Trabalho 2 - Analisador sintatico/semantico

Nathan Reuter Godinho
Ad Nunes Ribeiro
Eduardo Dias

Instruções para Rodar o Código:

1 - Entre na pasta **t1comp-master/**
2 **-** Rode o comando no terminal para compilar: $ **make**
3 - Rode o comando para executar o programa: $ **make run**

Conforme solicitado foram produzidos três algoritmos de cem linhas na gramática. Como a aplicação apresenta interface gráfica e a opção se selecionar as entradas no sistema de arquivos, para verificar os programas é apenas necessário usar o menu superior(no caminho arquivo->abrir) da aplicação para abrir os exemplos(que se encontram em src/testcodes/). Após abrir o arquivo especifico então deve-se ir no menu superior(no caminho analisar->Programa fonte) para executar a análise.

LL1
Para assegurar que a gramática está em LL1 é necessário assegurar três condições:
Dado A -> B | C ->* t
  1.first(B) ∩ first(C) = ∅
  2.B ->* '' implica !(C ->* '')
  3.B ->* '' implica first(B) ∩ follow(A) = ∅

Para exemplificar foram adicionadas algumas validações abaixo das produções, o mesmo foi feito para toda a gramática de modo a gerar a tabela de transições onde para cada cabeça de produção com uma entrada a escolha é única(determinística).

Gramatica LL1 ✔

PROGRAM -> **CLASSLIST**
Produção única

CLASSLIST -> **CLASSDECL CLASSLIST1**
Produção única

CLASSLIST1 -> ''
CLASSLIST1 -> **CLASSLIST**
first(‘’) = {‘’}
first(**CLASSLIST**) = {class}
follow(**CLASSLIST1**) = {$,},int,string,ident,constructor}

first(‘’) ∩ first(**CLASSLIST**) = {}
‘’ ->* '' porém **CLASSLIST** ->* class != ''
‘’ ->* '' e first(‘’) ∩ follow(**CLASSLIST1**) = {}

São satisfeitas as 3 condições

CLASSDECL -> *class ident* **CLASSDECL1**
Produção única

CLASSDECL1 -> **CLASSBODY**
CLASSDECL1 -> *extends ident* **CLASSBODY**
first(*extends ident* **CLASSBODY**) = {*extends*}
first(**CLASSBODY**) = {{}

follow(**CLASSDECL1**) = {$,class,},int,string,ident,constructor}

first(*extends*) ∩ first(**CLASSLIST**) = {}
Não existe produção para ‘’
Não existe produção para ‘’

São satisfeitas as 3 condições

CLASSBODY -> { **CLASSBODY1**
Produção única

CLASSBODY1 -> **PROGRAM CLASSBODY2**
CLASSBODY1 -> **CLASSBODY2**
first(**PROGRAM CLASSBODY2**) = {class}
first(**CLASSBODY2**) = {},int,string,ident,constructor}

first(**PROGRAM CLASSBODY2**) ∩ first(**CLASSBODY2**) = {}
Não existe produção para ‘’
Não existe produção para ‘’

São satisfeitas as 3 condições

CLASSBODY2 -> **VARDECLTYPE CLASSBODY22**
CLASSBODY2 -> **CLASSBODYCONSTRUCTDECL CLASSBODY4**
CLASSBODY2 -> }

first(}) = {}}
first(**VARDECLTYPE**) = {int,string,ident}
first(**CLASSBODYCONSTRUCTDECL**) = {constructor}

first(}) ∩ first(**VARDECLTYPE**) = {}
first(}) ∩ first(**CLASSBODYCONSTRUCTDECL**) = {}
first(**VARDECLTYPE**) ∩ first(**CLASSBODYCONSTRUCTDECL**) = {}

Não existe produção para ‘’
Não existe produção para ‘’

São satisfeitas as 3 condições

CLASSBODY22 -> ident CLASSBODY23
CLASSBODY22 -> VARDECLBRACKETS ident METHODBODY
CLASSBODYMETHODDECL1 }
first(ident) = {ident}
first(**VARDECLBRACKETS**) = {[}

first(ident) ∩ first(**VARDECLBRACKETS**) = {}
Não existe produção para ‘’
Não existe produção para ‘’

São satisfeitas as 3 condições

CLASSBODY23 -> METHODBODY CLASSBODYMETHODDECL1 }
CLASSBODY23 -> VARDECL1 ; CLASSBODY2
first(**METHODBODY**) = {(}
first(**VARDECL1 ; CLASSBODY2**) = {[,;,,}

first(**METHODBODY**) ∩ first(**VARDECL1**) = {}
Não existe produção para ‘’
Não existe produção para ‘’

São satisfeitas as 3 condições

CLASSBODY4 -> CLASSBODYMETHODDECL }
CLASSBODY4 -> }
first(**CLASSBODYMETHODDECL**) = {int,string,ident}
first(}) = {}}

first(**CLASSBODYMETHODDECL**) ∩ first(}) = {}
Não existe produção para ‘’
Não existe produção para ‘’

São satisfeitas as 3 condições

CLASSBODYVARDELC -> **VARDECL** ; **CLASSBODYVARDELC1**
Produção única

CLASSBODYVARDELC1 -> **CLASSBODYVARDELC**
CLASSBODYVARDELC1 -> ''
CLASSBODYCONSTRUCTDECL -> **CONSTRUCTDECL**
CLASSBODYCONSTRUCTDECL1
CLASSBODYCONSTRUCTDECL1 -> **CLASSBODYCONSTRUCTDECL**
CLASSBODYCONSTRUCTDECL1 -> ''
CLASSBODYMETHODDECL -> **METHODDECL CLASSBODYMETHODDECL1**
CLASSBODYMETHODDECL1 -> **CLASSBODYMETHODDECL**
CLASSBODYMETHODDECL1 -> ''
VARDECL -> **VARDECLTYPE** ident **VARDECL1**
VARDECL1 -> **VARDECLBRACKETS VARDECL2**
VARDECL1 -> **VARDECL2**
VARDECL2 -> **VARDECLWITHCOMA**
VARDECL2 -> ''
VARDECLTYPE -> int
VARDECLTYPE -> string
VARDECLTYPE -> ident
VARDECLBRACKETS -> [ ] **VARDECLBRACKETS1**
VARDECLBRACKETS1 -> ''
VARDECLBRACKETS1 -> **VARDECLBRACKETS**
VARDECLWITHCOMA -> , ident **VARDECLWITHCOMA1**
VARDECLWITHCOMA1 -> **VARDECLBRACKETS VARDECL2**
VARDECLWITHCOMA1 -> **VARDECLWITHCOMA**

```
VARDECLWITHCOMA1 -> ''
CONSTRUCTDECL -> constructor METHODBODY
METHODDECL -> VARDECLTYPE METHODDECL1
METHODDECL1 -> ident METHODBODY
METHODDECL1 -> VARDECLBRACKETS ident METHODBODY
METHODBODY -> ( METHODBODY1
METHODBODY1 -> PARAMLIST ) STATEMENT
METHODBODY1 -> ) STATEMENT
PARAMLIST -> VARDECLTYPE ident PARAMLIST12
PARAMLIST12 -> VARDECLBRACKETS PARAMLIST13
PARAMLIST12 -> PARAMLIST2
PARAMLIST12 -> ''
PARAMLIST13 -> PARAMLIST2
PARAMLIST13 -> ''
PARAMLIST2 -> , VARDECLTYPE ident PARAMLIST12
STATEMENT -> PRINTSTAT ;
STATEMENT -> READSTAT ;
STATEMENT -> RETURNSTAT ;
STATEMENT -> SUPERSTAT ;
STATEMENT -> IFSTAT
STATEMENT -> FORSTAT
STATEMENT -> { STATLIST }
STATEMENT -> break ;
STATEMENT -> ;
STATEMENT -> int ident VARDECL1 ;
STATEMENT -> string ident VARDECL1 ;
STATEMENT -> ident STATEMENT1
STATEMENT1 -> ident VARDECL1 ;
STATEMENT1 -> = ATRIBSTAT1 ;
STATEMENT1 -> LVALUEEXPLIST = ATRIBSTAT1 ;
ATRIBSTAT -> ident LVALUE1 = ATRIBSTAT1
ATRIBSTAT1 -> ALOCEXPRESSION
ATRIBSTAT1 -> EXPRESSION
```

```
PRINTSTAT -> print EXPRESSION
READSTAT -> read LVALUE
RETURNSTAT -> return RETURNSTAT1
RETURNSTAT1 -> EXPRESSION
RETURNSTAT1 -> ''
SUPERSTAT -> super ( SUPERSTAT1
SUPERSTAT1 -> )
SUPERSTAT1 -> ARGLIST )
IFSTAT -> if ( EXPRESSION ) STATEMENT IFSTAT1
IFSTAT1 -> else STATEMENT end
IFSTAT1 -> end
FORSTAT -> for ( FORSTAT1
FORSTAT1 ->ATRIBSTAT ; FORSTAT2
FORSTAT1 -> ; FORSTAT2
FORSTAT2 -> EXPRESSION ; FORSTAT3
FORSTAT2 -> ; FORSTAT3
FORSTAT3 -> ATRIBSTAT ) STATEMENT
FORSTAT3 -> ) STATEMENT
STATLIST -> STATEMENT STATLIST1
STATLIST1 ->  STATLIST
STATLIST1 ->  ''
LVALUE -> ident LVALUE1
LVALUE1 -> LVALUEEXPLIST
LVALUE1 -> ''
LVALUEEXPLIST -> [ EXPRESSION ] LVALUEEXPLIST1
LVALUEEXPLIST -> . ident LVALUEEXPLIST2
LVALUEEXPLIST1 -> LVALUEEXPLIST
LVALUEEXPLIST1 -> ''
LVALUEEXPLIST2 -> ( LVALUEEXPLIST3
LVALUEEXPLIST2 -> LVALUEEXPLIST
LVALUEEXPLIST2 -> ''
LVALUEEXPLIST3 -> ARGLIST ) LVALUEEXPLIST1
LVALUEEXPLIST3 -> ) LVALUEEXPLIST1
```

```
ALOCEXPRESSION -> new ALOCEXPRESSION1
ALOCEXPRESSION1 -> ident ALOCEXPRESSION2
ALOCEXPRESSION1 -> int ALOCEXPRESSIONPLUS
ALOCEXPRESSION1 -> string ALOCEXPRESSIONPLUS
ALOCEXPRESSION2 -> ( ALOCEXPRESSION3
ALOCEXPRESSION2 -> ALOCEXPRESSIONPLUS
ALOCEXPRESSION3 -> ARGLIST )
ALOCEXPRESSION3 -> )
ALOCEXPRESSIONPLUS -> [ EXPRESSION ]
ALOCEXPRESSIONPLUS1
ALOCEXPRESSIONPLUS1 -> ALOCEXPRESSIONPLUS
ALOCEXPRESSIONPLUS1 -> ''
EXPRESSION -> NUMEXPRESSION EXPRESSION1
EXPRESSION1 -> EXPRESSIONCOMPARE NUMEXPRESSION
EXPRESSION1 -> ''
EXPRESSIONCOMPARE -> !=
EXPRESSIONCOMPARE -> ==
EXPRESSIONCOMPARE -> >=
EXPRESSIONCOMPARE -> <=
EXPRESSIONCOMPARE -> >
EXPRESSIONCOMPARE -> <
NUMEXPRESSION -> TERM NUMEXPRESSION1
NUMEXPRESSION1 -> SUMMINUS NUMEXPRESSION
NUMEXPRESSION1 -> ''
SUMMINUS -> +
SUMMINUS -> -
TERM -> UNARYEXPR TERM3
TERM2 -> MULDIVMOD UNARYEXPR TERM3
TERM3 -> TERM2
TERM3 -> ''
MULDIVMOD -> %
MULDIVMOD -> /
MULDIVMOD -> *
```

UNARYEXPR -> **SUMMINUS FACTOR**
UNARYEXPR -> **FACTOR**
FACTOR -> int-constant
FACTOR -> string-constant
FACTOR -> null
FACTOR -> **LVALUE**
FACTOR -> (**EXPRESSION**)
ARGLIST -> **EXPRESSION ARGLIST2**
ARGLIST2 -> ARGLISTEXP
ARGLIST2 -> ''
ARGLISTEXP -> , **EXPRESSION ARGLIST2**

Quanto às regras semânticas, para se adequar às limitações impostas a gramática anterior foi alterada (as alterações foram destacadas) para se adequar às exigências e sobre a gramática atual foram construídas as SDD usando tanto valores sintetizados quanto herdados, sendo que os valores herdados só podem herdados pelos parentes próximos(pai e irmãos) seguindo a definição aceita para representar uma L-Atribuída.

Regras Semânticas

PROGRAM -> **CLASSLIST**
CLASSLIST -> **CLASSDECL CLASSLIST1**
CLASSLIST1 -> ''
CLASSLIST1 -> **CLASSLIST**
CLASSDECL -> *class ident* **CLASSDECL1**

```
CLASSDECL1 -> CLASSBODY
CLASSDECL1 -> extends ident CLASSBODY
CLASSBODY -> { CLASSBODY1
CLASSBODY1 -> PROGRAM CLASSBODY2
CLASSBODY1 -> CLASSBODY2
CLASSBODY2 -> VARDECLTYPE CLASSBODY22
CLASSBODY2 -> CLASSBODYCONSTRUCTDECL CLASSBODY4
CLASSBODY2 -> }
CLASSBODY22 -> ident CLASSBODY23
CLASSBODY22 -> VARDECLBRACKETS ident METHODBODY
CLASSBODYMETHODDECL1 }
CLASSBODY23 -> METHODBODY CLASSBODYMETHODDECL1 }
CLASSBODY23 -> VARDECL1 ; CLASSBODY2
CLASSBODY4 -> CLASSBODYMETHODDECL }
CLASSBODY4 -> }
CLASSBODYVARDELC -> VARDECL ; CLASSBODYVARDELC1
CLASSBODYVARDELC1 -> CLASSBODYVARDELC
CLASSBODYVARDELC1 -> ''
CLASSBODYCONSTRUCTDECL -> CONSTRUCTDECL
CLASSBODYCONSTRUCTDECL1
CLASSBODYCONSTRUCTDECL1 -> CLASSBODYCONSTRUCTDECL
CLASSBODYCONSTRUCTDECL1 -> ''
CLASSBODYMETHODDECL -> METHODDECL CLASSBODYMETHODDECL1
CLASSBODYMETHODDECL1 -> CLASSBODYMETHODDECL
CLASSBODYMETHODDECL1 -> ''
VARDECL -> VARDECLTYPE ident VARDECL1
VARDECL1.her = VARDECLTYPE.type
addType(ident,VARDECL1.sin)
VARDECL1 -> VARDECLBRACKETS VARDECL2
VARDECLBRACKETS.her = VARDECL1.her
VARDECL2.her = VARDECL1.her
VARDECL1.sin = VARDECLBRACKETS.sin
VARDECL1 -> VARDECL2
```

```
VARDECL2.her = VARDECL1.her
VARDECL1.sin = VARDECL2.sin
VARDECL2 -> VARDECLWITHCOMA
VARDECLWITHCOMA.her = VARDECL2.her
VARDECL2.sin = VARDECL2.her
VARDECL2 -> ''
VARDECL2.sin = VARDECL2.her
VARDECLTYPE -> int
VARDECLTYPE.type = 'int'
VARDECLTYPE -> string
VARDECLTYPE.type = 'string'
VARDECLTYPE -> ident
VARDECLTYPE.type = tabSimbolo(ident)

VARDECLBRACKETS -> [ int-costant ] VARDECLBRACKETS1
VARDECLBRACKETS1.her = VARDECLBRACKETS.her
VARDECLBRACKETS.sin =
array(tabSimbolo(int-constant),VARDECLBRACKETS1.sin)
VARDECLBRACKETS1 -> ''
VARDECLBRACKETS1.sin = VARDECLBRACKETS1.her
VARDECLBRACKETS1 -> VARDECLBRACKETS
VARDECLBRACKETS.her = VARDECLBRACKETS1.her
VARDECLBRACKETS1.sin = VARDECLBRACKETS.sin
VARDECLWITHCOMA ->  , ident VARDECLWITHCOMA1
VARDECLWITHCOMA1.her = VARDECLWITHCOMA.her
addType(ident,VARDECLWITHCOMA1.sin)
VARDECLWITHCOMA1 ->  VARDECLBRACKETS VARDECL2
VARDECLBRACKETS.her = VARDECLWITHCOMA1.her
VARDECLWITHCOMA1.sin = VARDECLBRACKETS.sin
VARDECL2.her = VARDECLWITHCOMA1.her
VARDECLWITHCOMA1 -> VARDECLWITHCOMA
VARDECLWITHCOMA.her = VARDECLWITHCOMA1.her
VARDECLWITHCOMA1.sin = VARDECLWITHCOMA1.her
```

```
VARDECLWITHCOMA1 -> ''
VARDECLWITHCOMA1.sin = VARDECLWITHCOMA1.her
CONSTRUCTDECL -> constructor METHODBODY
METHODDECL -> VARDECLTYPE METHODDECL1
METHODDECL1 -> ident METHODBODY
METHODDECL1 -> VARDECLBRACKETS ident METHODBODY
METHODBODY -> ( METHODBODY1
METHODBODY1 -> PARAMLIST ) STATEMENT
METHODBODY1 -> ) STATEMENT
PARAMLIST -> VARDECLTYPE ident PARAMLIST12
PARAMLIST12 -> VARDECLBRACKETS PARAMLIST13
PARAMLIST12 -> PARAMLIST2
PARAMLIST12 -> ''
PARAMLIST13 -> PARAMLIST2
PARAMLIST13 -> ''
PARAMLIST2 -> , VARDECLTYPE ident PARAMLIST12
STATEMENT -> PRINTSTAT ;
STATEMENT -> READSTAT ;
STATEMENT -> RETURNSTAT ;
STATEMENT -> SUPERSTAT ;
STATEMENT -> IFSTAT
STATEMENT -> FORSTAT
STATEMENT -> { STATLIST }
STATEMENT -> break ;
STATEMENT -> ;
STATEMENT -> int ident VARDECL1 ;
STATEMENT -> string ident VARDECL1 ;
STATEMENT -> ident STATEMENT1
STATEMENT1 -> ident VARDECL1 ;
STATEMENT1 -> = ATRIBSTAT1 ;
STATEMENT1 -> LVALUEEXPLIST = ATRIBSTAT1 ;
ATRIBSTAT -> ident LVALUE1 = ATRIBSTAT1
ATRIBSTAT1 -> ALOCEXPRESSION
```

```
ATRIBSTAT1 -> EXPRESSION
PRINTSTAT -> print EXPRESSION
READSTAT -> read LVALUE
RETURNSTAT -> return RETURNSTAT1
RETURNSTAT1 -> EXPRESSION
wRETURNSTAT1 -> ''
SUPERSTAT -> super ( SUPERSTAT1
SUPERSTAT1 -> )
SUPERSTAT1 -> ARGLIST )
IFSTAT -> if ( EXPRESSION ) STATEMENT IFSTAT1
IFSTAT1 -> else STATEMENT end
IFSTAT1 -> end
FORSTAT -> for ( FORSTAT1
FORSTAT1 ->ATRIBSTAT ; FORSTAT2
FORSTAT1 -> ; FORSTAT2
FORSTAT2 -> EXPRESSION ; FORSTAT3
FORSTAT2 -> ; FORSTAT3
FORSTAT3 -> ATRIBSTAT ) STATEMENT
FORSTAT3 -> ) STATEMENT
STATLIST -> STATEMENT STATLIST1
STATLIST1 ->  STATLIST
STATLIST1 ->  ''
LVALUE -> ident LVALUET2
LVALUET2.her = tabSimbolo(ident)
LVALUET2.hertype = type(ident)
LVALUE.node = new leaf(id, LVALUET2.sin)
LVALUET2 ->[ int-constant ] LVALUET2
LVALUET2.hertype =
validate(LVALUET2.hertype,tabSimbolo(int-constant))
%Retorna o tipo interno no caso de
array(2,array(3,integer)) retorna array(3,integer)
```

```
LVALUET2'.her =
LVALUET2.her+"[tabSimbolo(int-constant)]"
LVALUET2.sin = LVALUET2'.sin
LVALUET2 -> ''
LVALUET2.sin = LVALUET2.her
LVALUE1 -> LVALUEEXPLIST
LVALUE1 -> ''
LVALUEEXPLIST -> [ EXPRESSION ] LVALUEEXPLIST1
LVALUEEXPLIST -> . ident LVALUEEXPLIST2
LVALUEEXPLIST1 -> LVALUEEXPLIST
LVALUEEXPLIST1 -> ''
LVALUEEXPLIST2 -> ( LVALUEEXPLIST3
LVALUEEXPLIST2 -> LVALUEEXPLIST
LVALUEEXPLIST2 -> ''
LVALUEEXPLIST3 -> ARGLIST ) LVALUEEXPLIST1
LVALUEEXPLIST3 -> ) LVALUEEXPLIST1
ALOCEXPRESSION -> new ALOCEXPRESSION1
ALOCEXPRESSION1 -> ident ALOCEXPRESSION2
ALOCEXPRESSION1 -> int ALOCEXPRESSIONPLUS
ALOCEXPRESSION1 -> string ALOCEXPRESSIONPLUS
ALOCEXPRESSION2 -> ( ALOCEXPRESSION3
ALOCEXPRESSION2 -> ALOCEXPRESSIONPLUS
ALOCEXPRESSION3 -> ARGLIST )
ALOCEXPRESSION3 -> )
ALOCEXPRESSIONPLUS -> [ EXPRESSION ]
ALOCEXPRESSIONPLUS1

ALOCEXPRESSIONPLUS1 -> ALOCEXPRESSIONPLUS
ALOCEXPRESSIONPLUS1 -> ''
EXPRESSION -> NUMEXPRESSION EXPRESSION1
EXPRESSION1.her = NUMEXPRESSION.sin
EXPRESSION.sin = EXPRESSION1.node
EXPRESSION1 -> EXPRESSIONCOMPARE NUMEXPRESSION
```

```
EXPRESSION1.node = new
node(EXPRESSIONCOMPARE.node,EXPRESSION1.her,NUMEXPRESSI
ON.sin)
EXPRESSION1 -> ''
EXPRESSION1.node = EXPRESSION1.her
EXPRESSIONCOMPARE -> !=
EXPRESSIONCOMPARE.node = new leaf (id ,!=)
EXPRESSIONCOMPARE -> ==
EXPRESSIONCOMPARE.node = new leaf (id ,==)
EXPRESSIONCOMPARE -> >=
EXPRESSIONCOMPARE.node = new leaf (id ,>=)
EXPRESSIONCOMPARE -> <=
EXPRESSIONCOMPARE.node = new leaf (id ,<=)
EXPRESSIONCOMPARE -> >
EXPRESSIONCOMPARE.node = new leaf (id ,>)
EXPRESSIONCOMPARE -> <
EXPRESSIONCOMPARE.node = new leaf (id ,<)
NUMEXPRESSION -> TERM NUMEXPRESSION1
NUMEXPRESSION1.her = TERM.sin
NUMEXPRESSION.sin =  NUMEXPRESSION1.node
NUMEXPRESSION1 -> SUMMINUS NUMEXPRESSION
NUMEXPRESSION1.node = new node (SUMMINUS.node,
NUMEXPRESSION1.her, NUMEXPRESSION.node)
NUMEXPRESSION1 -> ''
NUMEXPRESSION1.node = NUMEXPRESSION1.her
SUMMINUS -> +
SUMMINUS.node = new leaf (id ,+)
SUMMINUS -> -
SUMMINUS.node = new leaf (id ,-)
TERM -> UNARYEXPR TERM3
TERM3.her = UNARYEXPR.sin
TERM.sin = TERM3.sin
TERM2 -> MULDIVMOD UNARYEXPR TERM3
```

```
TERM2.node = new node (MULDIVMOD.node, TERM2.her,
TERM3.sin)
TERM3.her = UNARYEXPR.sin
TERM3 -> TERM2
TERM2.her = TERM3.her
TERM3.sin = TERM2.node
TERM3 -> ''
TERM3.sin = TERM3.her
MULDIVMOD -> %
MULDIVMOD.node = new leaf( id,%)
MULDIVMOD -> /
MULDIVMOD.node = new leaf( id,/)
MULDIVMOD -> *
MULDIVMOD.node = new leaf( id,*)
UNARYEXPR -> SUMMINUS FACTOR
UNARYEXPR.sin = new node(SUMMINUS.node, FACTOR.node)
UNARYEXPR -> FACTOR
UNARYEXPR.sin = FACTOR.node
FACTOR -> int-constant
FACTOR.node = new leaf( id, tabSimbolo(int-constant))
FACTOR -> string-constant
FACTOR.node = new leaf( id,
tabSimbolo(string-constant))
FACTOR -> null
FACTOR.node = new leaf( id, tabSimbolo(null))
FACTOR -> LVALUE
FACTOR.node = LVALUE.node

FACTOR -> ( NUMEXPRESSION )
FACTOR.node = NUMEXPRESSION.sin
ARGLIST -> EXPRESSION ARGLIST2
ARGLIST2 -> ARGLISTEXP
ARGLIST2 -> ''
```

ARGLISTEXP -> , **EXPRESSION ARGLIST2**