

# Relatório - Analisador Semântico

25 de Junho de 2023

Nome Completo	Matrícula
Lívia Pereira Ozório	201835011
Patrick Canto de Carvalho	201935026

# Sumário

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Analizador Semântico</b>	<b>3</b>
2.1	Estruturas de dados . . . . .	3
2.2	Tipos que podem existir . . . . .	3
2.3	Visitor . . . . .	3
<b>3</b>	<b>Como compilar</b>	<b>8</b>

# 1 Introdução

Este relatório tem como objetivo explicitar os pontos mais relevantes para a compreensão do processo de desenvolvimento de um analisador semântico para a linguagem *lang*, incluindo as decisões de projeto como as estratégias adotadas e estruturas de dados auxiliares.

## 2 Analisador Semântico

### 2.1 Estruturas de dados

No analisador semântico possuímos um visitor que é uma classe abstrata implementada no `ScopeVisitor`. O `ScopeVisitor` por sua vez possui algumas estruturas de dados que auxiliam no processamento do analisador semântico, sendo elas um hashmap de string e inteiro, uma lista de hashmap de string e string, 2 listas de lista de string, um hashmap de string e string e uma lista de string.

O primeiro hashmap armazena as funções e qual valor de escopo representa cada uma, isto facilita a busca nas listas quando precisamos encontrar os parâmetros, os retornos ou as variáveis de uma determinada função. A lista de hashmap representa as variáveis, nela é armazenado em cada posição da lista um escopo de variáveis de uma função e em cada escopo possuímos o nome da variável e qual o tipo relacionado a ela. As lista de lista de string representam os parâmetros e os retornos de uma função respectivamente, da mesma forma que as variáveis as posições da lista representam os escopos das funções e cada um possui a lista com os tipos dos parâmetros e retornos. O segundo hashmap armazena os dados criados armazenando os tipos de cada atributo que o dado possui. E por último temos uma lista que armazena os tipos retornados por cada linha do código.

### 2.2 Tipos que podem existir

Durante a execução do analisador semântico é adicionado alguns tipos na lista de tipos, podendo estes serem:

- Int - que representa números do conjunto dos inteiros
- Float - que representa números do conjunto dos reais
- Bool - que representa valores booleanos, ou seja, verdadeiro ou falso
- Char - que representa caracteres
- Error - que representa que o erro aconteceu anteriormente
- CMD - que representa que o procedimento não possui um tipo mas está tudo certo
- Um D existente no hashmap de dados - que representa um objeto do tipo dado D
- T[ ], com T pertencente a um dos outros tipos - que representa um vetor do tipo T

### 2.3 Visitor

- visitor(Prog p) - O visitor do Prog ele chama o visitor do `DataList` e do `DataFunc`;

- `visitor(FuncList f)` - O visitor do `FuncList` ele cria os escopos adicionando a função no hashmap de escopo e criando o valor de escopo de cada função, preenche a lista de parâmetros e retornos de cada uma também. Além disto chama e processa o visitor de cada `Func` contida no `FuncList` adicionando `Error` na lista de tipos caso alguma função possua erro e `cmd` caso não possua;
- `visitor(DataList d)` - O visitor do `DataList` apenas chama o visitor do `Data` para cada dado contido na lista;
- `visitor(Data d)` - O visitor do `Data` apenas preenche o hashmap de dados para ser usado futuramente em novas funções;
- `visitor(func f)` - O visitor da `func` ele atualiza qual nível do escopo estamos analisando e depois chama o visitor do `CmdList`;
- `visitor(CmdList c)` - O visitor do `CmdList` ele chama e processa o visitor de cada comando contido na lista. Caso tenha algum comando com erro adiciona `Error` a lista de tipos caso contrário adiciona `cmd`;
- `visitor(Print p)` - O visitor do `Print` chama e processa visitor da expressão a ser impressa verificando se a expressão é do tipo erro, caso seja o `print` adiciona um erro na lista de tipos caso contrário adiciona `cmd`;
- `visitor(Type t)` - O visitor do `Type` ele confere qual o tipo adicionar na lista de tipos. Caso o tipo enviado não pertença aos tipos fundamentais da linguagem nem aos dados criados a função gera um erro na tela falando que o tipo não existe e adiciona `Error` à lista de tipos;
- `visitor(Add a)` - O visitor do `Add` ele chama e processa os tipos das duas expressões. E então conferem se pelo menos uma das duas expressões tem tipo `Error` caso sim a função adiciona o tipo `Error` na lista de tipos. Caso não, confere se as expressões são `FloatAst`, `FloatAst` ou `Int`, `Int`, caso sim adiciona o tipo da operação (`FloatAst` ou `Int`) na lista de tipos. Caso não, adiciona um tipo erro na lista de tipos e lança um erro na tela falando que os tipos das expressões dadas não podem ser somadas;
- `visitor(Mul a)` - O visitor do `Mul` chama e processa os tipos das duas expressões. E então conferem se pelo menos uma das duas expressões tem tipo `Error` caso sim a função adiciona o tipo `Error` na lista de tipos. Caso não, confere se as expressões são `FloatAst`, `FloatAst` ou `Int`, `Int`, caso sim adiciona o tipo da operação (`FloatAst` ou `Int`) na lista de tipos. Caso não, adiciona um tipo erro na lista de tipos e lança um erro na tela falando que os tipos das expressões dadas não podem ser multiplicadas;
- `visito(Rest a)` - O visitor do `Rest` ele chama e processa os tipos das duas expressões. E então conferem se pelo menos uma das duas expressões tem tipo `Error` caso sim a função adiciona o tipo `Error` na lista de tipos. Caso não, confere se as expressões são `Int`, `Int`, caso sim adiciona o tipo da operação (`Int`) na lista de tipos. Caso não, adiciona um tipo erro na lista de tipos e lança um erro na tela falando que os tipos das expressões dadas não podem ter resto;
- `visitor(Div a)` - O visitor do `Div` ele chama e processa os tipos das duas expressões. E então conferem se pelo menos uma das duas expressões tem tipo `Error` caso sim a função adiciona o tipo `Error` na lista de tipos. Caso não, confere se as expressões são `FloatAst`, `FloatAst` ou `Int`, `Int`, caso sim adiciona o tipo da operação (`FloatAst` ou `Int`) na lista de

tipos. Caso não, adiciona um tipo erro na lista de tipos e lança um erro na tela falando que os tipos das expressões dadas não podem ser divididas;

- `visitor(Sub a)` - O visitor do `Sub` chama e processa os tipos das duas expressões. E então conferem se pelo menos uma das duas expressões tem tipo `Error` caso sim a função adiciona o tipo `Error` na lista de tipos. Caso não, confere se as expressões são `FloatAst`, `FloatAst` ou `Int`, `Int`, caso sim adiciona o tipo da operação (`FloatAst` ou `Int`) na lista de tipos. Caso não, adiciona um tipo erro na lista de tipos e lança um erro na tela falando que os tipos das expressões dadas não podem ser subtraídas;
- `visitor(SubUni a)` - O visitor do `SubUni` chama e processa o tipo da expressão. E então conferem se a expressão tem tipo `Error` caso sim a função adiciona o tipo `Error` na lista de tipos. Caso não, confere se a expressão é `FloatAst` ou `Int`, caso sim adiciona o tipo da operação (`FloatAst` ou `Int`) na lista de tipos. Caso não, adiciona um tipo erro na lista de tipos e lança um erro na tela falando que o tipo da expressão dada não podem ser operado;
- `visitor(Neg a)` - O visitor do `Neg` ele chama e processa o tipo da expressão. E então conferem se a expressão tem tipo `Error` caso sim a função adiciona o tipo `Error` na lista de tipos. Caso não, confere se a expressão é `Bool`, caso sim adiciona o tipo da operação (`Bool`) na lista de tipos. Caso não, adiciona um tipo erro na lista de tipos e lança um erro na tela falando que o tipo da expressão dada não podem ser operado;
- `visitor(And a)` - O visitor do `And` chama e processa os tipos das duas expressões. E então conferem se pelo menos uma das duas expressões tem tipo `Error` caso sim a função adiciona o tipo `Error` na lista de tipos. Caso não, confere se as expressões são `Bool`, `Bool` caso sim adiciona o tipo da operação (`Bool`) na lista de tipos. Caso não, adiciona um tipo erro na lista de tipos e lança um erro na tela falando que os tipos das expressões dadas não podem ser operadas;
- `visitor(GreaterThan a)` - O visitor do `GreatherThan` ele chama e processa os tipos das duas expressões. E então conferem se pelo menos uma das duas expressões tem tipo `Error` caso sim a função adiciona o tipo `Error` na lista de tipos. Caso não, confere se as expressões são `FloatAst`, `FloatAst` ou `Int`, `Int`, caso sim adiciona o tipo da operação (`Bool`) na lista de tipos. Caso não, adiciona um tipo erro na lista de tipos e lança um erro na tela falando que os tipos das expressões dadas não podem ser operadas;
- `visitor(LessThan a)` - O visitor do `LessThan` chama e processa os tipos das duas expressões. E então conferem se pelo menos uma das duas expressões tem tipo `Error` caso sim a função adiciona o tipo `Error` na lista de tipos. Caso não, confere se as expressões são `FloatAst`, `FloatAst` ou `Int`, `Int`, caso sim adiciona o tipo da operação (`Bool`) na lista de tipos. Caso não, adiciona um tipo erro na lista de tipos e lança um erro na tela falando que os tipos das expressões dadas não podem ser operadas;
- `visitor(Diff a)` - O visitor do `Diff` ele chama e processa os tipos das duas expressões. E então conferem se pelo menos uma das duas expressões tem tipo `Error` caso sim a função adiciona o tipo `Error` na lista de tipos. Caso não, confere se as expressões possuem tipos iguais, caso sim adiciona o tipo da operação (`Bool`) na lista de tipos. Caso não, adiciona um tipo erro na lista de tipos e lança um erro na tela falando que os tipos das expressões não gera uma comparação válida;
- `visitor(Eq a)` - O visitor do `Eq` ele chama e processa os tipos das duas expressões. E então conferem se pelo menos uma das duas expressões tem tipo `Error` caso sim a função

adiciona o tipo Error na lista de tipos. Caso não, confere se as expressões possuem tipos iguais, caso sim adiciona o tipo da operação (Bool) na lista de tipos. Caso não, adiciona um tipo erro na lista de tipos e lança um erro na tela falando que os tipos das expressões não gera uma comparação válida;

- visitor(Int i) - O visitor do Int ele adiciona o valor Int na lista de tipos;
- visitor(FloatAst i) - O visitor do FloatAst ele adiciona o valor FloatAst na lista de tipos;
- visitor(Bool i) - O visitor do Bool ele adiciona o valor Bool na lista de tipos;
- visitor(iterate i) - O visitor do Iterate ele chama e processa o visitor da expressão e confere se a expressão é do tipo Int ou Error, caso não seja a função lança um erro na tela para o usuário e falando que a expressão deve ser um Int. Então chama-se e processa o visitor do then. Então é avaliado o que colocar na lista de tipos, se a expressão for diferente de Int adiciona-se Error, caso seja igual a Int mas tenham algum erro no then então adiciona Error também. Apenas se não tiver nenhum erro no then e a expressão for Int que a função adiciona cmd;
- visitor(If i) - O visitor do If ele chama e processa o visitor da expressão e confere se a expressão é do tipo Bool ou Error, caso não seja a função lança um erro na tela para o usuário e falando que a expressão deve ser um Bool. Então chama-se e processa o visitor do then. Se o else existir chama e processa o visitor do else. Então é avaliado o que colocar na lista de tipos, se a expressão for diferente de Bool adiciona-se Error, caso seja igual a Bool mas tenham algum erro no then ou no else então adiciona Error também. Apenas se não tiver nenhum erro no then, no else e a expressão for Bool que a função adiciona cmd;
- visitor(Attr a) - O visitor da atribuição ele inicialmente chama e processa o visitor da expressão. Então é visto se a variável a qual se quer atribuir valor é uma variável simples ou se é um atributo de dado ou se é um vetor. Caso seja uma variável simples confere se a variável já existe caso sim o valor atribuído a ela deve ter o mesmo tipo do anterior e caso descumpra essa exigência é gerado um erro na tela e adicionado Error na lista de tipos. Caso a variável não exista adiciona ela na lista de variáveis no seu devido escopo e depois adiciona cmd na lista de tipos. Já no caso de ser um vetor ou um atributo de dado, é chamado e processado o visitor do LValue. Então se o tipo da variável e o tipo da expressão são diferentes e nenhum dos dois é do tipo Error é gerado uma mensagem na tela falando que não se pode atribuir e adiciona Error na lista de tipos. Caso a variável ou a expressão forem do tipo Error adiciona Error também a lista de tipos. Caso esteja tudo certo é adicionado cmd;
- visitor(LValue l) - O visitor do LValue começa avaliando se o nome dado é um nome simples ou um nome de um acesso de atributo de dado ou nome de um acesso a uma posição de vetor. Caso seja um nome simples apenas procura o nome da variável em seu escopo e pega o tipo, se a variável existir no escopo adiciona o tipo na lista de tipos, caso não exista retorna uma mensagem na tela falando que a variável não foi inicializada, logo não possui um tipo ainda. Já se for um acesso a uma posição do vetor é chamado e processado o visitor da expressão, depois é visto se a expressão é do tipo Int, se for é chamado e processado o visitor do LValue para pegar o tipo do vetor, pegando este tipo é visualizado se o tipo é um Error se for só é adicionado Error na lista de tipos, caso contrário é adicionado o tipo do vetor. No caso da expressão ser do tipo Error é adicionado Error na lista de tipos, e no caso de não ser nem Int nem Error então é gerado uma mensagem na tela dizendo que a expressão esperava tipo Int. E o último caso é o

de ser um acesso a um atributo de um dado, neste caso é chamado e processado o visitor do LValue que vai me trazer o tipo do dado a qual o atributo pertence. Se o tipo for um Error é adicionado um Error na lista de tipos, caso não seja procura o tipo do atributo no hashmap de dados e então encontrando adiciona o tipo do atributo na lista de tipos, caso não encontre a função escreve na tela que o atributo procurado não existe naquele dado e adiciona Error na lista de tipos;

- visitor(ID i) - O visitor do ID ele adiciona na lista de tipos o tipo de uma determinada variável consultando o lista de variáveis do escopo;
- visitor(New n) - O visitor do New ele primeiro confere se estamos querendo criar um vetor ou apenas um tipo dado mesmo. Se for um vetor é chamado e processado o visitor da expressão, então é conferido se a expressão é do tipo Int ou Error. Se não for nenhum dos dois gera uma mensagem na tela dizendo que a expressão esperava um Int e adiciona Error na lista de tipos, caso seja um Error apenas adiciona Error na lista de tipos e se for Int é chamado e processado o visitor do Type que confere qual o tipo que desejamos criar, então adiciona este tipo na lista de tipos. Já no caso de não ser um vetor só chama e processa o visitor do Type e adiciona o valor do tipo na lista de tipos;
- visitor(Read r) - O visitor do Read apenas adiciona a variável lida na lista de variáveis em seu devido escopo e adiciona cmd na lista de tipos;
- visitor(CallFunction c) - O visitor do CallFunction primeiro avalia se a função existe no hashmap de funções, se não existe retorna um erro na tela informando que a função não foi definida e adiciona Error na lista de tipos. Caso ela exista, é conferido se o número de parâmetros passados é o mesmo número de parâmetros esperados. Caso não seja retorna uma mensagem na tela informando que o número de parâmetros está incorreto e adiciona Error na lista de tipos. No caso de o número está correto é chamado e processado o visitor da expressão para cada parâmetro passado e avaliado se o tipo passado corresponde ao tipo esperado. Caso algum parâmetro não seja do tipo esperado nem do tipo Error é retornado na tela uma mensagem informando que a lista de tipos recebe Error. Se algum for do tipo Error é adicionado um Error na lista de tipos também. Caso esteja tudo certo é passado para o retorno que confere também se o número esperado é o recebido, se for confere se a variável já existe se já confere se os tipos batem, caso batam adiciona cmd na lista de tipos, no caso de não bater é colocado o tipo Error e enviado uma mensagem avisando. Se a variável ainda não exista no escopo adiciona na lista de variáveis no escopo certo e coloca cmd na lista de variáveis também. Já caso o número de retornos seja diferente do esperado então gera uma mensagem avisando e coloca-se Error na lista de tipos;
- visitor(CallFunctionVet c) - O visitor do CallFunctionVet primeiro avalia se a função existe no hashmap de funções, se não existe retorna um erro na tela informando que a função não foi definida e adiciona Error na lista de tipos. Caso ela exista, é conferido se o número de parâmetros passados é o mesmo número de parâmetros esperados. Caso não seja retorna uma mensagem na tela informando que o número de parâmetros está incorreto e adiciona Error na lista de tipos. No caso de o número está correto é chamado e processado o visitor da expressão para cada parâmetro passado e avaliado se o tipo passado corresponde ao tipo esperado. Caso algum parâmetro não seja do tipo esperado nem do tipo Error é retornado na tela uma mensagem informando que a lista de tipos recebe Error. Se algum for do tipo Error é adicionado um Error na lista de tipos também. Caso esteja tudo certo é passado para o retorno. Então é chamado e processado o tipo da expressão e confere se a expressão é do tipo Int ou Error caso não seja, é adicionado

uma mensagem na tela informando que se espera um Int e coloca Error na lista de tipos, se for um Error somente adiciona Error na lista de tipos, já se for um Int adiciona o tipo do retorno corresponde a posição passada pela expressão na lista de tipos;

### 3 Como compilar

Para compilar e executar usando o programa Make, basta executar o seguinte comando no diretório do projeto:

```
make run <nome do arquivo lang>
```

Alternativamente, executar os seguintes comandos para compilar :

```
javac -cp .:antlr-4.8-complete.jar Interpretador.java  
java -jar antlr-4.8-complete.jar parser/lang.g4
```

E o seguinte para executar:

```
java -cp .:antlr-4.8-complete.jar Interpretador <nome do arquivo lang>
```

Para realizar os testes presentes na pasta testes/semantica, executar:

```
./teste.sh
```

O programa deverá ter as suas saídas apresentadas no terminal.