Caneca++

Chrystian Guth
Lucas Pereira
Renan Netto

Antlr

O Antlr permite especificar uma gramática que recebe como entrada uma árvore. As regras da gramática representarão nodos da árvore de entrada.

Utilizamos como árvore de entrada a mesma que foi descrita na etapa anterior e é gerada pelo **CanecaArvore**.

Também é possível filtrar os nodos que são “vistos” no percorrimento da árvore.

Para cada regra é possível especificar um código que será executado quando a regra for reconhecida.

Antlr

Na etapa de **análise sintática**, criamos a seguinte regra:

```
classe : CLASSE modificadorDeAcessoFeminino
IDENTIFICADOR listaDeTiposGenericos
listaDeInterfaces corpoDaClasse
```

Já na etapa de **geração da árvore sintatática** reescrevemos está regra para gerar a seguinte estrutura:

```
^(CLASSE_ modificadorDeAcessoFeminino
IDENTIFICADOR listaDeTiposGenericos
listaDeInterfaces corpoDaClasse)
```

Antlr

Agora, na etapa de **análise semântica** utilizaremos o `.` (ponto) para filtrar elementos da árvore sintática e trabalhar apenas com os elementos que forem de interesse. Por exemplo:

```
^(CLASSE_ . IDENTIFICADOR . . .)
```

Utilizamos essa regra para realizar a primeira etapa na análise semântica, a definição dos símbolos. Isso é útil, pois na etapa de definição dos símbolos somente precisamos do *token* `IDENTIFICADOR` da classe. Com isso, filtramos (ignoramos) os outros *tokens* através da utilização do ponto.

Antlr

Para cada regra é possível definir um código que será executado quando a regra for reconhecida. Com isso, definimos códigos Java para realizar a análise semântica.

O Antlr permite caminhar pela árvore sintática de cima para baixo, de baixo para cima e até mesmo das duas formas. Para isso, basta utilizar as regras predefinidas por ele: `topdown` e `bottomup`. Essas regras deverão conter as regras que serão reconhecidas no percorrimento da árvore.

Análise semântica

```
IniciarClasse : ^(CLASSE_ . IDENTIFICADOR . . .) {
            Classe classe = new Classe($IDENTIFICADOR.text);
            boolean classeDefinida = escopoAtual.definirClasse(classe);
            ...
            classe.abrir(escopoAtual);
            escopoAtual = classe;
            filaDeEscopos.add(escopoAtual);
            Atributo esse = new Atributo("esse",
                new Tipo($IDENTIFICADOR.text));
            Atributo essa = new Atributo("essa",
                new Tipo($IDENTIFICADOR.text));
            boolean esseDefinido = classe.definirAtributo(esse);
            boolean essaDefinida = classe.definirAtributo(essa);
            ...
            esse.abrir(escopoAtual);
            esse.fechar();
            essa.abrir(escopoAtual);
            esse.fechar();
        }
    ;
}
```

Análise semântica

Dividimos a análise semântica em quatro passos:

Definição das classes, métodos, atributos e variáveis.

Resolução dos símbolos utilizados: referências à variáveis, classes, métodos, etc.

Verificação estática dos tipos das **expressões**.

Verificações extras como: checagem de parâmetros e **chamadas** de métodos.

Análise semântica

Ao todo são feitas quatro passadas pela árvore sintática.

Porém, poderíamos ter feito a análise semântica apenas com duas passadas: uma para a **definição** dos escopos e símbolos e outra para a **resolução**, **verificação estática de tipos** e **verificações extras**.

Decidimos realizar a **verificação estática de tipos** e as **verificações extras** em passadas separadas para facilitar a leitura e manutenção do código.

Análise semântica

Pelo menos duas passadas são necessárias pelo fato de existir a possibilidade de se referenciar uma classe ou um método que ainda não foi definido.

Por exemplo, supondo que a classe `A` seja definida anteriormente e a classe `B` logo em seguida. Caso não houvesse duas passadas, a classe `A` não conseguiria resolver o símbolo `B`, pois este ainda não teria sido definido na tabela de símbolos.

Análise semântica

Na parte da **definição**, utilizamos o conceito de tabela de símbolos e de escopo. Definimos que a tabela de símbolos será simplesmente um escopo global.

Os escopos conterão outros escopos e também conterão a definição de variáveis, métodos, classes, etc.

Classes, métodos, blocos, construtores e destrutores também são escopos.

Análise semântica

```
1  ...
   pacote @pacoteQualquer
2  classe publica ClasseQualquer inicio
       atributo publico Texto nome;  2
       atributo publico Rela pi;  2
3      metodo publico Nada nadaFaz() inicio  2
           nova OutraClasse().atributoDaOutraClasse;  6
4          se (verdadeiro) inicio
5              se (1 == 1) inicio
                   Real pi;  5
                   pi = 3 + 0.14;  5
               fim
           fim
       fim
   Fim
   ...
6  Classe publica OutraClasse inicio
   ...
```

Análise semântica

A **tabela de símbolos** possui em seu escopo apenas *classes*.

As **classes** possuem em seu escopo *atributos* e *métodos*. Enquanto que os **métodos** possuem em seu escopo os *argumentos* e os *blocos de instruções*.

Os **blocos de instruções** possuem em seu escopo *variáveis* e outros *blocos de instruções*.

O processo de resolução consiste em verificar se determinada classe, atributo, método ou variável se encontra nos escopos superiores.

Análise semântica

```
...
classe publica ClasseQualquer inicio
   atributo publico Texto nome;
   metodo publico Nada nadaFaz() inicio
      se (verdadeiro) inicio
         nome = “Teste”;
      fim
   fim
fim
...
```

Por exemplo, no código acima é feita uma referencia a um atributo que está dentro de um bloco que está dentro de um método. Os passos para resolver o atributo serão:

Análise semântica

O **bloco** verifica se a referência é uma *variável* que está dentro do seu escopo.

Como não encontra nada, o **bloco** pede ao escopo pai (o **método**) para resolver a referência.

O **método** verifica se a referência é um *argumento* que pertence ao seu escopo. Como não é, o **método** pede para que a **classe** resolva a referência.

Finalmente a **classe** resolverá a referência já que se trata de um *atributo* dela.

Análise semântica

```
...
classe publica ClasseQualquer inicio
   metodo publico Nada nadaFaz() inicio
      Texto texto = “cameca”;
      texto.paraMaiusculas();
   fim
fim
...
```

Para referências a atributos externos, a análise complica um pouco, porém segue o mesmo princípio. A diferença está no fato de que a referência será buscada no escopo ao qual pertence o tipo do objeto referenciado.

Nesse caso a referência será resolvida no escopo do tipo **Texto** e não no escopo do tipo **ClasseQualquer**.

Análise semântica

Para a **verificaçõa estática dos tipos** foi utilizado um conceito semelhante ao conceito dos escopos.

No nosso caso, cada expressão possui um ou dois operandos sendo que estes podem ser elementos da linguagem ou até mesmo outras expressões.

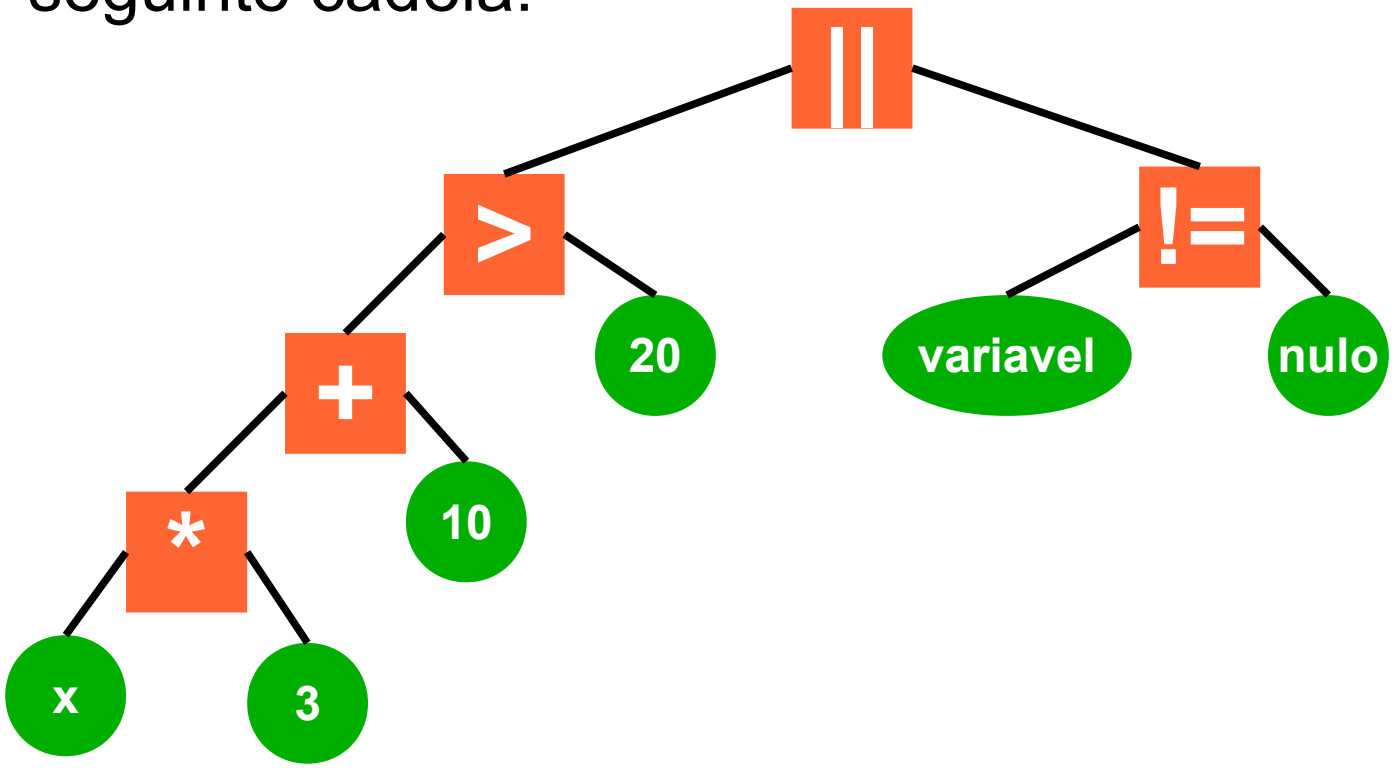
Cada tipo de expressão é reponsável por verificar se os seus operandos respeitam os símbolos exigidos.

Análise semântica

Por exemplo, na expressão:

```
10 + x * 3 > 20 || variavel != nulo
```

Segundo a ordem de precedência das expressões, temos a seguinte cadeia:



Análise semântica

Primeiramente a `expressaoMultiplicativa` irá verificar se os seus operandos são numéricos. Além da verificação, a `expressaoMultiplicativa` repassa o seu tipo para a `expressaoAditiva`. O tipo dependerá dos operandos. Por exemplo, vamos supor que `x` seja do tipo `Real`, então o tipo da `expressaoMultiplicativa` será do tipo `Real`.

A `expressaoAditiva` age da mesma forma que a `expressaoMultiplicativa` e repassa o seu tipo à `expressaoComparativa` que verificará os operandos e passará o seu tipo (`Booleano`) à `expressaoOuLogico`.

Por sua vez, a `expressaoOuLogico` espera que seus operandos sejam `Booleanos` e o seu tipo será também `Booleano`.

Análise semântica

Após a realização das análises anteriores, a verificação de parâmetros **de chamadas de métodos** e **de instaciações** se tornou mais simples.

Cada expressão sabe determinar o seu tipo. A lista de parâmetros é formada por expressões. Com isso, basta verificar se os tipos dos parâmetros combinam com a assinatura do **método** ou do **construtor** que se deseja invocar.

Compilador

Nessa quarta etapa o compilador passa a ter uma nova opção:

(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.
(semantica) Análise semântica com exibição dos erros no console.

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

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

Exemplos

fontes/caneca/**ErrosSemanticosDefinicaoResolucao.caneca**
fontes/caneca/**ErrosSemanticosExpressoes.caneca**
fontes/caneca/**ErrosSemanticosChamada.caneca**
fontes/caneca/**AcertosSemanticos.caneca**