Abstract Syntax Tree

Dr. Nguyen Hua Phung
phung@cse.hcmut.edu.vn

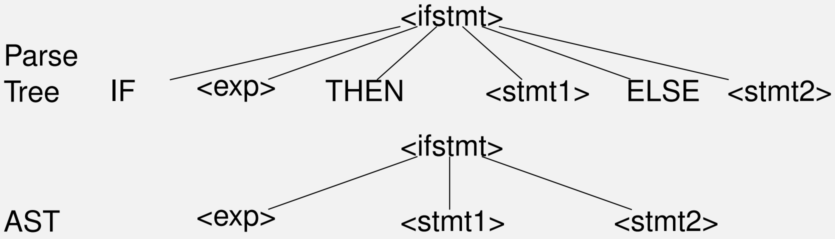
HCMC University of Technology, Viet Nam

09, 2017

Abstract Syntax Tree (AST)

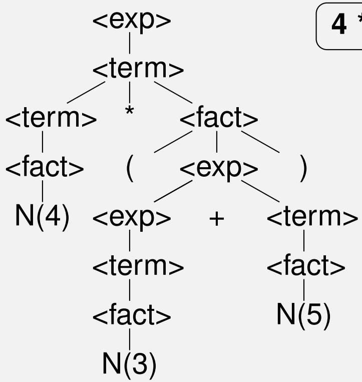
Definition

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

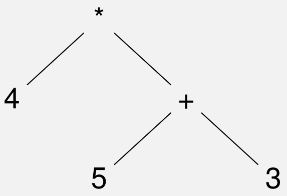


Example

exp	$\rightarrow$	exp + term | term
term	$\rightarrow$	term * fact | fact
fact	$\rightarrow$	(exp) | ID | N



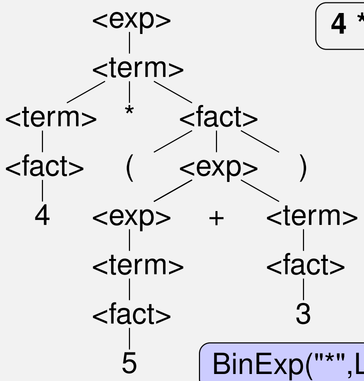
$\boxed{\textbf{4 * (5 + 3)}}$



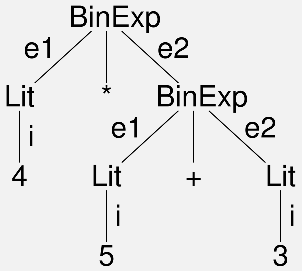
AST for an expression in Scala

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)))
```

Generated Classes by ANTLR

A grammar => a class

A nonterminal => an inner class

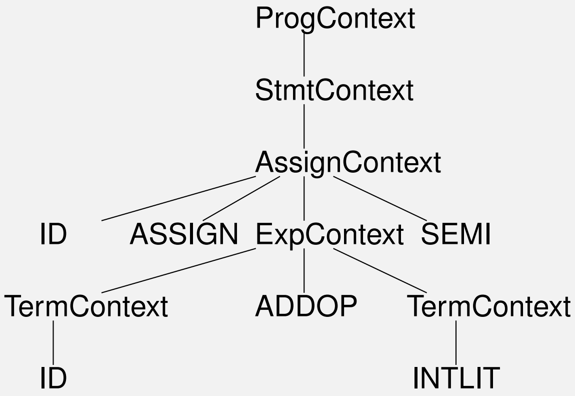
```
grammar MC;
prog   : stmt+ ;
stmt   : assign | ifstmt ;
assign : ID ASSIGN exp SEMI ;
ifstmt : IF exp THEN stmt
         ELSE stmt ;
exp    : term (ADDOP term)*;
term   : ID | INTLIT
       | LP exp RP ;
```

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

Example

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 ;
```



Access Children on Parse Tree

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

```
assign: ID ASSIGN exp SEMI ;     class AssignContext {
                                     TerminalNode ID() {...}
                                     TerminalNode ASSIGN() {...}
                                     ExpContext exp() {...}
                                     TerminalNode SEMI() {...}

stmt: assign | ifstmt ;          class StmtContext {
                                     AssignContext assign() {...}
                                     IfsmtContext ifstmt() {...}
```

Multiple Appearance of Symbol in RHS

Same symbol appears many times in RHS => 2 methods

```
ifstmt: IF exp THEN stmt
        ELSE stmt;
```

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

```
prog: stmt+ ;
```

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

Some useful methods

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

```
stmt: assign
    | ifstmt;
```

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

//do something on getChild(0)

Some useful methods

int getChildCount(): number of children

```
term: ID
    | INTLIT
    | LP exp RP;
```

```
if (getChildCount() == 3)
    //do something on exp()
else
    //do something on getChild(0)
```

Visitor Generated by ANTLR

- 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)

Example: MCVisitor

```
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);
}
```

Example: In2Pos in Scala

```
/* 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
}
```

Naming Right Hand Size

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

AST for MC

```
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
```

AST Generation Visitor

```
/* 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.INT.getText.toInt)
```

AST Generation Visitor (cont)

```
/* 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 = BinOp(
      ctx.ADDOP(i-1).getText,
      res.asInstanceOf[Exp],
      ctx.term(i).accept(this).asInstanceOf[Exp])
  res
}
```

Generate AST in Scala

Recognizer

```
def fact: Parser[Any] = wholeNumber | "(" ~ exp ~ ")"
```

Parser

```
def fact: Parser[Exp] =
      wholeNumber ^^ {case x => Lit(Integer.parseInt(x))}
    | "(" ~> exp <~ ")"
```

12 => Lit(12)
(120) => Lit(120)

Generate AST in Scala (cont'd)

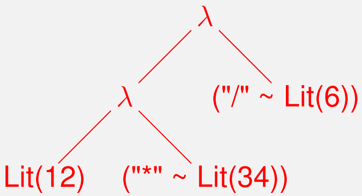
Recognizer

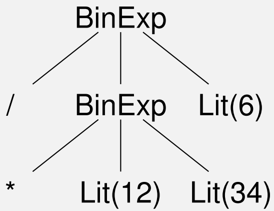
```
def term: Parser[Any] = fact ~ rep(("*"|"/") ~  fact)
```

Parser

```
def term: Parser[Exp]= fact ~ rep(("*"|"/") ~  fact) ^^ {
case a ~ il => il.foldLeft(a)((b,x)=> x match {
                                        case c~d =>BinExp(c,b,d) })
```

12 * 34 / 6 => Lit(12) ~ [("*" ~ Lit(34)); ("/" ~ Lit(6))]





BinExp("/",BinExp("*",Lit(12),Lit(34)),Lit(6))

Summary

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