Linguagem de Análise Dimensional

Compiladores 2019/20

Trabalho realizado por :
Daniel Gomes NMEC: 93015
Gonçalo Pereira NMEC: 93310
Miguel Marques NMEC: 100850
Pedro Bastos NMEC: 93150
Pedro Tavares NMEC: 93103

Índice

Compiladores 2019/20 **1**

Introdução **3**

Linguagem Complementar **4**

Definições de Dimensões e ordens de grandeza 4

Análise Semântica 5

Linguagem Principal **6**

Declarações e atribuições de valores a variáveis 7

Interação com o Utilizador 8

Impressão de dados no Terminal 8

Expressões de comparação 9

Instruções Condicionais 10

Instruções Iterativas 11

Verificação de grandezas 12

Análise Semântica 12

Geração de Código **13**

Conclusão e Notas Finais **14**

Introdução

Com vista à realização do trabalho Prático da disciplina de Compiladores, onde o objetivo do Trabalho consistia no desenvolvimento de um Compilador e de um Interpretador, criamos duas linguagens diferentes: uma linguagem Principal que serve de base para o Compilador e fazendo uso de uma Linguagem Complementar que é relativa à forma como definição de unidades e grandezas novas (Interpretador).

Linguagem Complementar

Na gramática relativa às dimensões permitimos que haja a definição de dimensões base, de dimensões compostas por duas dimensões definidas previamente e a criação de ordens de grandeza dentro duma determinada dimensão definida anteriormente, sempre que uma destas operações é realizada a dimensão é guardada no mapa dimTable que está definido na gramática Dimensions.g4 (associado ao Parser da gramática) sendo que a chave é o nome da dimensão e o valor associado é uma dimensão que é constituída pelo nome da dimensão, tipo primitivo (integer ou float) e por unidade.

Definições de Dimensões e ordens de grandeza

De forma a definir novas dimensões no programa o programador pode usar uma das seguintes formas:

- "<Nova_Dimensão> -> <tipo_primitivo> (<unidade>);"
- "<Nova_Dimensão> -> addUnit <Dimensão_Base><operador><Dimensão_Base2>;"

Alguns exemplos de definição de dimensões são:

- *Length -> real (m);*
- *Time -> real (s);*
- *Speed -> Length / Time;*

Erros na definição de dimensões:

- *Length -> real m;*
- *Time = real (s);*
- *length -> real (m);*
- *Speed -> m/Time;*

De forma a definir uma nova ordem de grandeza dentro duma dimensão predefinida deve seguir a seguinte forma:

- "<Dimensao> -> addUnit <NovaUnidade> = <polinomiodegrau1>;"

Alguns exemplos para adicionar uma nova unidade à dimensão existente são:

- *Length -> addUnit cm = 100*m;*
- *Temperature -> addUnit F = 1.8*C+32;*

Erros na definição de ordens de grandeza das unidades:

- *Temperature -> addUnit C = 100*F^2;*
- *Length -> addUnit m = 2;*
- *Force -> addUnit N = m*a;*

Análise Semântica

De forma a realizar a análise semântica relativa à gramática complementar foi criado o visitor **DimCheck.java,** de forma a garantir que não existem erros quando o código é compilado.

Para isso são feitas as seguintes verificações:

- O nome da dimensão tem de começar por letra maiúscula e não pode ter espaços.
- Não se pode definir a mesma dimensão ou unidade duas vezes.
- Ao definir uma dimensão base a unidade tem de ter parênteses em volta.
- Uma grandeza que não seja base apenas pode ser definida através de grandezas previamente definidas.
- Ao definir uma grandeza tendo por base outras grandezas, é necessário que o número de grandezas existentes seja sempre 2.
- Ao definir uma dimensão através de outras dimensões existentes apenas é possível usar os operadores “*” ou “/” entre as grandezas existentes.
- Novas ordens de grandeza duma determinada dimensão apenas podem ser criadas à custa de uma unidade previamente definida relativa a essa mesma dimensão.
- Não são permitidas unidades repetidas quer seja na mesma dimensão que em dimensões diferentes.

Linguagem Principal

Na gramática principal, foram definidos todos os tipos de estruturas para compor o resto da linguagem, como estruturas condicionais if, for, while, declarações, prints, assignments, incrementos, scans, declaração de Dimensões e imports de ficheiro de texto que contenham as dimensões declaradas e façam a verificação das mesmas. Podem ser feitos vários imports, sendo isto opcional.
A classe **Symbol.java** representa uma variável e a classe **Type.java** é a classe abstrata do tipo de dados.

De forma a facilitar a interpretação do relatório daqui em diante as expressões <expr> que vão ser mencionadas podem ser uma das seguintes:

- Potência;
- Somas e subtrações;
- Expressão e sua negação (representado por:”!”);
- Multiplicação, divisão e resto de divisão;
- Valor;
- Valor com sinal;
- Comparação (“==”, ”!=”, ”<=”, ”>=”, ”<”, ”>”)
- Expressões booleanas (“&&” e “||”);
- Expressão entre parênteses;
- Input;
- Incremento/Decremento (“++”, “--”)
- Variável;

Declarações e atribuições de valores a variáveis

Na nossa linguagem a declaração e atribuição de valores a variáveis é feita da seguinte forma:

- *<tipo_dados> <nome_variavel> ;*
- *<tipo_dados> <nome_variavel> (, <nome_variavel>)? ;*
- *<tipo_dados> <nome_variavel> = <expressão_a_atribuir> ;*
- *<nome_variavel> = <expressão _a_ atribuir> ;*

São permitidos os tipo primitivos como **integer**, **real**, **string**, **boolean** e também as novas **dimensões** que são definidas com a linguagem de definição de dimensões acima descrita, para isto é necessário importar um ou mais ficheiros que contenha a definição das dimensões, e caso análise semântica a este ficheiro não detete qualquer erro,então poder-se-á declarar variáveis do tipo dimensão. A realização da importação é feita da seguinte maneira:

- *import dimensoes1;*
  import dimensoes2;

Exemplos de declarações e atribuições que se podem fazer na nossa linguagem:

- *Velocity vel;*
- *Length d = 0.0 (m);*
- *integer i;*
- *real r;*
- *i = 10;*
- *integer s = i;*

Na nossa linguagem não é possível declarar variáveis com o mesmo nome, não é possível declarar Dimensões que não tenham sido criadas previamente, ou seja que não esteja no ficheiro importado. A nível de atribuição de valores, não é possível na nossa linguagem atribuir um valor de uma dimensão a uma outra diferente, ou seja, **não é possível** fazer isto:

- *Velocity vel ;*
- *Time t = 2.0;*
- *vel = t;*

Interação com o Utilizador

Em termos de input de dados do utilizador, a nossa linguagem permite a seguinte estrutura:

- *<tipo_dados> <nome_variavel> =* ***scan****(<texto_para_display>, <tipo_dados>);*
- *<nome_variavel> =* ***scan****(<texto_para_display>, <tipo_dados>);*

Exemplo de casos em que se pode usar o scan:

- *integer i =* ***scan****("Insira um número", integer);*
- *real j = scan("insira um número", integer);*
- *real j = scan("insira um número", real);*
- *Velocity vel =* ***scan****("Insira uma velocidade", velocity);*
- *string s =* ***scan****("Insira o seu nome", string);*

Não é permitido a introdução de dados sem ter uma variável associada para receber o valor.

Impressão de dados no Terminal

A impressão de dados no Terminal é permitida consoante a estrutura:

- **println(**<expr>**)**;

Onde <expr> consiste em qualquer expressão permitida nesta linguagem.

Exemplos de uso de println:

- **println(**2>1**)**;

- integer i,x=2;
  println(i+ " igual a "+x**)**;

Expressões de comparação

A nível de expressões de comparação, nos definimos para a nossa linguagem expressões com operadores de igualdade (==, !=), expressões com operadores relacionais (>, <, >=, <=) e expressões com operadores and/or (&&, | |).

A nível das **expressões com operadores de igualdade**, na nossa linguagem, é possível os exemplos:

- *boolean a = 1 == 2 ;*

Na nossa linguagem **não é possível** comparar 2 variáveis ou expressões cujos tipos não se conformem um com o outro, por exemplo é comparar uma expressão de tipo **Real** com outra do tipo **Integer**, contudo não é possível comparar uma **String** com um **Boolean**

A nível de **expressões com operadores relacionais**, é possível na nossa linguagem os exemplos:

- boolean a = 1 < 2;
  a = 2 > 1;
- Velocity v = 10 (m/s);
  Velocity s = 20 (m/s);
  boolean b = v < s;

A nível de **expressões com operadores and/or**, é possível na nossa linguagem os exemplos:

- 1 > 2 && 2 < 1;
- Time t = 30 (s);
  t == 30 (s) | | t != 20 (s);
- true && false;

Instruções Condicionais

Instruções condicionais estão presentes em qualquer linguagem de programação. Escolhemos implementar estas instruções com a sintaxe:

- if(<expr>) {
- <statements>
- }
- else if(<expr>){
- <statements>
- }
- else {
- <statements>
- };

Nesta sintaxe <expr> consiste numa **Expressão de comparação**.

Exemplo de utilização de Instruções Condicionais:

- if (1==2) {
- println(“Não entra aqui”);
- }
- else if (2==2){
- println(“Entra aqui”);
- } else{
- println(“Não chega aqui”);
- };

Instruções Iterativas

Instruções iterativas também são muito importantes nas linguagens de programação, sendo responsáveis pela criação de ciclos lógicos.
Implementámos estas instruções com as seguintes sintaxes:

For loop:

- **for (** <assignment>**;** <expr>**;** <expr> **) {**
- <statements>
- **};**

While loop:

- **while(**<expr>**){**
- <statements>
- **};**

Uso correto de Instruções Iterativas:

For loop:

- **for(**integer x=0**;**x<5**;**x++**){**
- println(“> ” + x)
- **}**

While loop:

- Time z = 3(s);
- **while(**z>0(s)**){**
- println(“z: “ + z);
- z -= 1(s);
- **}**

Verificação de grandezas

Para verificar a dimensão e a unidade de certa expressão optámos por usar a sintaxe:

- **(**<expr>**).dimensionInfo**;

O resultado é o print no terminal da dimensão seguida da unidade da expressão.Se não existir dimensão associada a uma certa expressão, no código gerado é impressa uma mensagem a anunciar que não existe dimensão ou unidade para a expressão.

Um caso possível de utilização de verificação de grandezas é:

- Time tempo = 1 (s);
- **(**tempo**).dimensionInfo**;

Análise Semântica

De forma a não incorrer em erros futuros na geração de código java foram feitas restrições no código do utilizador ,entre as quais:

- Ao declarar uma variável de tipo não primitivo, o tipo tem que ter sido declarado previamente e a unidade tem que estar especificada e associada ao mesmo.
- Uma variável apenas pode ser definida uma vez.
- Apenas se podem fazer operações numéricas a valores numéricos mas pode-se usar “+” para concatenar texto com todo o tipo de variáveis exceto boolean.
- Em somas e subtrações os operandos têm que pertencer à mesma dimensão.
- Nas expressões condicionais tem de existir pelo menos uma expressão.
- Incrementar apenas funciona caso as variáveis a incrementar sejam inteiras.
- Dividir duas unidades iguais, o resultado será adimensional.
- Entre outras;

Geração de Código

A nível da geração de código e compilação dos programas da nossa linguagem, nós optamos por escolher como linguagem destino, a linguagem Java, por ser uma linguagem na qual nós temos mais experiência, e também por ser uma linguagem versátil e fácil de usar.

Para a geração de código a primeira coisa que fizemos foi criar um ficheiro do tipo String Template Group File, chamado **java.stg**, que foi usado como o ficheiro template para a geração da versão em Java do nosso programa.

Para efeitos de compilação e usando o ficheiro mencionado acima, criámos um novo visitor chamado **Compiler.java,** que depois de análise semântica ser feita pelo visitor MainGramCheck.java (o visitor da gramática principal) irá percorrer a árvore sintática da gramática MainGram.g4,caso a análise Semântica não tenha detectado erros, outra vez e em cada um dos filhos da árvore irá gerar, com os templates existententes no ficheiro java.stg, os campos do ficheiro que depois será utilizado para correr o código da nossa gramática.

Após o compilador ter percorrido a árvore sintática toda, e gerado o código da nossa linguagem em código Java, deverá ter sido criado um ficheiro ***output.java*** com o código executável da nossa linguagem em Java. Na pasta onde se encontra este documento, estão também alguns ficheiros com programas concretos (“***programa1.txt, programa2.txt”)***, assim como ficheiros onde ocorrem erros semânticos em cada linguagem, estando erros respetivamente referidos em comentário (***“Errosmain.txt”, “Errosdimensoes.txt”***), e por fim um ficheiro com funcionalidades em geral, ***“Funcionalidades1.txt”***).
O código presente na pasta já se encontra compilado, pelo que para testar a geração de código basta fazer o seguinte comando:
- *java **MainGramMain** <”nome programa”>*

Conclusão e Notas Finais

Como o objetivo deste projeto era desenvolver uma linguagem funcional e pronta a utilizar tivemos de limitar alguns aspetos da implementação.

Um dos principais problemas é a forma como são mostradas as unidades das dimensões complexas visto que em vez de verificar quais as operações utilizadas e com essas operações simplificar as unidades, estas são simplesmente concatenadas, ou seja caso se quisesse definir uma Dimensão através da operação Length/Length, o resultado vai ser “m/m”, isto caso a unidade base de Length seja m, em vez de 1 ou sem unidade associada.

Outra limitação é o facto de ao definir uma nova dimensão através de duas já existentes os únicos operadores possíveis de forma a criar a nova dimensão são a multiplicação e a divisão.

Apesar das limitações, consideramos ter conseguido concluir o principal objetivo do trabalho.

Contribuição:

- Daniel Gomes: 20%
- Gonçalo Pereira: 20%
- Miguel Marques: 20%
- Pedro Bastos: 20%
- Pedro Tavares: 20%