

Formal Semantics

Grammars define the structure of a language, 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:
 $\langle exp \rangle^* ::= \langle exp \rangle + \langle mulexp \rangle \mid \langle mulexp \rangle$
 $\langle mulexp \rangle ::= \langle mulexp \rangle * \langle rootexp \rangle \mid \langle rootexp \rangle$
 $\langle rootexp \rangle ::= (\langle exp \rangle) \mid \langle constant \rangle$
 $\langle constant \rangle ::=$ all valid integer constants

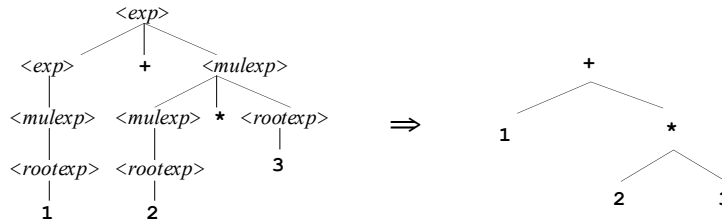
Note: The grammar is unambiguous, both precedence and associativity rules of "standard" arithmetic are observed.

Do the following sentences belong to $L(ONE)$? 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)

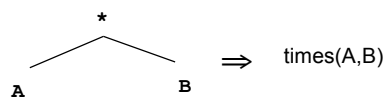
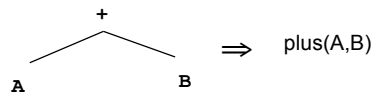


Definition: An abstract syntax tree is a finite, labeled, directed tree, where the internal nodes are labeled by operators, and the leaf nodes represent the operands of the node operators. -Wikipedia, 2006

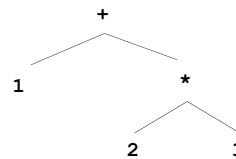
Observation: The abstract syntax tree is a simplified form: same order as the parse tree, but no non-terminals.

Prolog ASTs

We can represent ASTs in Prolog:



A (constant) \Rightarrow const(A)



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

Abstract Syntax

- Note: the set of legal Prolog ASTs can be defined by a grammar:

```
<ast> ::= plus (<ast> , <ast>)  
        | times (<ast> , <ast>)  
        | const (<constant>)
```

- This is called an *abstract syntax* for the language
- The original grammar was the *concrete syntax* of the language

ASTs & Parentheses

- What happened to parentheses in the AST representation of a program?
- They are not needed!
- ASTs naturally represent associativity and precedence relations.
- Consider: $(1 + 2) * 3$

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: val1 / 2, AST input and semantic value as output.

```
val1(plus(X,Y),Value) :-  
    val1(X,XValue),  
    val1(Y,YValue),  
    Value is XValue + YValue.
```

```
val1(times(X,Y),Value) :-  
    val1(X,XValue),  
    val1(Y,YValue),  
    Value is XValue * YValue.
```

```
val1(const(X),Value) :- Value = X.
```

```
?- val1(const(1),X).  
X = 1  
Yes  
?- val1(plus(const(1),const(2)),X).  
X = 3  
Yes  
?- val1(plus(const(1),times(const(2),const(3))),X).  
X = 7  
Yes
```