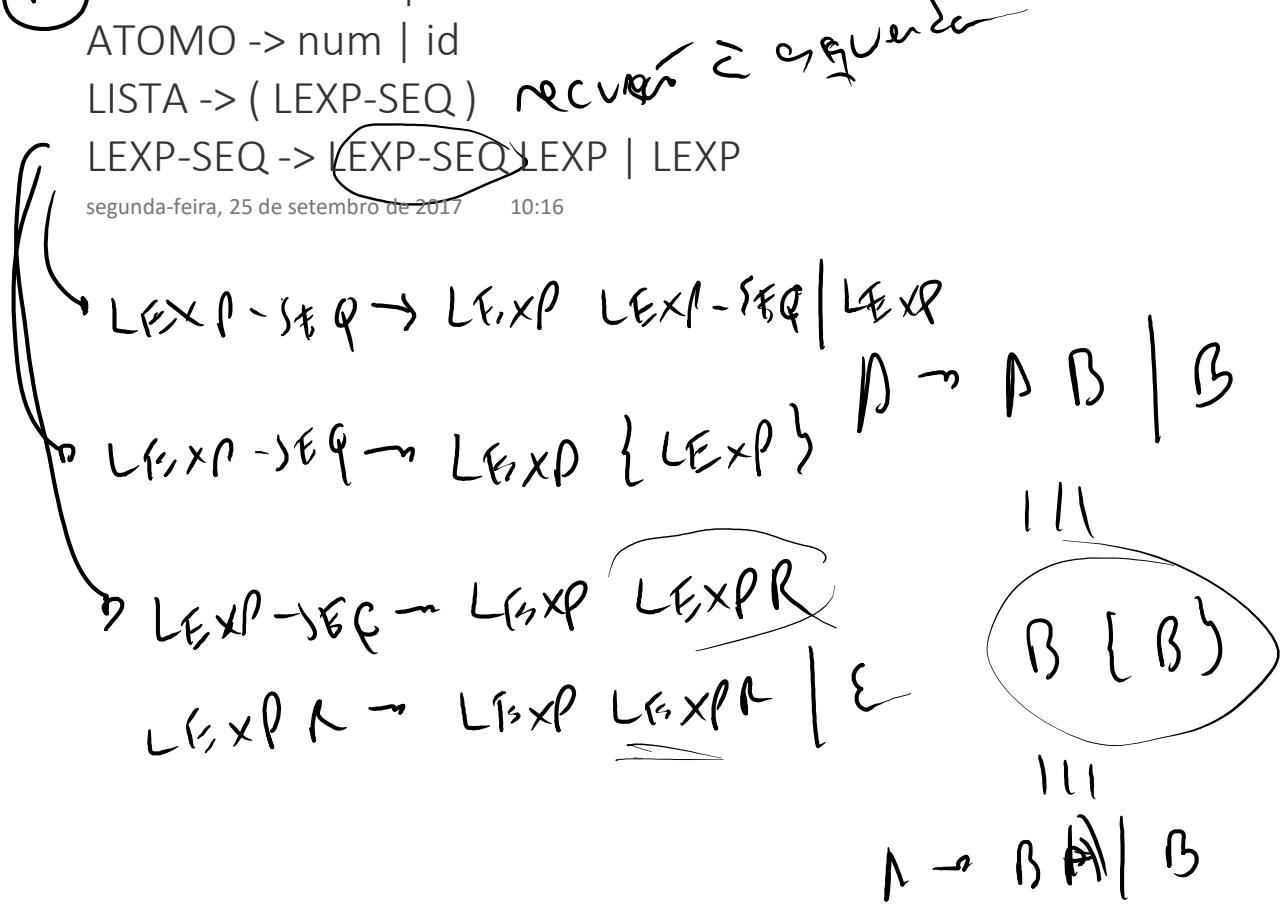
17

LEXP -> ATOMO | LISTA
ATOMO -> num | id
LISTA -> (LEXP-SEQ)
LEXP-SEQ -> LEXP-SEQ LEXP | LEXP

segunda-feira, 25 de setembro de 2017 10:16




```
Tree LEXP() {
  val res = Tree("LEXP")
  val atual = pos
  try {
    res.child(ATOMO())
  } catch(Falha f) {
    pos = atual
    res.child(LISTA())
  }
  return res
}

Tree ATOMO() {
  val res = Tree("ATOMO")
  try {
    res.child(match(Token.NUM))
  } catch(Falha f) {
    res.child(match(Token.ID))
```

```
  }
  return res
}

Tree LISTA() {
  val res = Tree("LISTA")
  res.child(match('('))
  res.child(LISTA_SEQ())
  res.child(match(')'))
  return res
}

Tree LISTA_SEQ() {
  val res = Tree("LISTA_SEQ")
  val atual = pos
  try {
    val rascunho = Tree()
    rascunho.child(LEXP())
    rascunho.child(LEXP_SEQ())
    res.children.addAll(rascunho.children)
  } catch(Falha f) {
    pos = atual
    res.child(LEXP())
  }
  return res
}

Tree LISTA_SEQ() {
  val res = Tree("LISTA_SEQ")
  res.child(LEXP())
  while(true) {
    val atual = pos
    try {
      res.child(LEXP())
    } catch(Falha f) {
      pos = atual
```

```
      break
    }
  }
  return res
}
```

VAR-LISTA -> id , VAR-LISTA | id <=> VAR-LISTA -> id { , id }

CMD -> ATRIB
CMD -> CHAMADA
CMD -> outro
ATRIB -> id := exp
CHAMADA -> id (exp)

```
// CMD -> id (:= exp | '(' exp ')') | outro
Tree CMD() {
    val res = Tree("CMD")
    when(la.tipo) {
      Token.ID -> {
        val id = match(Token.ID)
        when(la) {
          '(' -> {
            val chamada = Tree("CHAMADA")
            chamada.child(id)
            chamada.child(match('('))
            chamada.child(match(Token.EXP))
            chamada.child(match(')'))
            res.child(chamada)
          }
          else -> {
            val atrib = Tree("ATRIB")
```

```
            atrib.child(id)
            atrib.child(match(Token.ATRIB))
            atrib.child(match(Token.EXP))
            res.child(atrib)
          }
        }
      }
      else -> res.child(match(Token.OUTRO))
    }
    return res
}
```

A -> (A) A | *vazio*

```
Tree A() { // preditivo LL(1)
  val res = Tree("A")
  when(la.tipo) {
    '(' -> {
      res.child(match('('))
      res.child(A())
      res.child(match(')'))
      res.child(A())
    }
    ')' -> {}
    Token.EOF -> {}
    else -> throw RuntimeException("erro de sintaxe")
  }
  return res
}
```

```
case '{':
```

(b)

```
case '{':
        int n = 1;
      nextChar();
        while(lookAhead != EOF && n > 0) {
        if(lookAhead == '{') n++;
        elseif(lookAhead == '}') n--;
          nextChar();
        }
      if(n != 0)
          error("comentário não terminado");
        continue;
```

```
case '{':
        COMENTARIO();

// COMENTARIO -> '{' { COMENTARIO | . } '}'
void COMENTARIO() {
        nextChar();
      while(lookAhead != EOF && lookAhead != '}') {
           if(lookAhead == '{') COMENTARIO();
         else nextChar();
      }
      if(lookAhead == EOF) error("comentário não terminado");
      nextChar();
      return;
}
```

(a) S → n ; { n ; }

(c) S → S n ; | n ;

(1) e (4)

LL(1)

③ $S \to s;\, S \mid s;$

② e ③ !LL(1)

④ $S \to s;\, S'$

$S' \to s;\, S' \mid \varepsilon$

(S - ambíguo)

$S \to S\,S \mid s;$