Introduction to Grammar

Introduction to Grammar

Every intelligible sentence must follow a grammar [1]

Grammar describes how the basic building blocks of a language can be used to construct a "valid" sentence in that language

A sentence is said to be **syntactical correct** if it follows the grammar of the language in which the sentence is written. Every programming language has a corresponding grammar/syntax.

[1] For Yoda always exceptions are there

Grammar: Informal vs. Formal

Informal Grammar: collection of rules.

- Subject and verb must agree on the number (singular or plural).
- You may not start a sentence with because because because is a conjunction.

Formal Grammar: specifies the atoms in the language and the rules of composition of these atoms to form a valid sentence in the language.

- Typically does not have any exceptions.
- Typically avoids ambiguity in the application of rules.

Formal Grammars

- Regular grammar for regular languages: Typical pattern-based searching
- Context-free grammar from context-free languages: Typical programs
- Context-sensitive grammar for context-sensitive languages: Typical counting patterns, scoping in programs
- Unrestricted grammar for recursively enumerable languages

Grammars and expressive power: COM S 331 Theory of Computing. *Chomsky Hierarchy*.

Context-Free Grammar (CFG)

A CFG contains:

- A set of terminals or atoms in the language
- A set of non-terminals
- A set of (production rules) which describes how a non-terminal can be expanded/rewritten to a sequence of terminals and non-terminals

Example CFG

- Terminals: 0 1 2 3 4 5 6 7 8 9 . + -
- Non-terminals: real-number, part, digit, sign
- Production rules:
  - A digit is a single terminal except . + and -
  - A part is a sequence of digits
  - A sign is either + or -
  - A real-number is a sign followed by a part optionally followed by a . and another part

Example CFG

- Terminals: 0 1 2 3 4 5 6 7 8 9 . + -
- Non-terminals: real-number, part, digit, sign
- Production rules:
  - A digit is a single terminal except . + and -
  - A part is a sequence of digits
  - A sign is either + or -
  - A real-number is a sign followed by a part optionally followed by a . and another part

Example

+23.1

Example CFG

- Terminals: 0 1 2 3 4 5 6 7 8 9 . + -
- Non-terminals: real-number, part, digit, sign
- Production rules:
  - A digit is a single terminal except . + and -
  - A part is a sequence of digits
  - A sign is either + or -
  - A real-number is a sign followed by a part optionally followed by a . and another part

Example

+23.1, 0..0

Example CFG

- Terminals: 0 1 2 3 4 5 6 7 8 9 . + -
- Non-terminals: real-number, part, digit, sign
- Production rules:
    - A digit is a single terminal except . + and -
    - A part is a sequence of digits
    - A sign is either + or -
    - A real-number is a sign followed by a part optionally followed by a . and another part

Example

+23.1, 0..0, 0.0.1

Example CFG

- Terminals: 0 1 2 3 4 5 6 7 8 9 . + -
- Non-terminals: real-number, part, digit, sign
- Production rules:
  - A digit is a single terminal except . + and -
  - A part is a sequence of digits
  - A sign is either + or -
  - A real-number is a sign followed by a part optionally followed by a . and another part

Example

+23.1, 0..0, 0.0.1, +-23.1

Example CFG

- Terminals: 0 1 2 3 4 5 6 7 8 9 . + -
- Non-terminals: real-number, part, digit, sign
- Production rules:
  - A digit is a single terminal except . + and -
  - A part is a sequence of digits
  - A sign is either + or -
  - A real-number is a sign followed by a part optionally followed by a . and another part

Example

+23.1, 0..0, 0.0.1, +-23.1, 23.+1

Example CFG

- Terminals: 0 1 2 3 4 5 6 7 8 9 . + -
- Non-terminals: real-number, part, digit, sign
- Production rules:
  - A digit is a single terminal except . + and -
  - A part is a sequence of digits
  - A sign is either + or -
  - A real-number is a sign followed by a part optionally followed by a . and another part

Example

+23.1, 0..0, 0.0.1, +-23.1, 23.+1, 12.

Example CFG

- Terminals: 0 1 2 3 4 5 6 7 8 9 . + -
- Non-terminals: real-number, part, digit, sign
- Production rules:
  - A digit is a single terminal except . + and -
  - A part is a sequence of digits
  - A sign is either + or -
  - A real-number is a sign followed by a part optionally followed by a . and another part

Example

+23.1, 0..0, 0.0.1, +-23.1, 23.+1, 12., 12

Formal Definition of CFG

A CFG is a tuple $G = (\Sigma, V, S, P)$, where

- Σ is a set of terminals
- V is a set of non-terminals such that $\Sigma \cap V = \emptyset$
- $S \in V$ is a start non-terminal
- P is a set of product rules, each of the form: $X \longrightarrow \omega$, such that $X \in V$ and $\omega \in (\Sigma \cup V)^+$

Formal Definition of CFG: Real number example

A CFG for real number is a tuple $G = (\Sigma, V, S, P)$, where

- $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, .\}$
- $V = \{\texttt{real-number}, \texttt{part}, \texttt{digit}, \texttt{sign}\}$
- $S = \texttt{real-number}$
- P

Formal Definition of CFG: Real number example

A CFG for real number is a tuple $G = (\Sigma, V, S, P)$, where

- $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, .\}$
- $V = \{\texttt{real-number}, \texttt{part}, \texttt{digit}, \texttt{sign}\}$
- $S = \texttt{real-number}$
- P (Backus-Naur form/BNF)

$$
\begin{array}{rcl}
\texttt{real-number} & \longrightarrow & \texttt{sign part . part} \\
 & & \mid \texttt{sign part} \\
\texttt{sign} & \longrightarrow & \texttt{+} \mid \texttt{-} \\
\texttt{part} & \longrightarrow & \texttt{digit} \mid \texttt{digit part} \\
\texttt{digit} & \longrightarrow & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{array}
$$

CFG Cont'd.

A grammar G generates a string over terminals if there exists a sequence of application of production rules starting from the the start symbol.

$$
\begin{array}{rcl}
\texttt{real-number} & \longrightarrow & \texttt{sign part . part} \\
 & & \mid \texttt{sign part} \\
\texttt{sign} & \longrightarrow & \texttt{+} \mid \texttt{-} \\
\texttt{part} & \longrightarrow & \texttt{digit} \mid \texttt{digit part} \\
\texttt{digit} & \longrightarrow & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{array}
$$

`real-number`

CFG Cont'd.

A grammar G generates a string over terminals if there exists a sequence of application of production rules starting from the the start symbol.

$$
\begin{array}{rcl}
\texttt{real-number} & \longrightarrow & \texttt{sign part . part} \\
 & & \mid \texttt{sign part} \\
\texttt{sign} & \longrightarrow & \texttt{+} \mid \texttt{-} \\
\texttt{part} & \longrightarrow & \texttt{digit} \mid \texttt{digit part} \\
\texttt{digit} & \longrightarrow & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{array}
$$

$\texttt{real-number} \rightarrow \texttt{sign part}$

CFG Cont'd.

A grammar G generates a string over terminals if there exists a sequence of application of production rules starting from the the start symbol.

$$\begin{array}{rcl} \texttt{real-number} & \longrightarrow & \texttt{sign part . part} \\ & & \mid \texttt{sign part} \\ \texttt{sign} & \longrightarrow & \texttt{+} \mid \texttt{-} \\ \texttt{part} & \longrightarrow & \texttt{digit} \mid \texttt{digit part} \\ \texttt{digit} & \longrightarrow & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{array}$$

$\texttt{real-number} \rightarrow \texttt{sign part} \rightarrow +\texttt{part}$

CFG Cont'd.

A grammar G generates a string over terminals if there exists a sequence of application of production rules starting from the the start symbol.

$$
\begin{array}{rcl}
\texttt{real-number} & \longrightarrow & \texttt{sign part . part} \\
 & & \mid \texttt{sign part} \\
\texttt{sign} & \longrightarrow & \texttt{+} \mid \texttt{-} \\
\texttt{part} & \longrightarrow & \texttt{digit} \mid \texttt{digit part} \\
\texttt{digit} & \longrightarrow & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{array}
$$

$\texttt{real-number} \rightarrow \texttt{sign part} \rightarrow +\texttt{part} \rightarrow + \texttt{digit}$

CFG Cont'd.

A grammar G generates a string over terminals if there exists a sequence of application of production rules starting from the the start symbol.

$$
\begin{array}{rcl}
\texttt{real-number} & \longrightarrow & \texttt{sign part . part} \\
 & & \mid \texttt{sign part} \\
\texttt{sign} & \longrightarrow & \texttt{+} \mid \texttt{-} \\
\texttt{part} & \longrightarrow & \texttt{digit} \mid \texttt{digit part} \\
\texttt{digit} & \longrightarrow & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{array}
$$

$\texttt{real-number} \rightarrow \texttt{sign part} \rightarrow +\texttt{part} \rightarrow + \texttt{digit} \rightarrow + 1$

CFG Cont'd.

A grammar G generates a string over terminals if there exists a sequence of application of production rules starting from the the start symbol.

$$
\begin{array}{rcl}
\texttt{real-number} & \longrightarrow & \texttt{sign part . part} \\
 & & \mid \texttt{sign part} \\
\texttt{sign} & \longrightarrow & \texttt{+} \mid \texttt{-} \\
\texttt{part} & \longrightarrow & \texttt{digit} \mid \texttt{digit part} \\
\texttt{digit} & \longrightarrow & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{array}
$$

$\texttt{real-number} \rightarrow \texttt{sign part} \rightarrow +\texttt{part} \rightarrow + \texttt{digit} \rightarrow +1$

How many strings can you generate using the above grammar?

Language

A language of a grammar G is denoted by $L(G)$, which is the set of all strings generated by G.

Exercise

Find the pattern of strings generated by the following grammars.

1. $S \longrightarrow aSb \mid ab$
2. $S \longrightarrow aSa \mid bSb \mid \epsilon$
3. $S \longrightarrow SS \mid (S) \mid ()$
4. $S \longrightarrow i\ c\ S\ t\ S \mid a$
5. $S \longrightarrow \texttt{part} \mid S + S \mid S - S$

Derivation

Every programming language has a grammar describing the valid syntax.

Compilers or interpreters use the given grammar to validate the grammatical correctness of a given program as a sequence of strings. Typical compiler errors are syntactic/grammatical errors.

Compilation Technique

A program P is a string s over term (keywords, symbols, numbers, operators, etc). P is said to be syntactically correct if and only if s can be generated/derived from the grammar G for the language in which the program is written (i.e., $s \in L(G)$).

Compilation Technique

A program P is a string s over term (keywords, symbols, numbers, operators, etc). P is said to be syntactically correct if and only if s can be generated/derived from the grammar G for the language in which the program is written (i.e., $s \in L(G)$).
The derivation of s from G is a sequence of application of production rules of the grammar.

Compilation Technique

A program P is a string s over term (keywords, symbols, numbers, operators, etc). P is said to be syntactically correct if and only if s can be generated/derived from the grammar G for the language in which the program is written (i.e., $s \in L(G)$).
The derivation of s from G is a sequence of application of production rules of the grammar.

> The derivation of $+1$ as per the grammar for real numbers is
> $\texttt{real-number} \rightarrow \texttt{sign part} \rightarrow +\texttt{part} \rightarrow + \texttt{digit} \rightarrow + 1$

Grammars and Compilers: COM S 440 Principles and Practice of Compiling

Example

$$\texttt{digit} \longrightarrow 0 \mid 1 \mid \ldots \mid 9$$
$$\texttt{part} \longrightarrow \texttt{digit} \mid \texttt{digit part}$$
$$S \longrightarrow \texttt{part} \mid S + S \mid S - S \mid S * S \mid S \,/\, S$$

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

Leftmost vs. Rightmost Derivation

- Leftmost derivation: At each derivation point, the leftmost non-terminal is expanded
- Rightmost derivation: At each derivation point, the rightmost non-terminal is expanded

Leftmost vs. Rightmost Derivation

Derivation of 1 + 2 + 3

Leftmost	**Rightmost**
$S \rightarrow S + S$	$S \rightarrow S + S$
$\rightarrow$ part + S	$\rightarrow$ S + part
$\rightarrow$ digit + S	$\rightarrow$ S + digit
$\rightarrow 1 + S$	$\rightarrow S + 3$
$\rightarrow 1 + S + S$	$\rightarrow S + S + 3$
$\rightarrow$ 1 + part + S	$\rightarrow$ S + part + 3
$\rightarrow$ 1 + digit + S	$\rightarrow$ S + digit + 3
$\rightarrow 1 + 2 + S$	$\rightarrow S + 2 + 3$
$\rightarrow$ 1 + 2 + part	$\rightarrow$ part + 2 + 3
$\rightarrow$ 1 + 2 + digit	$\rightarrow$ digit + 2 + 3
$\rightarrow 1 + 2 + 3$	$\rightarrow 1 + 2 + 3$

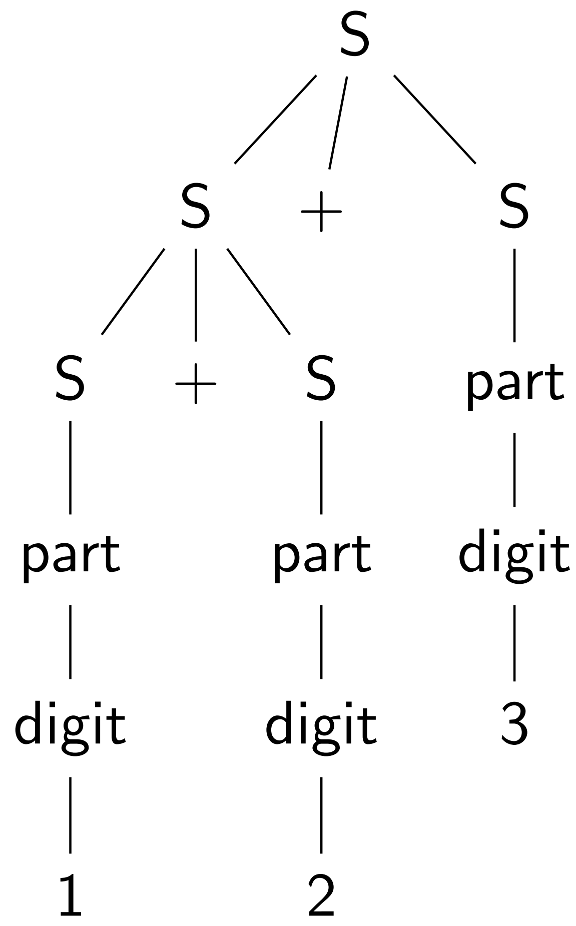
Leftmost vs. Rightmost Derivation

Parse Tree

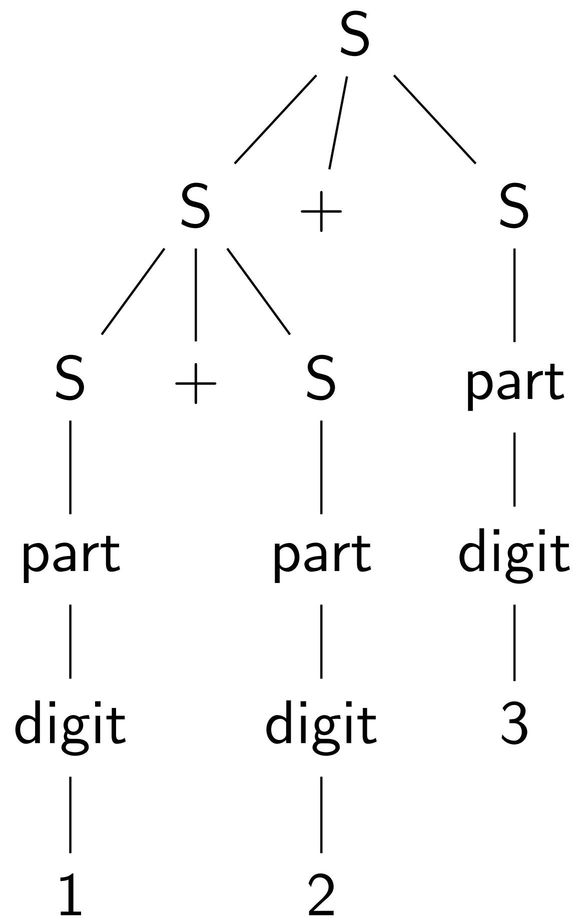
A parse tree results from the derivation sequence.

- Each node in the tree is a terminal or non-terminal in the production rule.
- Each edge in the tree from a non-terminal results from the application of production rule on the non-terminal.
- Application of production rule always result in new nodes in the tree.
- A terminal is a leaf node

Parse Tree for $1 + 2 + 3$



Parse Tree for $1 + 2 + 3$



The same parse tree can be generated by left-most and right-most derivation.

Parse Tree & Semantics

Semantics

Meaning of a syntactically correct sentence.

1. Classify the terminals into atom and operator classes.
2. Associate meaning with each atom.
3. Associate meaning with the application of operator(s) on atom(s).

Evaluate the semantics using parse tree.

Example Semantics

$\texttt{digit} \longrightarrow 0 \mid 1 \mid \ldots \mid 9$
$\texttt{part} \longrightarrow \texttt{digit} \mid \texttt{digitpart}$
$S \longrightarrow \texttt{part} \mid S + S \mid S - S \mid S * S \mid S / S$

Atoms: 0 . . . 9 and their sequences (numbers).

- Meaning of a number n can be n points.

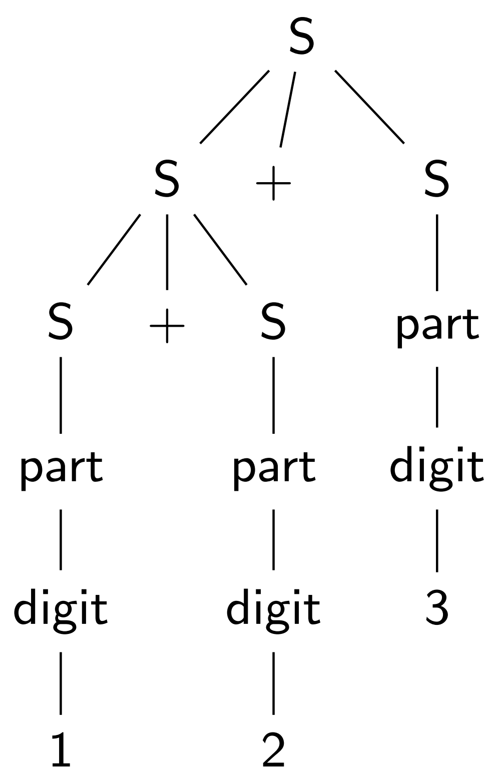
Operators:+, -, *,

- Meaning of application of + on two numbers m and n can be placing m points besides n points.

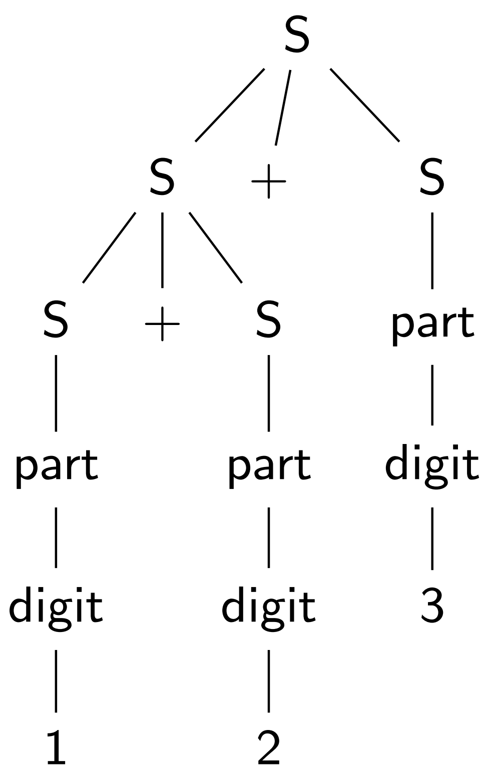
Parse Tree and Semantics

Evaluate the semantics using parse tree.

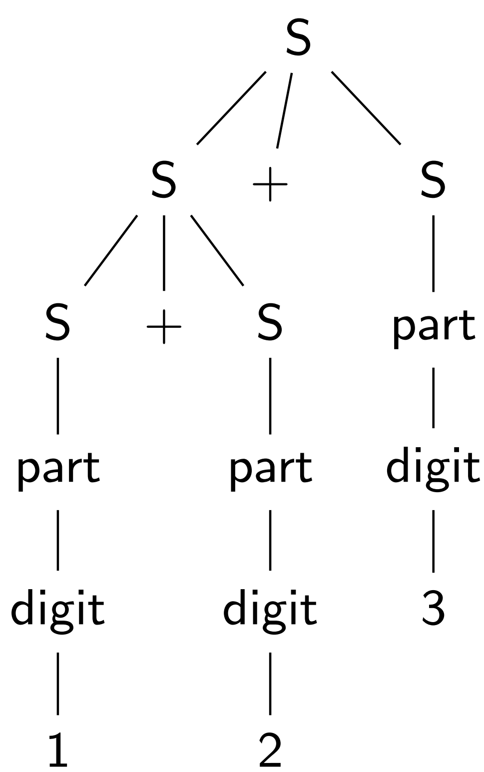
- Start from the leaf-nodes to create the atoms and find their meanings.
- Apply the operators on the generated atoms to obtain the meaning of the application of the operators.
- Continue untill you reach the root-node.



Evaluating Sematics

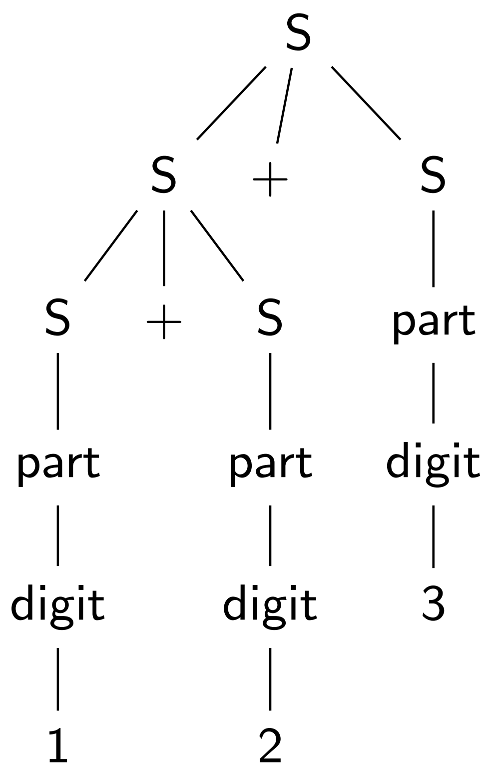


Evaluating Sematics



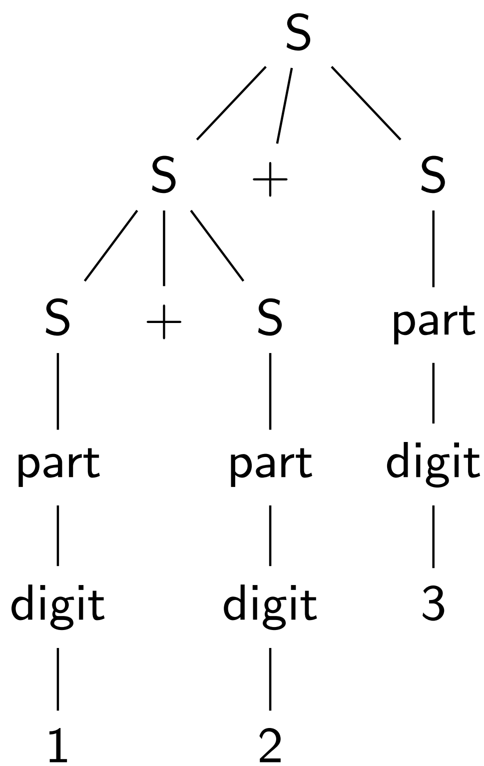
sem(1)

Evaluating Sematics



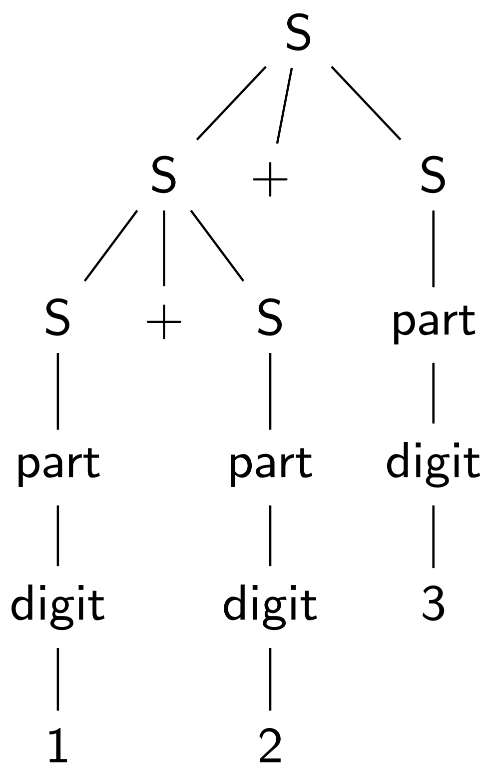
sem(1) sem(2)

Evaluating Sematics



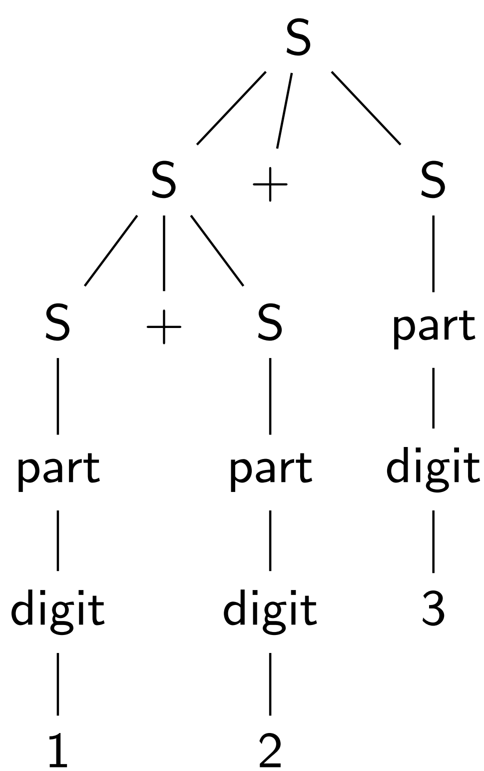
sem(1) sem(2) sem(3)

Evaluating Sematics



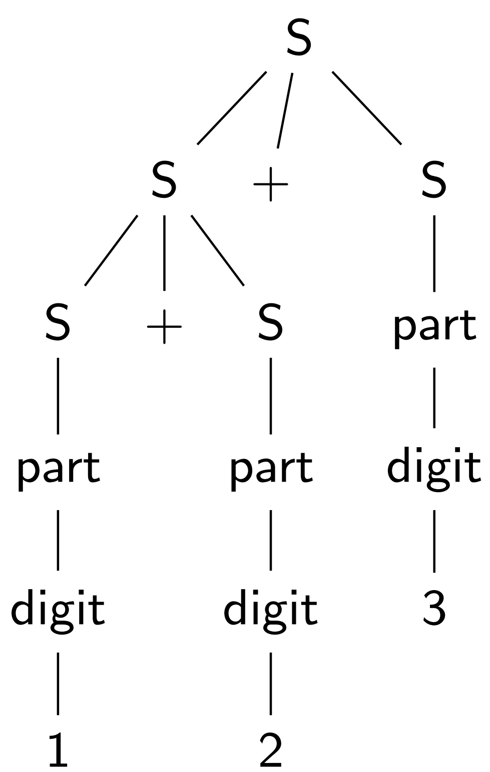
$$\text{sem}(+,\ \text{sem}(1),\ \text{sem}(2))\quad \text{sem}(3)$$

Evaluating Sematics



$$\text{sem}(+, \text{sem}(+, \text{sem}(1), \text{sem}(2)), \text{sem}(3))$$

Evaluating Sematics



sem(+, sem(+, sem(1), sem(2)), sem(3))