Robot Language (Rlang)

Uma linguagem compilada para facilitar programação de robôs

JULHO 2019

GRUPO:
Grupo 15

MEMBROS:
Tomás Costa - 89016
João Marques - 89234
João Silva - 88813
André Catarino - 89337

Índice

Compilador

Um compilador pode ser encarado como sendo um tradutor da linguagem fonte (i.e. da linguagem a compilar) para uma linguagem destino.
o compilador deve não só garantir a validade sintática do programa, como também a sua correção semântica no processo de tradução.

Proposta do projeto

Foi-nos proposto a conceção e definição de uma linguagem de programação (nomeadamente para a definição e manipulação de robots), na qual foi implementado em ANTLR4 a análise léxica e sintática e foram definidas as regras semânticas a aplicar na linguagem.
Consta neste relatório a descrição das instruções existentes, clarificando-se o significado das mesmas, com código exemplo para melhor compreensão.
Foi-nos também proposto a escolha de uma linguagem destino, onde se possa implementar a síntese do compilador, tendo sido a linguagem target escolhida C

Escolha do tema:

O objetivo deste tema era construir uma linguagem que seria usada na academia de verão em concursos como o ciber-rato, onde jovens com pouco ou nenhum conhecimento na área da programação aprendem como programar um pequeno robot a fazer um pequeno labirinto.

A proposta é construir uma linguagem de programação com as ferramentas necessárias para completar o desafio, e ao mesmo tempo ser mais simples e orientada a programadores inexperientes.

Análise de Requisitos:

Inicialmente, o desafio foi perceber de que forma esta nova linguagem poderia ser mais útil no sentido de ser mais intuitiva para quem nunca programou, para isto contactamos o professor Artur Pereira e fomos recolhendo algumas ideias importantes que eram importantes incluir na linguagem. Algo destacado pelo professor foi que a análise semântica deveria retornar erros mais explícitos e perceptíveis, pelo que nos focamos bastante nessa análise.

Foi importante desde início definir algumas instruções básicas da linguagem:

- Função move - que serve para mover o robô ajustando a velocidade das rodas, passando como argumento dois inteiros que servem para cada roda.
- Função stop - equivalente a move 0 0, serve para parar o robo.
- Ciclos while e for mais perceptíveis.
- Declaração de funções.
- Prints de funções.
- Instrução semelhante a Switch Case - One of ou Any of, para num bloco de instruções, ou executar apenas um (if ou else if) ou executar qualquer uma delas (ifs seguidos, pelo que pode executar várias)

Concepção da linguagem:

O objetivo com esta linguagem era tornar o código mais legível, tivemos um abordagem mais pitónica, com algumas mudanças e operações que o python não permite fazer. Inicialmente também queríamos ter a o fecho de blocos de código com idents e dedents em vez de chavetas, mas esta mudança foi repensada pois a indentação não é facilmente perceptível para iniciantes em programação.

-Principais características:

1. Boolean expressions com comparadores:

Problema:

If (a > 5 and a < 10)

Porquê usar "a" 2x ?

Ideia:

If (5 < a > 10)

If (a<b<c<d<e<f)

Simplificação de comparação de uma variável com várias outras.

2. Blocos de if Vs blocos de elseif:

Problema:

If (x) { … }

If (y) { … }

If (z) { … }

VS

If(x) { … }

elseIf(y) { … }

elseIf(z) { ... }

Um uso habitual da condição “se” é a seguinte:

“Se acontecer algo, faço x. Se acontecer aquilo, faço y”.

O problema aqui é que estamos habituados a usar o “se” exclusivamente, quando em programação não é necessariamente o que acontece (tal como em vários ifs seguidos).

Ideia:

```
any of:
case(x) ...
case(x) ...
else ...
end
```

Ou para if exclusivo:

```
one of:
case(x) ...
case(x) ...
else ...
end
```

any of irá executar todos os cases com uma condição verdadeira, enquanto que one of irá executar apenas o primeiro “case” que seja verdadeiro. Em ambos os casos se nenhum case for executado, será executado o bloco “else”.

3. Move e apply

Problema:

If(x) { move(10,10) }

Elseif(y) { move(10,20) }

Aplly(1)

Move e Apply são funções definidas previamente para manipular os robots, mas que são usadas pelos jovens que os programam. Move apenas será efectuado quando se fizer um Apply, porém é muito fácil esquecer de fazer o Apply e não perceber o problema.

Ideia:

move 10 20

move -50 50

Move é uma palavra reservada que recebe dois valores como antes(para parar o robot é aconselhado executar a instrução “stop" em vez de “move 0 0“). Não é necessário fazer apply, sendo automaticamente executado, prevenindo eventuais esquecimentos do programador.

Ciclos:

É usado como argumento dos loops as boolean expressions, que à exceção da utilização do true/false nos ciclos “for”, são utilizadas em conjunto com assignables que permitem a chamada de funções ou a realização de expressões, sendo que em caso de possibilidade será executado posteriormente o código seguinte.

Os breaks so poderam ser utilizados dentro dos loops, sendo gerado um erro na sua utilização fora desses mesmos loops

Para concluir um ciclo será necessário usar o endwhile ou endfor nos respetivos ciclos ou então usar só end.

Boolean comparators (usadas nas boolean expressions):

As boolean expressions, usadas por exemplo como argumento nos ciclos (while,for) ou na instrução choose (“one of” / “any of”), recorrem aos seguintes comparadores : (“<”,”>”,”<=”,”>=”, “=”, “different from”, “equals”, de maneira a ser o mais intuitivo possível para o programador inexperiente utilizar.)

Exemplos de utilização:

```
while (groundType() different from bn):
    //code
endwhile
```

```
for (integer i = 0, i < 10 ): // for i in range(0, 10)
   break
end

for (integer i = 10, i >= 0, -1 ): // for i in range(10, 0, -1)
   break
end
```

O segundo "for" usa uma "expression" opcional como argumento com o intuito de decrementar a variável "i", sendo possível como demonstrado no código exemplo a sua declaração nesse mesmo ciclo "for".

Assignments:

A instrução assignment pode ser feita de 2 maneiras distintas: Pode realizar-se a instrução **declaration,** de maneira a que se associa um id(nome) de uma variável ao seu tipo de dados e posteriormente atribui-se a essa variável o valor de um **assignable (**por exemplo o valor retornado por uma função,uma expressão, o valor de outra variável já declarada...), ou então esse "assign" a um **assignable** poderá ser feito diretamente a uma variável já declarada anteriormente.

Ex de código :

```
function hellofunc(integer i):
   return i
end
```

integer i = 0

integer a = 10 + hellofunc(i)

É declarado uma variável “a” do tipo inteiro, na qual será atribuído o valor de uma expressão que representa a soma entre um inteiro e o valor retornado por uma function call.

Function Definition and Function Calls:

Ao executar uma instrução do tipo “**function_def**” é necessário definir o seu nome sendo depois possível executar uma instrução do tipo **declaration (**declaração de uma nova variável associando-a a um determinado tipo de dados, sendo gerado erro caso seja declarado uma variável já previamente declarada**)** ou várias, dependendo do número de argumentos da função pretendidos pelo programador.

Será também gerado erro caso o programador tente definir uma função com o nome igual ao de outra já definida.

Dentro da função poderão ser realizadas diversas instruções, nas quais deve-se destacar a instrução “**return_statement**” que apenas poderá ser executada dentro de uma função sendo gerado o devido erro caso o mesmo não aconteça, assim como também não será possível retornar 2 tipos de dados diferentes numa função.

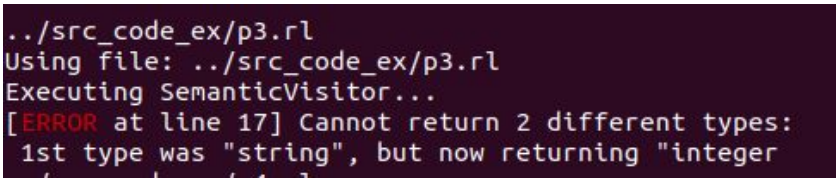
Ao executar uma instrução do tipo “**function_call**” é verificado primeiramente se existe alguma função já definida com o mesmo nome da que está a ser chamada, confirmando a sua existência, sendo também analisado se o número de argumentos da função chamada se encontra em conformidade com os da função definida, assim como verifica a conformidade entre o tipo de dados dos argumentos entre ambas as funções.

Código exemplo:

```
function hellofunc(integer i):
        i = i + 3
        print i
        return 'a'
        while( 2<i<10 and 1<2):
                integer a = 1
                one of:
                   case (1 equals 1):
                           break
                   case (2 equals 1):
                           break
                   else:
                           a = 2
                   end
                a = 1
        endwhile
```

return a

endfunction



Exemplo da definição de uma função em que estão ser retornados 2 tipos de dados diferentes, sendo gerado o devido erro explicitando claramente o problema.

Implementação em ANTLR4

Para o parser foram criados dois mapas, um que continha o nome das funções previamente definidas e o outro o tipo de retorno dessas funções. Também foi criada uma symbolTable que foi uma classe criada para guardar as variáveis.

```
@parser::members{
        public static SymbolTable symbolTable = new SymbolTable();

        public static Map<String,ArrayList<Type>> functionMap = new HashMap<>();

        public static Map<String,Type> returnTypes = new HashMap<>();
}
```

A base da gramática é semelhante a gramáticas lecionadas na disciplina de Compiladores, tem uma regra main que pode ou não ter um cabeçalho de imports, foi relevante incluir isto pois precisamos para o código exemplo a0.cpp incluir o cabeçalho CiberAV.h, nós adaptamos o cabeçalho e criamos os nossos próprios, com a extensão .rlh (RobotLangHeader), que permite definir funçoes e algumas variáveis antes de correr o código em si.

```
main: import_statement* chunk* EOF;

/* chunk: (instruction NEWLINE | instruction | NEWLINE)+; */
chunk: instruction+;

import_statement: IMPORT_WORD IDENTIFIER;

instruction: move
           | function_def
           | function_header
           | while_loop
           | for_loop
           | assignment
           | function_call
           | choose
           | if_statement
           | print
           | return_statement // need to verify if it is inside function definition
           | declaration
           | break_statement // need to see if this is inside brakable loop
           ;
```

Para a grámatica completa, consulte o ficheiro RobotLang.g4

Mudanças durante o desenvolvimento

- A ideia inicial era construir uma linguagem semelhante a python no que toca a identação, ou seja, identação obrigatória. Por uma questão de simplicidade, voltámos atrás nessa ponto. Para substituir a identação usamos palavras como “endwhile”, “endif”, etc ou simplesmente “end”. Como é óbvio identação continua a poder ser usada, apenas não é obrigatória.

- Tinhamos também pensado em implementar dicionários (chegou a estar na gramática) mas por falta de tempo decidimos não o fazer.

- Na definição de uma função inicialmente desenhámos a linguagem para ser obrigatório escrever o tipo de dados que retorna. Em vez disso decidimos dar um passo em frente e retirar essa regra. O tipo de retorno de uma função é calculado quando é encontrado um return (o tipo de retorno da função é o tipo da variável que está a ser retornada) .

- Acrescentamos também uma nova opção, a de fazer imports, usados sobretudo para declarar funções definidas noutros ficheiros (necessário para funções como beaconAngle()).

Regras semânticas:

Foi usado um visitor (SemantcVisitor.java) para a análise semântica. É aqui que são gerados os erros da linguagem desenvolvida referentes às instruções executadas pelo programador, sendo que estes erros têm de ser de fácil compreensão, já que a linguagem destina-se a programadores com poucas bases.

Algumas regras:

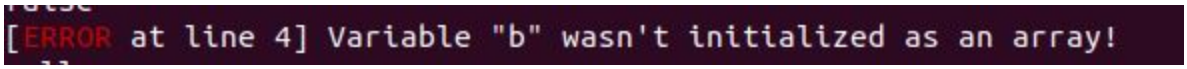
1. Declaração e inicialização de variáveis;
   -É possível declarar uma variável sem lhe atribuir um valor na mesma linha. Excepto se a variável a declarar for um Array.
   -Como na nossa linguagem a declaração de um array é igual à de uma variável normal (double a = [1,2]), ao fazer "double a" o código c gerado já excluiu a hipótese de a variável ser um array.
   -Tudo que retorna um valor pode ser dado como valor para guardar na variável desde que seja do mesmo tipo (outra variável, uma expressão matemática, uma chamada a uma função).

2. Instrução break;
   -Apenas pode ser usada dentro de loops como em outras linguagens

3. Instrução return;
   -Apenas pode ser usada dentro de funções.
   -Podem existir vários returns dentro da mesma função mas todos necessitam de retornar o mesmo tipo de dados.

4. Integer e real;
   -A linguagem aceita que um valor real aceite integer como valor mas não o contrário (ex: real a = [1.1, 2], real b = 4).

5. Operações com vectores;
   -Não é permitido operações com vectores nem com variáveis cujos valores sejam vectores (por simplificação).
   -É possível usar numa operação um valor de um vector como por exemplo a[3] a menos que a não seja um vector.



String Template:

Falar da aprendizagem deste metodo, falar um pouco dele e o que e possivel fazer e explicar o nosso caso da template para C.

Recorremos ao método *String Template* para gerar o código target (linguagem C).

Este método permite-nos definir templates, ou modelos de Strings, que recebem argumentos com os dados que apresentarão. Desta maneira, pudemos concentrar o nosso código de visitors em ler os valores necessários da nossa linguagem e colocá-los no sítio correto em C.

Aprender este método trouxe alguns desafios, nomeadamente devido à falta de documentação variada e abundante online. No entanto, depois de familiarizados com o conceito, e de estudada a documentação, conseguimos desenhar todos os templates que necessitaríamos e aplicá-los ao nosso programa.

Para isso, criámos o ficheiro *c_templaces.stg*, que contém um grupo de String Templates (ST). Este ficheiro é importado pelo visitor responsável por criar o output do programa. Ao fazer o visit a cada elemento, depois de ler os valores necessários nos tokens ou regras, era invocado um dos templates, sendo-lhe adicionados os dados referidos. Finalmente, era executado o método render() do objeto ST, que retorna uma String, a qual por sua vez era retornada pelo visitor, até que fosse impressa no terminal como instrução final.

Por exemplo, no caso de um loop for, usámos o template:

```
for_loop(start_condition, finish_condition, step, body) ::= <<
for(<start_condition>; <finish_condition>; <step>) {
<body>
}
>>
```

No nosso visitor fizemos:

```
for_loop = group.getInstanceOf("for_loop");
```

```
String start_condition = visit(ctx.start_condition);
String stop_condition = visit(ctx.stop_condition);
String body = visit(ctx.chunk());
String step = for_var + "+=" + visit(ctx.step);

for_loop.add("start_condition", start_condition);
for_loop.add("finish_condition", stop_condition);
for_loop.add("step", step);
for_loop.add("body", body);

String output = for_loop.render();
```

Desta maneira, a String output já terá a estrutura do código em C que queremos imprimir.

Linguagem Destino

Foi escolhida como linguagem destino: C . Fizemos esta escolha pois para além de ser de baixo nível, mas principalmente porque a comunicação do código tinha que gerado em C, e então esta conversão iria sempre ter que ser feita.
!!!!!Help here

Operações Possíveis

A nossa linguagem permite todas estas operações:

- Definir funções
- Ciclos while
- Ciclos For (c/ step ou s/ step)
- Ciclos ForEach
- Choose:
    - Any Of
    - One Of
- Condições If e Else
- Move
- Stop
- Prints
- Operações básicas entre expressões:
    - Adição
    - Subtração
    - Divisão
    - Multiplicação
- Declaração de variáveis
- Function calls
- Múltiplos comparadores de variáveis (p. Ex: 5 < a > 10)

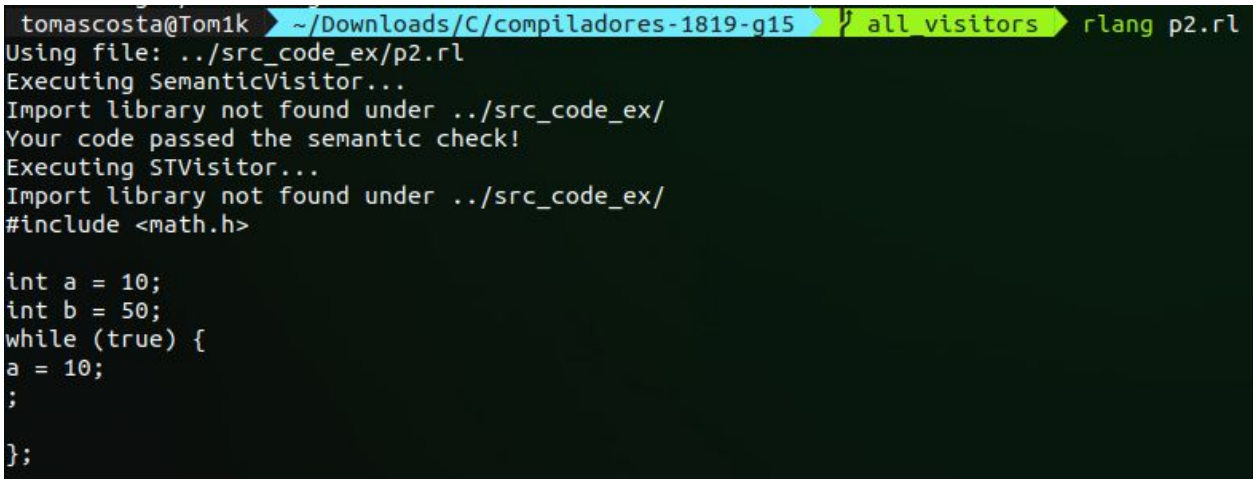
Tipos das variáveis:

- String
- Real
- Inteiro
- Arrays

Como correr a linguagem:

Recomendamos que corra o script bash rlanginstall para que posso criar um alias na sua terminal e consiga correr a linguagem com apenas o termo <rlang>
Como o exemplo em baixo:

Após a instalação, pode apenas correr: rlang <nome_ficheiro>
Nota: O nome do ficheiro tem que estar contido na pasta src_code_ex



Caso não seja possivel fazer a instalação, pode sempre correr o comando ./rlangbuild.sh seguido de um ./rlangrun.sh <nome_ficheiro>

Scripts Extra:

Para compilar e correr mais fácilmente o nosso código criámos 3 scripts extra:

1. **Rlangbuild:** limpa tudo o que tínhamos anteriormente compilado e gera novamente a gramática e os visitors.
2. **Rlang:** executa antlr-test para algum ficheiro, para provar se o ficheiro está de acordo com a nossa gramática. Pode receber uma argumento (<nome_do_ficheiro>) sendo que o ficheiro estará contido na pasta src_code_ex/. Caso nenhum argumento seja passado, o script é executado para todos os ficheiros contidos na pasta referida.
3. **Rlangrun:** executa antr4-run para a nossa gramática, seguindo o mesmo modelo do scritp ./rlang.sh - caso um argumento seja passado, é executado apenas para esse ficheiro, caso contrário, para todos os ficheiros em src_code_ex/.

Maiores obstáculos encontrados:

Os maiores obstáculos que enfrentamos o longo deste trabalho foram a dificuldade em trabalhar com alguns temas que nunca tínhamos usado na prática antes, como os String Templates, o uso de 2 visitors e perceber como se dividem e relacionam, e, sem dúvida, a complexidade associada à criação de uma linguagem de programação completa, a qual exigiu, apesar de bom planeamento prévio, muitas alterações ao longo do processo e a repetição de algumas das etapas associada a essas mudanças.

Participação do grupo:

Estimamos que a participação do grupo seja esta:

Tomás Costa: 28%
João Marques: 28%
João Silva: 28%
André Catarino: 16%
Luis Silva: 0%

Trabalho futuro:

No futuro gostávamos de implementar algo que não tivemos tempo para o fazer, e melhorar alguns aspetos que pensamos que a linguagem não permite fazer, tais como:

- Experimentar Identação
- Acrescentar mais operações e flexibilidade no uso de arrays(falta de tempo)
- Permitir concatenação de strings com strings/números (como em java)
- Declaração de dicionários (não eram pertinentes para o CiberAV)

Bibliografia:

https://theantlrguy.atlassian.net/wiki/discover/all-updates
Guiões resolvidos nas aulas práticas
Solução do Bloco 2 - Tomás Oliveira Silva