

Formal Semantics

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

⇒ Behavior!

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

⇒ Operational Semantics

Operational Semantics

Let's develop an operational semantics for a simple programming language called *ONE*;

ONE:
 $\langle \text{exp} \rangle^* ::= \langle \text{exp} \rangle + \langle \text{mulexp} \rangle \mid \langle \text{mulexp} \rangle$
 $\langle \text{mulexp} \rangle ::= \langle \text{mulexp} \rangle * \langle \text{rootexp} \rangle \mid \langle \text{rootexp} \rangle$
 $\langle \text{rootexp} \rangle ::= (\langle \text{exp} \rangle) \mid \langle \text{constant} \rangle$
 $\langle \text{constant} \rangle ::= \text{all valid integer constants}$

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?

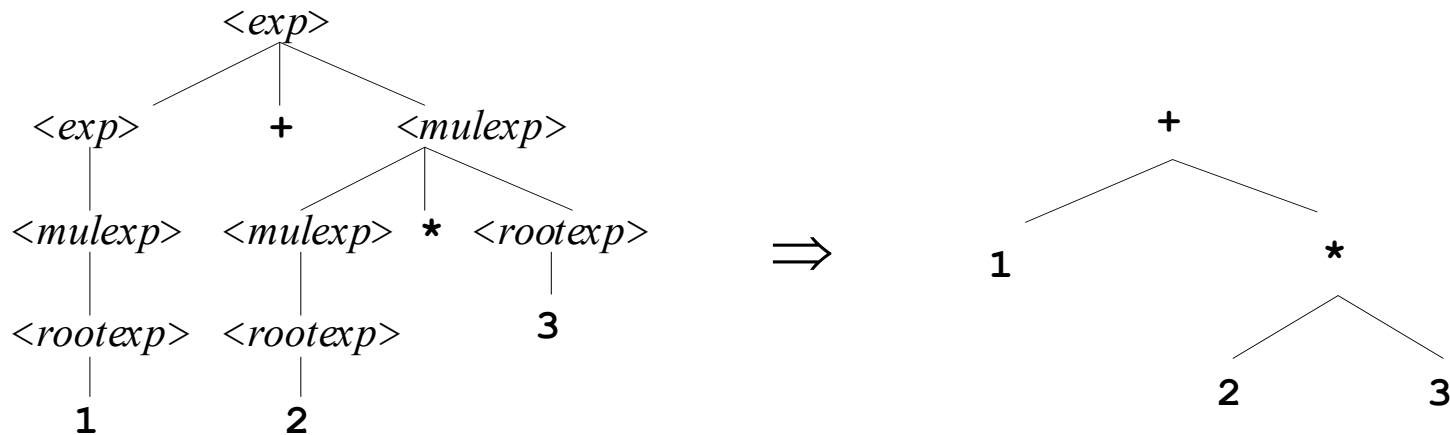
$s = 1 + 2 * 3$

$s = (1 + 2) * 3$

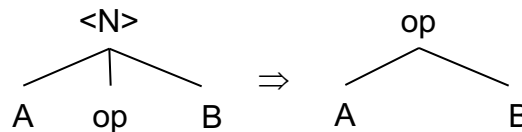
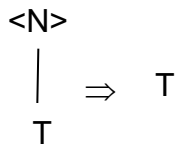
$s = a + 3$

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 \Rightarrow Abstract Syntax Tree (AST)



Transformation Rules:



Note: This rule also applies to unary operators and operators with arity > 2 .

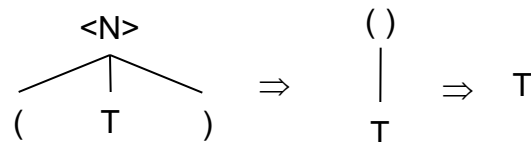
Observations

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.

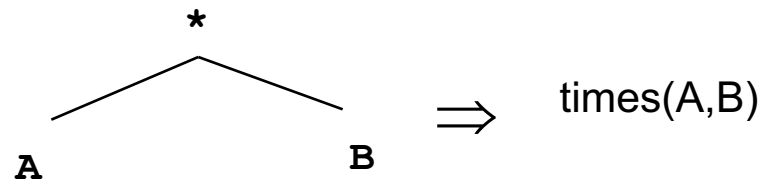
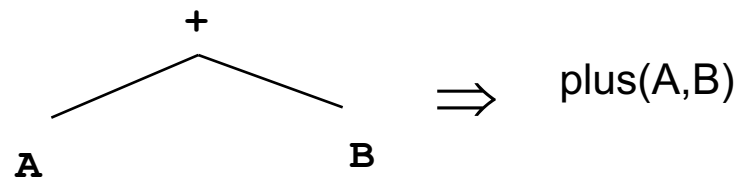
ASTs & Parentheses

- What happens to parentheses in the AST representation of a program?
- They are not needed!
- ASTs naturally represent associativity and precedence relations.
- Consider: $(1 + 2) * 3$
- Parentheses do not contribute to computations, therefore the following tree transformations can be applied:

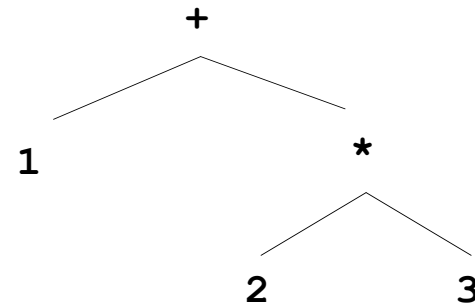


Prolog ASTs

We can represent ASTs in Prolog:



`c (constant)` \Rightarrow `const(c)`



`plus(const(1),times(const(2),const(3)))`

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: eval / 2, AST input and semantic value as output.

```
eval(plus(X,Y),Value) :-  
    eval(X,XValue),  
    eval(Y,YValue),  
    Value is XValue + YValue.
```

```
eval(times(X,Y),Value) :-  
    eval(X,XValue),  
    eval(Y,YValue),  
    Value is XValue * YValue.
```

```
eval(const(X),Value) :- Value = X.
```

```
?- eval(const(1),X).  
  
X = 1  
  
Yes  
?- eval(plus(const(1),const(2)),X).  
  
X = 3  
  
Yes  
?- eval(plus(const(1),times(const(2),const(3))),X).  
  
X = 7  
  
Yes
```

Exercises

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