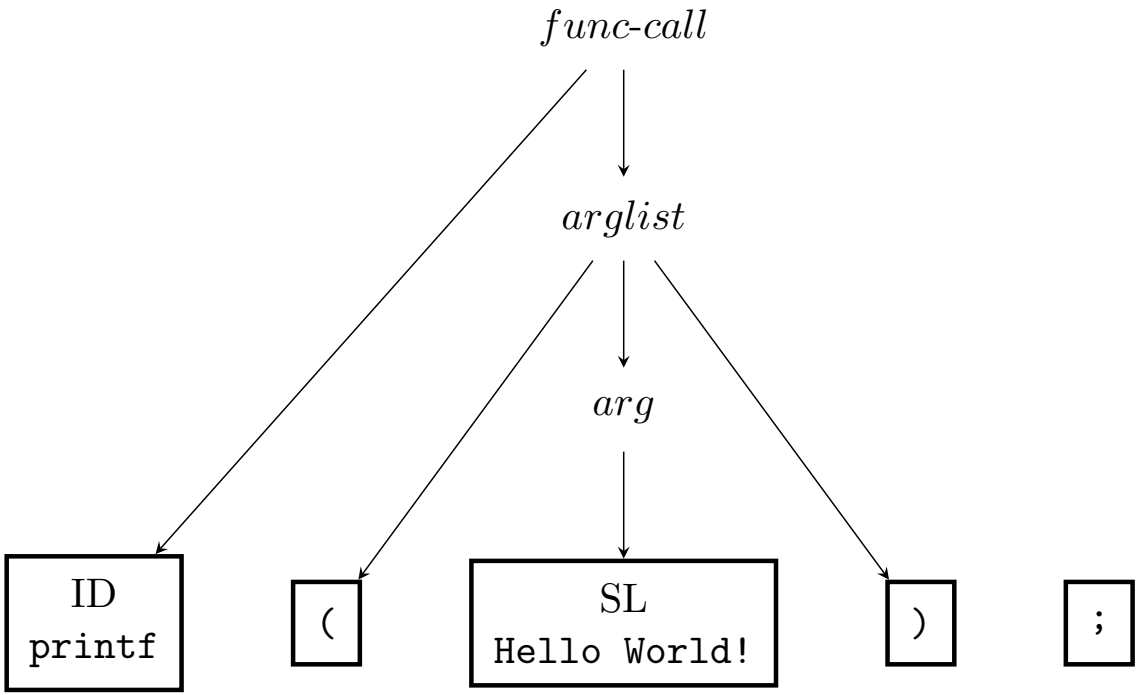
Parsing

Parse Tree



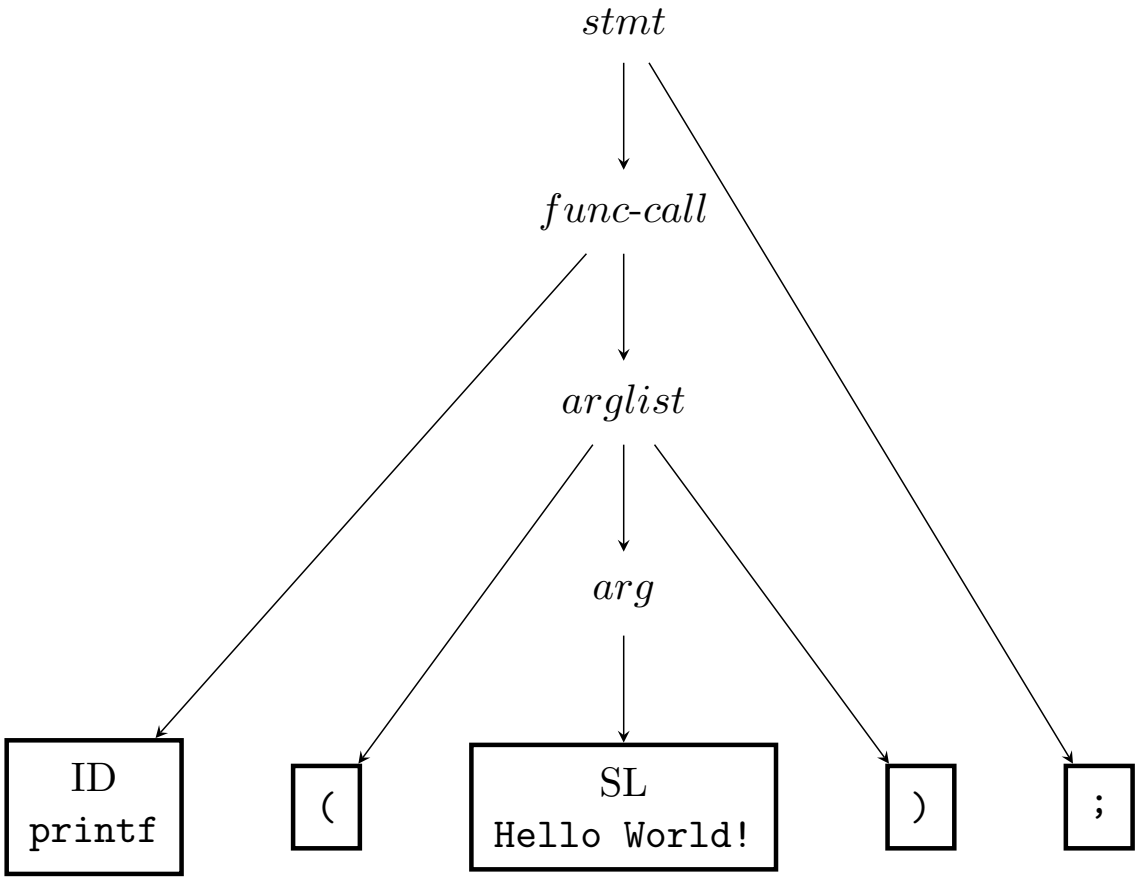
Parsing

Parse Tree



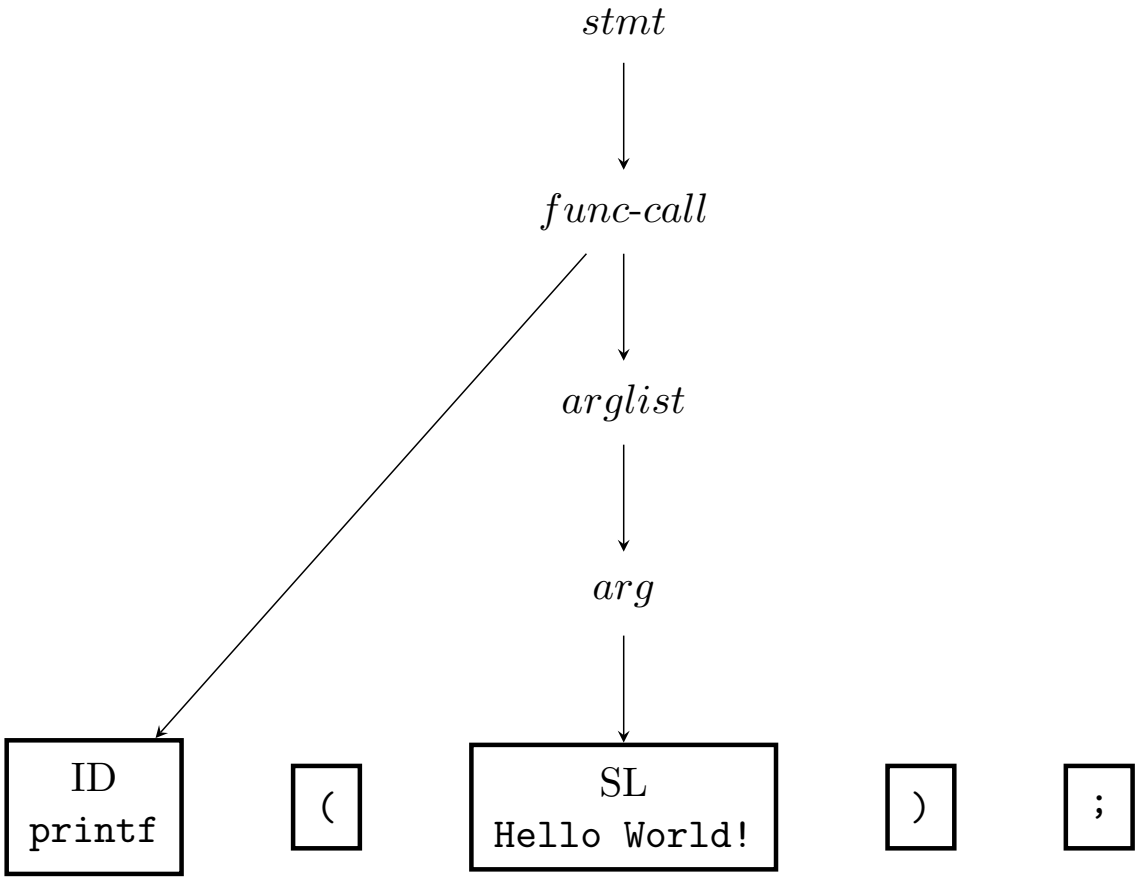
Parsing

Parse Tree



Parsing

Abstract Syntax Tree



Parsing

Parse Tree

- Grammar symbol $\mapsto$ Nodes
- Starting symbol $\mapsto$ Root node
- Non-terminals $\mapsto$ internal nodes
- Terminals $\mapsto$ leaves
- Productions $\mapsto$ Edges

Parsing

Ambiguity

$1 + 2 * 3$

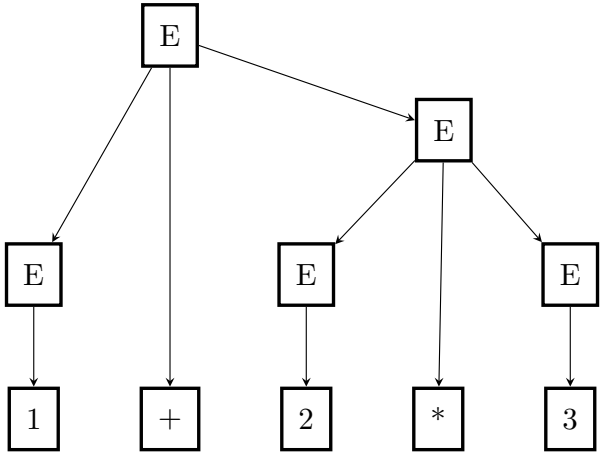
$$
\begin{array}{lcl}
E & \rightarrow & E + E \\
E & \rightarrow & E * E \\
E & \rightarrow & (E) \\
E & \rightarrow & 0 \mid 1 \ldots \mid 9
\end{array}
$$

Parsing

Ambiguity

$1 + 2 * 3$

$$
\begin{array}{lcl}
E & \rightarrow & E + E \\
E & \rightarrow & E * E \\
E & \rightarrow & (E) \\
E & \rightarrow & 0 \mid 1 \ldots \mid 9
\end{array}
$$

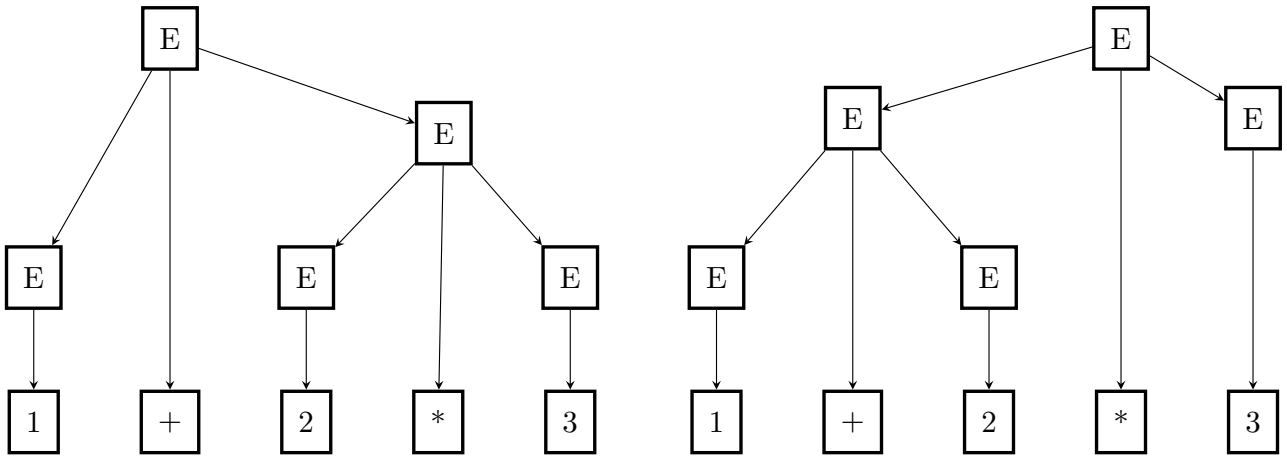


Parsing

Ambiguity

$1 + 2 * 3$

$$
\begin{array}{lcl}
E & \rightarrow & E + E \\
E & \rightarrow & E * E \\
E & \rightarrow & (E) \\
E & \rightarrow & 0 \mid 1 \ldots \mid 9
\end{array}
$$



Parsing

Ambiguity

- Language processors can't deal with ambiguity.
- Ambiguous grammars are common.
- Methods of dealing with ambiguity:
  - Fixing the grammar
  - Associativity
  - Operator precedence

Parsing

Handling Ambiguity – Fixing the grammar

Dangling else

$$
\begin{array}{lcl}
stmt & \rightarrow & \textbf{if } expr \textbf{ then } stmt \textbf{ else } stmt \\
 & | & \textbf{if } expr \textbf{ then } stmt
\end{array}
$$

```
if C1 then
  if C2 then
    S1
  else
    S2
```

Parsing

Handling Ambiguity – Fixing the grammar

Dangling else

$$
\begin{array}{lcl}
\textit{stmt} & \rightarrow & \textbf{if } \textit{expr} \textbf{ then } \textit{stmt} \textbf{ else } \textit{stmt} \\
& | & \textbf{if } \textit{expr} \textbf{ then } \textit{stmt}
\end{array}
$$

$$
\begin{array}{lcl}
\textit{stmt} & \rightarrow & \textit{matched\_stmt} \\
& | & \textit{open\_stmt} \\
\textit{matched\_stmt} & \rightarrow & \textbf{if } \textit{expr} \textbf{ then } \textit{matched\_stmt} \textbf{ else } \textit{matched\_stmt} \\
& | & \textbf{other} \\
\textit{open\_stmt} & \rightarrow & \textbf{if } \textit{expr} \textbf{ then } \textit{stmt} \\
& | & \textbf{if } \textit{expr} \textbf{ then } \textit{matched\_stmt} \textbf{ else } \textit{open\_stmt}
\end{array}
$$

Parsing

Handling Ambiguity – Operator Precedence

$1 + 2 * 3$

$$
\begin{array}{lcl}
E & \rightarrow & E + E \\
E & \rightarrow & E * E \\
E & \rightarrow & (E) \\
E & \rightarrow & 0 \mid 1 \ldots \mid 9
\end{array}
$$

Parsing

Handling Ambiguity – Operator Precedence

$1 + 2 * 3$

$$
\begin{array}{lcl}
E & \rightarrow & E + E \\
E & \rightarrow & E * E \\
E & \rightarrow & (E) \\
E & \rightarrow & 0 \mid 1 \ldots \mid 9
\end{array}
$$

* has higher precedence than +.

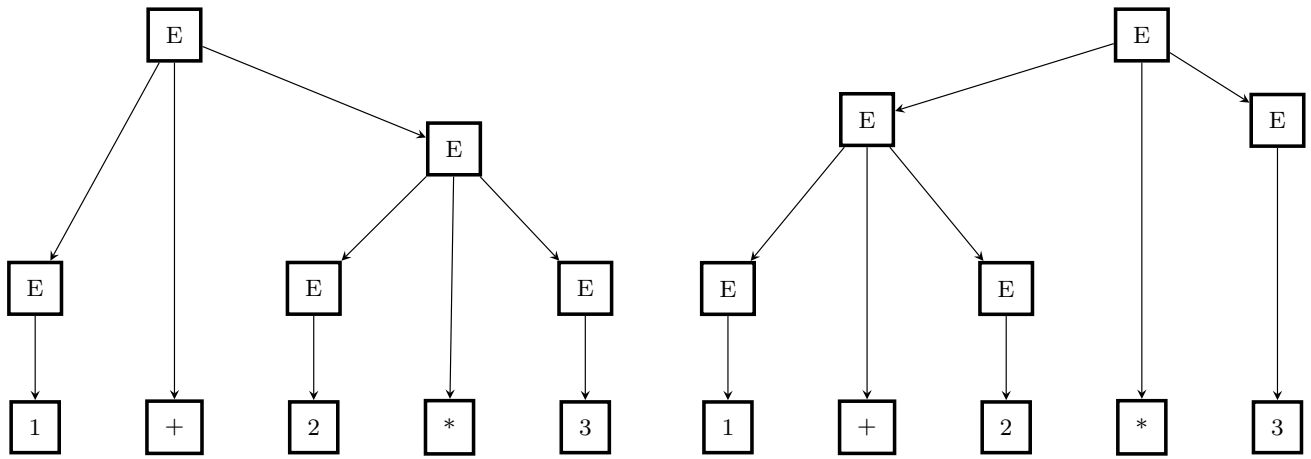
Parsing

Handling Ambiguity – Operator Precedence

$1 + 2 * 3$

$$
\begin{array}{lcl}
E & \rightarrow & E + E \\
E & \rightarrow & E * E \\
E & \rightarrow & (E) \\
E & \rightarrow & 0 \mid 1 \ldots \mid 9
\end{array}
$$

* has higher precedence than +.



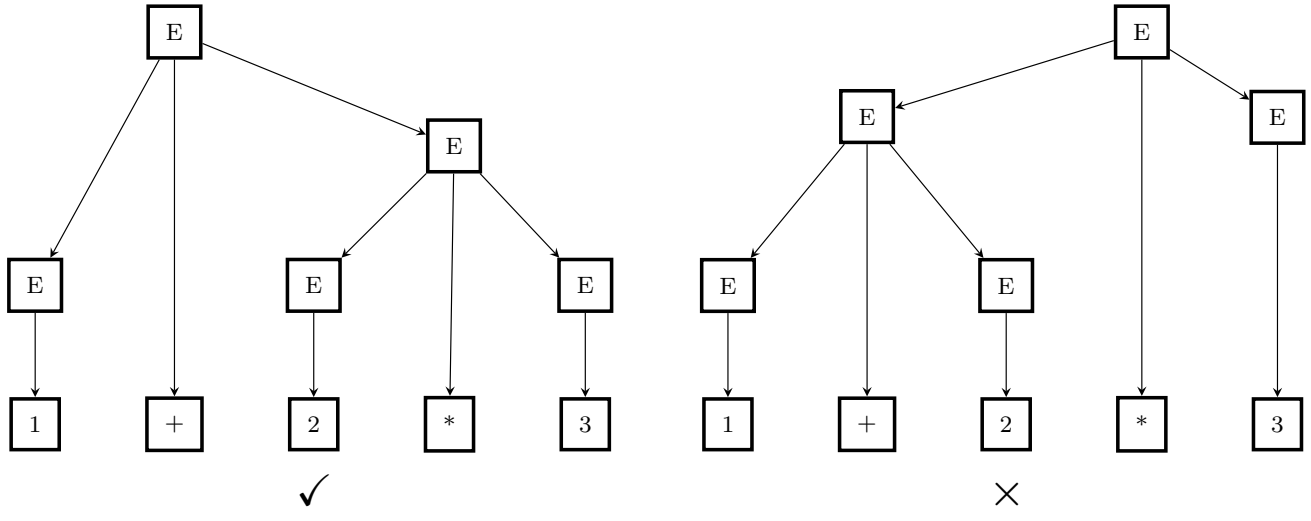
Parsing

Handling Ambiguity – Operator Precedence

$1 + 2 * 3$

$$
\begin{array}{lcl}
E & \rightarrow & E + E \\
E & \rightarrow & E * E \\
E & \rightarrow & (E) \\
E & \rightarrow & 0 \mid 1 \ldots \mid 9
\end{array}
$$

* has higher precedence than +.



Parsing

Handling Ambiguity – Associativity

4 - 2 - 1

$$
\begin{array}{lcl}
E & \rightarrow & E + E \\
E & \rightarrow & E \text{ - } E \\
E & \rightarrow & (E) \\
E & \rightarrow & 0 \mid 1 \ldots \mid 9
\end{array}
$$

Parsing

Handling Ambiguity – Associativity

4 - 2 - 1

$$
\begin{array}{lcl}
E & \rightarrow & E + E \\
E & \rightarrow & E \text{ - } E \\
E & \rightarrow & (E) \\
E & \rightarrow & 0 \mid 1 \ldots \mid 9
\end{array}
$$

- is left associative.

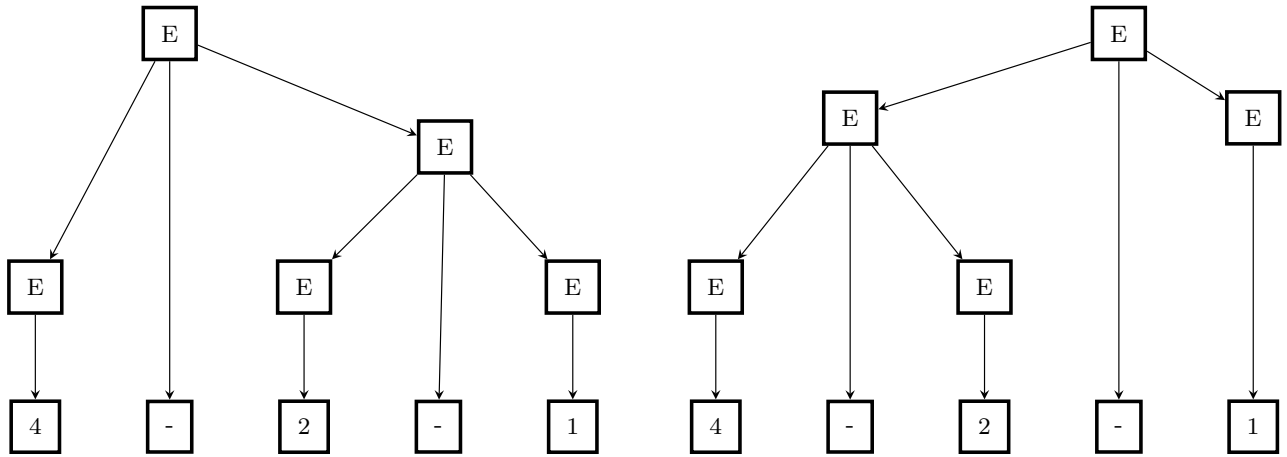
Parsing

Handling Ambiguity – Associativity

4 - 2 - 1

$$
\begin{array}{lcl}
E & \rightarrow & E + E \\
E & \rightarrow & E \text{ - } E \\
E & \rightarrow & (E) \\
E & \rightarrow & 0 \mid 1 \ldots \mid 9
\end{array}
$$

- is left associative.



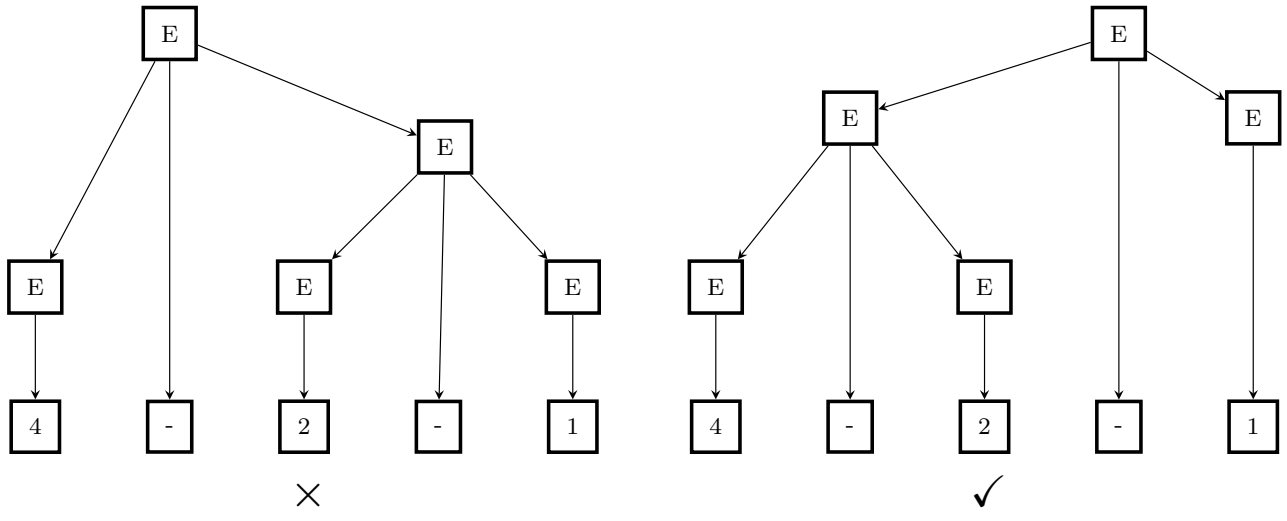
Parsing

Handling Ambiguity – Associativity

4 - 2 - 1

$$
\begin{array}{lcl}
E & \rightarrow & E + E \\
E & \rightarrow & E \text{ - } E \\
E & \rightarrow & (E) \\
E & \rightarrow & 0 \mid 1 \ldots \mid 9
\end{array}
$$

- is left associative.



Parsing

Ambiguity

- Language processors can't deal with ambiguity.
- Ambiguous grammars are common.
- Methods of dealing with ambiguity:
  - Fixing the grammar
  - Associativity
  - Operator precedence
- Non-trivial
- Not covered

Parsing

Ambiguity

- Language processors can't deal with ambiguity.
- Ambiguous grammars are common.
- Methods of dealing with ambiguity:
  - Fixing the grammar
  - Associativity
  - Operator precedence
- Non-trivial
- Not covered

Next

Recursive Descent Parsing