The structure of a language defines its syntax, but what defines semantics or meaning?

⇒ Behavior!

The most straightforward way to define semantics is to provide a simple interpreter for the programming language that highlights the behavior of the language,

⇒ Operational Semantics

Chapter 23
Let’s develop an operational semantics for a simple programming language called \textit{ONE};

\begin{align*}
\textit{ONE} : & \quad <\text{exp}>^* \ ::= \ <\text{exp}> \ + \ <\text{mulexp}> \ | \ <\text{mulexp}> \\
& \quad <\text{mulexp}> \ ::= \ <\text{mulexp}> \ * \ <\text{rootexp}> \ | \ <\text{rootexp}> \\
& \quad <\text{rootexp}> \ ::= \ (<\text{exp}> ) \ | \ <\text{constant}> \\
& \quad <\text{constant}> \ ::= \ \text{all valid integer constants}
\end{align*}

Note: The grammar is unambiguous, both precedence and associativity rules of “standard” arithmetic are observed.

Do the following sentences belong to \(L(\textit{ONE})\)? Why? Why not?

\begin{align*}
s &= 1 + 2 \times 3 \\
s &= (1 + 2) \times 3 \\
s &= a + 3
\end{align*}
Abstract Syntax Trees

We want to define an operational semantics, i.e., an abstract interpreter for the language, but parse trees are not very convenient, too many non-terminal symbols ⇒ Abstract Syntax Tree (AST)

Transformation Rules:

Note: This rule also applies to unary operators and operators with arity > 2.
Definition: An abstract syntax tree is a finite, labeled, directed tree, where the internal nodes are labeled by operators, and the leaf nodes represent the operands of the node operators. -Wikipedia, 2006

Observation: The abstract syntax tree is a simplified form of the parse tree: same order as the parse tree, but no non-terminals.
• What happens to parentheses in the AST representation of a program?
• They are **not needed**!
• ASTs naturally represent associativity and precedence relations.
• Consider: \( (1 + 2) \times 3 \)
• Parentheses do not contribute to computations, therefore the following tree transformations can be applied:
We can represent ASTs in Prolog:

\[
\begin{align*}
\text{+} & \quad \Rightarrow \quad \text{plus}(A,B) \\
\ast & \quad \Rightarrow \quad \text{times}(A,B) \\
c \quad \text{(constant)} & \quad \Rightarrow \quad \text{const}(c)
\end{align*}
\]

\[
\begin{align*}
\text{plus} & \quad \text{(const(1), times(c, const(3)))}
\end{align*}
\]
ONE: Prolog Interpreter

A simple interpreter that computes a semantic value for syntactic constructs, the computation of this semantic value can be interpreted as the behavior: val1 / 2, AST input and semantic value as output.

\[
\text{val1(plus(X,Y),Value) :-}
\text{val1(X,XValue),}
\text{val1(Y,YValue),}
\text{Value is XValue + YValue.}
\]

\[
\text{val1(times(X,Y),Value) :-}
\text{val1(X,XValue),}
\text{val1(Y,YValue),}
\text{Value is XValue * YValue.}
\]

\[
\text{val1(const(X),Value) :- Value = X.}
\]
Extend the grammar for language ONE with the subtraction operator

Extend the operational semantics appropriately, e.g.,
- 6 – 3 should give the value 3
  Assume that the abstract syntax of this operator is sub(x,y).

Compute the semantic value for the following expressions:
- sub(3,1)
- sub(4,2)
Exercises

- Extend the grammar for language ONE with the ‘!’ factorial operator
- Extend the operational semantics appropriately, e.g.,
  - 3! should give the value 6
    Assume that the abstract syntax of this operator is fact(x).
- Compute the semantic value for the following expressions:
  - fact(3)
  - fact(4)