Formal Semantics

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

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

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

Do the following sentences belong to \(L(\text{ONE})\)? 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:

```
<N>  Þ  T
T   Þ  T

<N>  Þ  op
A   B  Þ  op
```

**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:

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

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

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

\[
\text{val1(const}(X),\text{Value}) :- \text{Value = } X.
\]

?- \text{val1(const}(1),X).
X = 1
Yes

?- \text{val1(plus(const}(1),\text{const}(2)),X).
X = 3
Yes

?- \text{val1(plus(const}(1),\text{times(const}(2),\text{const}(3))),X)}.
X = 7
Yes
Exercise

- 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 `$\text{sub}(x,y)$`.
- Compute the semantic value for the following expressions:
  - $\text{sub}(3,1)$
  - $\text{sub}(4,2)$
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)