Caneca++

Chrystian Guth
Lucas Pereira
Renan Netto

Antlr

O Antlr já possui uma opção de recuperação de erros.

Ao encontrar um símbolo inválido, o Antlr elimina o símbolo bem como todos os próximos símbolos seguintes até encontrar um **follow** do símbolo inválido.

Entretanto, o Antlr trata de forma especial os casos de **UnwantedTokenException** e **MissingTokenException**.

Antlr

Ao se deparar com um **MissingTokenException**, o Antlr utiliza a estratégia de **recuperação por inserção**, inserindo o símbolo faltante.

Exemplo: `10 * (12 + 4;`

Como está faltando o `)`, o Antlr irá gerar uma mensagem de erro, inserir o símbolo faltante e prosseguir a análise.

Antlr

Para os casos de **UnwantedTokenException** o **Antlr** utiliza a estratégia de **recuperação por remoção**, removendo o símbolo não desejado.

Exemplo: `10 * (12 + 4));`

Como está sobrando o `)`, o Antlr irá gerar uma mensagem de erro, verificar se o próximo símbolo era o esperado, e caso seja, irá eliminar o símbolo inválido e prosseguir com a análise.

Antlr

Optamos por tratar todos os erros de reconhecimento de forma igual: eliminando todos os símbolos até encontrar um **follow** do símbolo inválido.

O Antlr utiliza os seguintes métodos para recuperação de erros:

```
recover
recoverFromMismatchedToken
RecoverFromMismatchedSet
```

Onde os dois últimos são utilizados para os casos especiais mencionados anteriormente.

Tratamento de erros sintáticos

fontes/g/**CanecaSintatico.g**

```
@members {
...
@Override
public void recover(IntStream entrada,
RecognitionException erro) {
   if (state.lastErrorIndex == entrada.index()) {
         entrada.consume();
   }
   state.lastErrorIndex = entrada.index();
   BitSet conjuntoDeFollows = computeErrorRecoverySet();
   beginResync();
   consumeUntil(entrada, conjuntoDeFollows);
   endResync();
}
...
```

Tratamento de erros sintáticos

```
...
@Override
public Object recoverFromMismatchedToken(IntStream
entrada, int tipoDoSimbolo, BitSet conjuntoDeFollows)
throws RecognitionException {
   throw new MismatchedTokenException(tipoDoSimbolo,
entrada);
}

@Override
public Object recoverFromMismatchedSet(IntStream entrada,
RecognitionException erro, BitSet conjuntoDeFollows)
throws RecognitionException {
   throw erro;
}
...
}
```

Tratamento de erros léxicos

fontes/g/**CanecaLexico.g**

```
@members {
...
@Override
public void recover(RecognitionException erro) {
   input.consume();
}
...
}
```

Arvore sintática

O Antlr possui uma opção para gerar automaticamente a árvore sintática:

`fontes/g/CanecaArvore.g`

```
options {
   output = AST;
}
```

Entretanto, a árvore gerada consiste de um nodo pai contendo, em um mesmo nível da árvore, todas as sub-regras reconhecidas. Se nada for especificado, o Antlr irá gerar uma "**flat tree**".

Para gerar uma árvore com uma estrutura mais adequada é necessário dizer ao Antlr o nodo raiz das sub-regras.

Arvore sintática

O Antlr fornece o módulo **tree rewrite syntax** que possibilita reescrever as regras sintáticas bem como manipular a árvore de sintaxe.

Com o **tree rewrite syntax** também é possível remover informações que são necessárias apenas para a descrição da linguagem mas não têm importância no processo de geração de código.

Arvore sintática

fontes/g/**CanecaArvore.g**

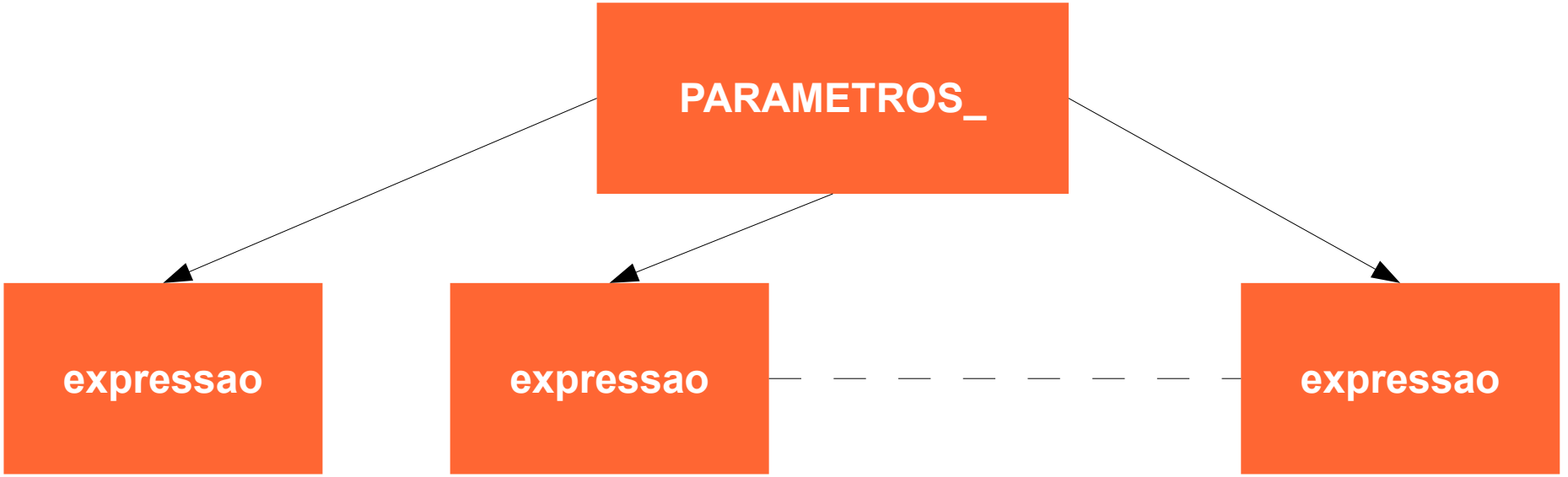
```
listaDeParametros
: PARENTESE_ESQUERDO (expressao (SEPARADOR
expressao)*)? PARENTESE_DIREITO
-> ^(PARAMETROS_ (expressao)*)
;
```

A regra **listaDeParametros** é reescrita de modo que a saída gerada (a árvore sintática) não inclua os parenteses e nem os separadores.

O símbolo **^** é utilizado para indicar que o primeiro item será o nodo raiz da sub-regra.

Arvore sintática

A reescrita da sub-regra anterior irá resultar na seguinte sub-árvore:



Da mesma forma, a sub-árvore **expressao** será composta por uma outra sub-árvore.

Compilador

Nessa terceira etapa o compilador passa a ter novas opções:

(**lexica**) Análise léxica com exibição dos erros no console.
(**sintatica**) Análise sintática com exibição dos erros no console.
(**simbolos**) Apresentação dos símbolos de uma análise.
(**arvore**) Apresentação da árvore sintática de uma análise.

`fontes/java/.../Compilador.java`

Uso do compilador:

```
java -classpath
binarios/class:bibliotecas/jar/antlr.jar
br.ufsc.inf.ine5426.caneca.Compilador.java
<arquivo.caneca>
<opcao: {lexica, sintatica, arvore, simbolo}>
```

Exemplos

Análise léxica, análise sintática, apresentação dos símbolos e apresentação da árvore sintática dos arquivos:

`fontes/caneca/ErrosLexicos.caneca`
`fontes/caneca/ErrosSintaticos.caneca`
`fontes/caneca/AcertosLexicos.caneca`
`fontes/caneca/AcertosSintaticos.caneca`