Análisador Léxico - Tradutores

Marcus Vinícius da Silva Borges[130032891]

Departamento de Ciência da Computação, Universidade de Brasília
130032891@aluno.unb.br
cic.unb.br

1 Motivação

Os conceitos de teoria de conjuntos são muito importantes na matemática contemporânea [4], mas não se limitam somente a essa área. Esses conceitos são base dentro da Ciência da Computação já que a Teoria da Computação tem bastante de suas definições baseadas em conjuntos como, por exemplo, as máquinas de Turing e os autômatos finitos.

Este trabalho propõe a construção de um compilador para a linguagem Bemolang, que é um subconjunto da linguagem C com a adição de primitivas que dão suporte para o tratamento de conjuntos matemáticos. Nessa primeira etapa do trabalho o foco é uma dar uma breve descrição sobre a linguagem e da implementação de seu analisador léxico.

2 Primitivas da Linguagem

A Bemolang possui algumas das funcionalidades básicas da linguagem C, conforme pode ser visto na gramática da linguagem no Anexo A. As novas primitivas da linguagem, responsáveis por facilitar o trabalho com conjuntos, estão brevemente descritas nas seções 2.1, 2.2 e 2.3.

2.1 Tipos

O tipo `set` representa um conjunto e sua declaração não necessita de um tipo atribuído ao conjunto. O tipo `elem` é uma variável polimórfica, ou seja, pode assumir a forma dos outros três tipos disponíveis na linguagem: `set`, `int` ou `float`. Além disso, foi adicionada a constante `EMPTY` na linguagem que representa o conjunto vazio.

2.2 Operadores

Os operadores responsáveis pela verificação de pertinência são o `in` e o `exists`, respectivamente. Os operadores responsáveis pela adição e remoção de um elemento em um conjunto são o `add` e o `remove`, respectivamente. Por fim, o operador `is_set` verifica se uma variável polimórfica do tipo `elem` é um `set`.

2.3 Comandos

Para facilitar a iteração de elementos de um `set` será implementado o comando `forall`. Os comandos responsáveis por lidar com entrada e saída da linguagem são: `read`, `write` e `writeIn`.

3 Análise Léxica

Para implementação do analisador léxico foi utilizado a ferramenta Flex [2]. Dentre os arquivos desta entrega, o bemolang.l é o arquivo utilizado pelo Flex para criar o analisador léxico, onde são definidas as regras de casamento de padrão entre os caracteres de um programa e a geração de tokens para cada lexema encontrado.

As funções `count` e `handle_unrecognized_symbol` foram as principais adições nesse processo. A `count` é responsável pela contagem das linhas e colunas de um programa, enquanto que a `handle_unrecognized_symbol` é chamada sempre que um símbolo não definido na linguagem é encontrado para gerar uma mensagem de erro informativa para o usuário.

4 Testes

Os arquivos de testes, encontrados na pasta `tests`, estão divididos entre os que contém apenas símbolos válidos para a Bemolang, `valid_symbols_1.bml` e `valid_symbols_2.bml`, e entre os que possuem símbolos inválidos para a linguagem, `invalid_symbols_1.bml` e `invalid_symbols_2.bml`.

No arquivo `/tests/invalid_symbols_1.bml` é esperado que o análisador léxico aponte erros nas linhas 10 e 11 sobre os caracteres `%` e `^`, respectivamente. Já no arquivo `/tests/invalid_symbols_2.bml` é esperado que os erros apontados sejam sobre os caracteres `@` na linha 2 e `^` e `#` na linha 4.

5 Instruções de uso

Dentre os arquivos entregues existe um *Makefile* para facilitar o uso do analisador léxico. Caso esteja utilizando a distribuição Linux Ubuntu/Debian é possível rodar os seguinte comandos no terminal para instalar o Flex e compilar o analisador léxico:

```
sudo make install
make flex
```

A saída esperada são dois arquivos: `lex.yy.c` e `bemolang.out`. Para fazer a análise léxica de um arquivo de teste basta rodar o comando `make run` no terminal passando o argumento `P` com o caminho de um dos arquivos de teste, como mostrado abaixo:

```
make run P="./tests/invalid_symbols_2.bml"
```

A Gramática da Linguagem Bemolang

Abaixo é a apresentada a gramática da linguagem Bemolang na forma *Backus-Naur*. Para definição da Bemolang foi utilizado a gramática da linguagem C definida pelo Professor Ajay Gupta [3] da *Purdue University* também na forma *Backus-Naur*.

⟨*translation-unit*⟩ ::= ⟨*external-declaration*⟩*

⟨*external-declaration*⟩ ::= ⟨*function-definition*⟩
| ⟨*declaration*⟩

⟨*function-definition*⟩ ::= ⟨*type-specifier*⟩ ⟨*declarator*⟩ ⟨*compound-statement*⟩

⟨*type-specifier*⟩ ::= int
| float
| elem
| set

⟨*declarator*⟩ ::= ⟨*identifier*⟩ (⟨*parameter-list*⟩*)

⟨*parameter-list*⟩ ::= ⟨*parameter-declaration*⟩
| ⟨*parameter-list*⟩ , ⟨*parameter-list*⟩

⟨*parameter-declaration*⟩ ::= ⟨*type-specifier*⟩ ⟨*identifier*⟩

⟨*conditional-expression*⟩ ::= ⟨*logical-or-expression*⟩

⟨*logical-or-expression*⟩ ::= ⟨*logical-and-expression*⟩
| ⟨*logical-or-expression*⟩ || ⟨*logical-and-expression*⟩

⟨*logical-and-expression*⟩ ::= ⟨*equality-expression*⟩
| ⟨*logical-and-expression*⟩ && ⟨*equality-expression*⟩

⟨*equality-expression*⟩ ::= ⟨*relational-expression*⟩
| ⟨*equality-expression*⟩ == ⟨*relational-expression*⟩
| ⟨*equality-expression*⟩ != ⟨*relational-expression*⟩

⟨*relational-expression*⟩ ::= ⟨*additive-expression*⟩
| ⟨*relational-expression*⟩ ⟨ <⟨*additive-expression*⟩

| | ⟨*relational-expression*⟩ > ⟨*additive-expression*⟩
| | ⟨*relational-expression*⟩ ⟨= <*additive-expression*⟩
| | ⟨*relational-expression*⟩ >= ⟨*additive-expression*⟩

⟨*additive-expression*⟩ ::= ⟨*multiplicative-expression*⟩
| | ⟨*additive-expression*⟩ + ⟨*multiplicative-expression*⟩
| | ⟨*additive-expression*⟩ - ⟨*multiplicative-expression*⟩

⟨*multiplicative-expression*⟩ ::= ⟨*unary-expression*⟩
| | ⟨*multiplicative-expression*⟩ * ⟨*unary-expression*⟩
| | ⟨*multiplicative-expression*⟩ / ⟨*unary-expression*⟩

⟨*unary-expression*⟩ ::= ⟨*postfix-expression*⟩
| | ⟨*unary-operator*⟩ ⟨*unary-expression*⟩

⟨*postfix-expression*⟩ ::= ⟨*primary-expression*⟩
| | ⟨*postfix-expression*⟩ (⟨*assignment-expression*⟩*)

⟨*primary-expression*⟩ ::= ⟨*identifier*⟩
| | ⟨*constant*⟩
| | (⟨*expression*⟩)

⟨*constant*⟩ ::= ⟨*integer-constant*⟩
| | ⟨*character-constant*⟩
| | ⟨*floating-constant*⟩
| | ⟨*empty-constant*⟩
| | ⟨*string*⟩

⟨*expression*⟩ ::= ⟨*assignment-expression*⟩
| | ⟨*type-check-expression*⟩
| | ⟨*expression*⟩ , ⟨*assignment-expression*⟩

⟨*assignment-expression*⟩ ::= ⟨*conditional-expression*⟩
| | ⟨*unary-expression*⟩ = ⟨*assignment-expression*⟩

⟨*unary-operator*⟩ ::= +
| | -
| | !

⟨*compound-statement*⟩ ::= ⟨*declaration*⟩* ⟨*statement*⟩*

⟨*declaration*⟩ ::= ⟨*type-qualifier*⟩ ⟨*identifier*⟩ ;

⟨*statement*⟩ ::= ⟨*expression-statement*⟩
| ⟨*compound-statement*⟩
| ⟨*selection-statement*⟩
| ⟨*iteration-statement*⟩
| ⟨*inclusion-statement*⟩ ;
| ⟨*removal-statement*⟩
| ⟨*io-statement*⟩
| ⟨*jump-statement*⟩

⟨*expression-statement*⟩ ::= ⟨*expression*⟩? ;

⟨*membership-expression*⟩ ::= ⟨*expression*⟩ in ⟨*expression*⟩

⟨*type-check-expression*⟩ ::= is_set (⟨*identifier*⟩)

⟨*selection-statement*⟩ ::= if (⟨*expression*⟩) ⟨*statement*⟩
| if (⟨*expression*⟩) ⟨*statement*⟩ else ⟨*statement*⟩
| ⟨*existence-statement*⟩

⟨*existence-statement*⟩ ::= exists (⟨*membership-expression*⟩)
| exists (⟨*identifier*⟩ in ⟨*identifier*⟩)

⟨*iteration-statement*⟩ ::= for (⟨*expression*⟩? ; ⟨*expression*⟩? ; ⟨*expression*⟩?)
⟨*statement*⟩
| forall (⟨*membership-expression*⟩) ⟨*statement*⟩

⟨*inclusion-statement*⟩ ::= ⟨*inclusion-expression*⟩ ;

⟨*inclusion-expression*⟩ ::= add (⟨*membership-expression*⟩)
| add (⟨*expression*⟩ in ⟨*inclusion-statement*⟩)

⟨*removal-statement*⟩ ::= remove (⟨*membership-expression*⟩) ;

⟨*io-statement*⟩ ::= write (⟨*expression*⟩) ;
| writeln (⟨*expression*⟩) ;
| read (⟨*identifier*⟩) ;

⟨*jump-statement*⟩ ::= return ⟨*expression*⟩? ;

Referências

1. Alfred V. Aho et al. *Compilers: Principles, Techniques, and Tools*. Addison Wesley, 2nd edition, 2006.
2. Flex manual. Acessado em 17 de fevereiro de 2021 às 20:53 https://westes.github.io/flex/manual/
3. The syntax of C in Backus-Naur Form. Acessado em 17 de fevereiro de 2021 à 20:59 https://cs.wmich.edu/ gupta/teaching/cs4850/sumII06/The%20syntax%20of%20C%20in%20Backus-Naur%20form.htm
4. Domingues, H.H. and Iezzi, G. *ALGEBRA MODERNA: EDIÇAO REFORMULADA*, ISBN 9788535704013, 4ª edição, 2003.