

# Abstract Syntax Tree

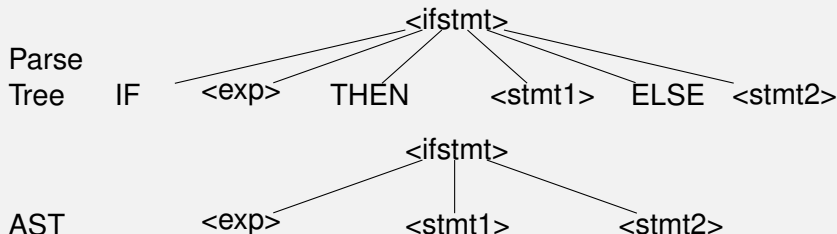
Dr. Nguyen Hua Phung  
nhphung@hcmut.edu.vn

HCMC University of Technology, Viet Nam

October 23, 2020

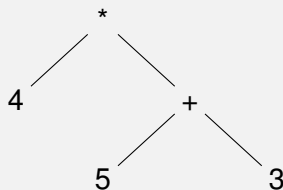
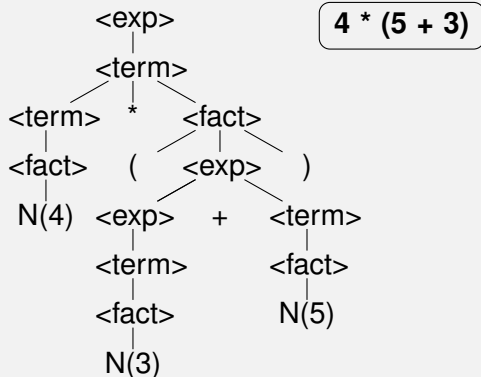
## Definition

- tree representation of the abstract syntax structure of source code.
- differ from concrete syntax tree (parse tree) by some details ignored
- help subsequence phases not depend on parse process



## Example

$\text{exp} \rightarrow \text{exp} + \text{term} \mid \text{term}$   
 $\text{term} \rightarrow \text{term} * \text{fact} \mid \text{fact}$   
 $\text{fact} \rightarrow (\text{exp}) \mid \text{ID} \mid \text{N}$



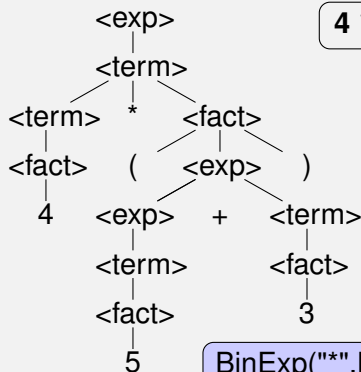
## Expression AST

```
trait Exp
```

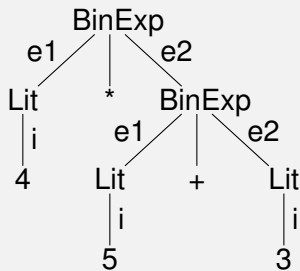
```
case class BinExp(op:String,e1:Exp,e2:Exp) extends Exp
```

```
case class UnaExp(op:String,e:Exp) extends Exp
```

```
case class Lit(i:Int) extends Exp
```



`4 * (5 + 3)`



`BinExp("...",Lit(4),BinExp("+",Lit(5),Lit(3)))`

A grammar => a class

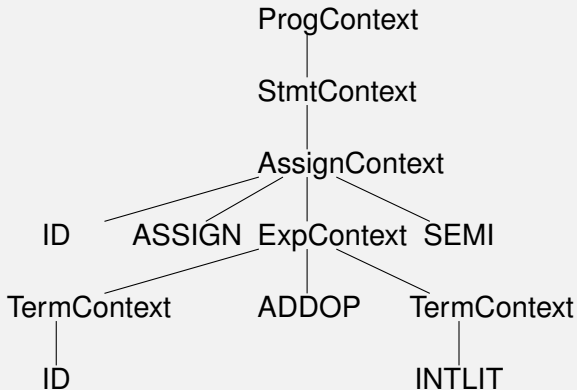
A nonterminal => an inner class

```
grammar MC;
prog  : stmt+ ;
stmt  : assign | ifstmt ;
assign: ID ASSIGN exp SEMI ;
ifstmt: IF exp THEN stmt
        ELSE stmt ;
exp
: term (ADDOP term)*;
term  : ID | INTLIT
        | LP exp RP ;
```

```
class MCParser {
    class ProgContext
    class StmtContext
    class AssignContext
    class IfstmtContext
    class ExpContext
    class TermContext
}
```

a = a + 4 ;

```
grammar MC;
prog  : stmt+ ;
stmt  : assign | ifstmt ;
assign: ID ASSIGN exp SEMI ;
ifstmt: IF exp THEN stmt
        ELSE stmt ;
exp    : term (ADDOP term)+;
term   : ID | INTLIT
        | LP exp RP ;
```



Each symbol on RHS => one or two methods in the inner class

```
class AssignContext {  
    TerminalNode ID() {...}  
    TerminalNode ASSIGN() {...}  
    ExpContext exp() {...}  
    TerminalNode SEMI() {...}  
}  
  
assign: ID ASSIGN exp SEMI ;  
  
class StmtContext {  
    AssignContext assign() {...}  
    IfstmtContext ifstmt() {...}  
}  
  
stmt: assign | ifstmt ;
```

Same symbol appears many times in RHS => 2 methods => đâsdá

```
class IfstmtContext {  
    List<StmtContext> stmt() {...}  
    StmtContext stmt(int i) {...}  
    ExpContext exp() {...}  
    TerminalNode IF () {...}  
    ...  
}
```

ifstmt: IF exp THEN stmt  
      ELSE stmt;

```
class ProgContext {  
    List<StmtContext> stmt() {...}  
    StmtContext stmt(int i) {...}  
}
```

prog: stmt+ ;



RuleContext getChild(int i):  $i^{th}$  child

```
stmt: assign          if (assign() != null)
                       //do something on assign()
    | ifstmt;         else
                       //do same thing on ifstmt()

                       //do something on getChild(0)
```

```
int getChildCount(): number of children
```

```
term : ID                if (getChildCount() == 3)
    | INTLIT             //do something on exp()
    | LP exp RP;         else
                        //do something on getChild(0)
```

- Use option -visitor
- Method **accept** generated in each inner class
- <grammar name>+Visitor.java and  
 <grammar name>+BaseVisitor.java generated  
 grammar MC; => MCVisitor.java;MCBaseVisitor.java
- Each nonterminal symbol a => visitA(ctx:AContext)

```
interface MCVisitor<T> extends ParseTreeVisitor<T>
    T visitProg (MCParser.ProgContext ctx );
    T visitStmt (MCParser.StmtContext ctx );
    T visitAssign (MCParser.AssignContext ctx );
    T visitIfstmt (MCParser.IfstmtContext ctx );
    T visitExp (MCParser.ExpContext ctx );
    T visitTerm (MCParser.TermContext ctx );
}
```

```
/* term: ID | INTLIT | LP exp RP*/
override def visitTerm(ctx:TermContext) =
  if (ctx.getChildCount() == 3)
    ctx.exp().accept(this)
  else ctx.getChild(0).getText

/* exp: term (ADDOP term)* */
override def visitExp(ctx:ExpContext) = {
  val len = ctx.ADDOP.size()
  var res = ctx.term(0).accept(this)
  for (i <- 1 to len) res = res + ' ' +
    ctx.term(i).accept(this) + ' ' +
    ctx.ADDOP(i-1).getText
  res
}
```

To distinguish RHS, naming all RHS of a nonterminal

exp: ID	#ident	visitIdent (ctx : IdentContext)
INTLIT	#lit	visitLit (ctx : LitContext)
LP exp RP	#pexp	visitPexp (ctx : PexpContext)
exp ADD exp	#addexp	visitAddexp (ctx : AddexpContext)
exp MUL exp	#mulexp	visitMulexp (ctx : MulExpContext)

There is no **visitExp** anymore

```
trait AST
case class Prog(sl:List[Stmt]) extends AST
trait Stmt extends AST
case class Assign(id:String,e:Exp) extends Stmt
case class IfStmt(e:Exp,s1:Stmt,s2:Stmt) extends Stmt
trait Exp extends AST
case class BinOp(op:String,e1:Exp,e2:Exp) extends Exp
case class Id(id:String) extends Exp
case class Intlit(lit:Int) extends Exp
```

```
/* term: ID | INTLIT | LP exp RP*/  
override def visitTerm(ctx:TermContext) =  
  if (ctx.getChildCount() == 3)  
    ctx.exp().accept(this)  
  else if (ctx.ID != null) Id(ctx.ID.getText)  
  else Intlit(ctx.INTLIT.getText.toInt)
```



```

/* exp: term (ADDOP term)* */
/* case class BinOp(op:String,e1:Exp,e2:Exp) extends Exp
override def visitExp(ctx:ExpContext) = {
  val len = ctx.ADDOP.size()
  var res = ctx.term(0).accept(this)
  for (i <- 1 to len) res = BinOp(
    ctx.ADDOP(i-1).getText,
    res.asInstanceOf[Exp],
    ctx.term(i).accept(this).asInstanceOf[Exp])
  res
}

```

3 + 4 - 5 =>

res = IntLit(3)

res = BinOp("+",IntLit(3),IntLit(4))

res = BinOp("-",BinOp("+",IntLit(3),IntLit(4)),IntLit(5))

```
/* term: ID | INTLIT | LP exp RP*/
def visitTerm(self, ctx):
    if ctx.getChildCount() == 3:
        return ctx.exp().accept(self)
    elif ctx.ID():
        return Id(ctx.ID().getText())
    else:
        return IntLit(int(ctx.INTLIT().getText()))
```

```
/* exp: term (ADDOP term)* */
/* class BinOp(Exp): #op:String, e1:Exp, e2:Exp */
def visitExp(self, ctx):
    addsize = len(ctx.ADDOP())
    res = ctx.term(0).accept(self)
    for i in range(1, addsize+1):
        res = BinOp(
            ctx.ADDOP(i-1).getText(),
            res,
            ctx.term(i).accept(self))
    return res
```

```
/* exp: term (ADDOP term)* */  
/* class BinOp(Exp): #op:String, e1:Exp, e2:Exp */  
/* 3 + 4 - 5 */  
def visitExp(self, ctx):  
    return reduce(lambda acc, ele:  
        BinOp(ele[0].getText(),  
        acc,  
        ele[1].accept(self)),  
        zip(ctx.ADDOP(), ctx.term()[1:]), # [(+,4), (-,5)  
        ctx.term(0).accept(self))
```

- AST vs. Parse tree
- AST representation in Scala
- how to build AST in Scala