

Analizador Léxico e Sintático

Vitor Fernandes Dullens - 16/0148260

Universidade de Brasília

1 Introdução

Este relatório abordará sobre as primeiras etapas do processo de criação de um tradutor, que consiste na implementação do analisador léxico e sintático para uma linguagem específica, como apresentado no livro base da disciplina [ALSU07]. Neste documento serão descritos com detalhes a motivação por trás desta implementação, assim como a descrição das análises e instruções para a compilação e execução do programa. Também será apresentada uma descrição da linguagem à ser analisada.

2 Motivação

Na linguagem C, muitas vezes sentimos falta de operações que facilitam a utilização de conjuntos, como existe em outras linguagens de mais alto nível. Dito isso, afim de facilitar essas operações, uma implementação de uma nova primitiva de dados para conjuntos foi proposta dentro da linguagem C - **set**, assim como operações para a mesma - **add**, **remove**, entre outras.

Abaixo segue um exemplo de código na nova linguagem proposta.

```
int main() {
    set s;
    s = EMPTY;

    add(1 in add(2 in add(3 in s)));
    /* s = (1, 2, 3) */

    remove(1 in s);
    /* s = (2, 3) */
}
```

Além de operações com conjuntos, também foi adicionado um tipo polimórfico - **elem** que facilita, também, no uso de conjuntos. Mais detalhes sobre a linguagem podem ser encontrados no apêndice A.

3 Descrição da análise léxica

Para a implementação do analisador léxico foi utilizado o programa *Fast Lexical Analyzer Generator* - FLEX [Est], que consiste em uma ferramenta geradora de programas que reconhecem padrões léxicos em textos.

No arquivo de nome `lexical.1` é possível visualizar as regras léxicas. Para o tratamento das mesmas são declaradas expressões regulares (regex) que às identificam e após essas declarações existe uma sequência de ações que o analisador executa ao encontrar uma regra. Além das regras e ações, no arquivo `lexical.1` também foram definidas funções para leitura de arquivo, assim como três variáveis `int line`, `int column` e `int errors`, que representam, respectivamente, a linha em qual está acontecendo a ação, a coluna e o número de erros encontrados até o momento.

O analisador léxico também é responsável pela atribuição dos tokens que serão utilizados durante a análise sintática. Os tokens são declarados como uma `struct Token` que possui os parâmetros `char* body` que consiste no token propriamente dito, `int line` e `int column` que, respectivamente, representam a linha e coluna daquele token. Essa `struct` é declarada no arquivo do analisador sintático, e pode ser utilizada no analisador léxico utilizando a variável global `yylval`.

4 Descrição da análise sintática

Para a implementação do analisador sintático foi utilizado o programa Bison [DS], que consiste em um gerador de parser que utiliza de uma gramática livre do contexto para criar uma derivação LR. Para este trabalho foi utilizada a flag `%define lr.type canonical-lr` para que a derivação realizada seja a LR(1) canônica.

No arquivo de nome `syntax.y` é possível visualizar a gramática presente no apêndice A com algumas modificações para se encaixar na sintaxe do próprio Bison.

4.1 Árvore Sintática

Pelo modo como é feito o parser pelo próprio Bison, é possível contruir uma árvore sintática abstrata. Para isso, cada não terminal agora é um nó da árvore, e cada terminal é um símbolo, assim como será apresentado na Seção 4.2. Cada um desses nós, consiste em uma estrutura que armazena o símbolo(terminal), qual a regra que foi vista, o nó próximo e o filho. Com essas informações é possível realizar uma DFS a partir da raiz apresentando cada um dos nós que foram visitados e todos os detalhes necessários.

A Figura 1 representa a árvore gerada para um programa de entrada simples.

4.2 Tabela de Símbolos

Durante a passagem do analisador sintático, ele também é responsável por salvar os símbolos, afim de utiliza-los futuramente na análise semântica.

Cada símbolo consiste em uma estrutura que armazena dados que podem ser úteis na próxima etapa do projeto, como o ID da variável ou função declarada, suas respectivas linha e coluna, tipo e se é uma função ou apenas variável.

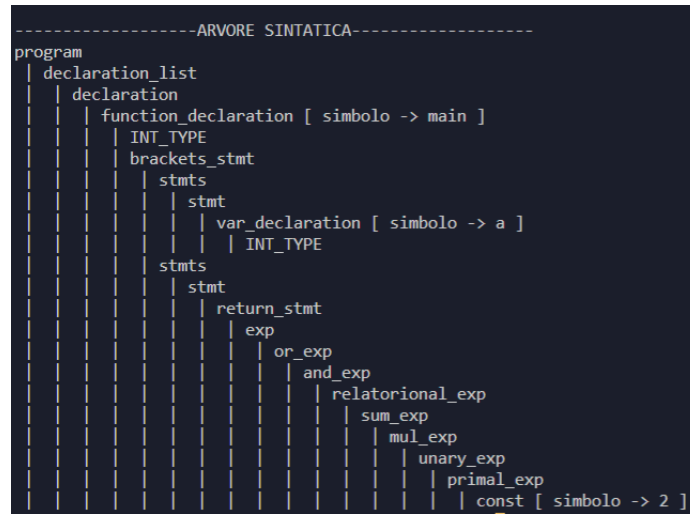


Figura 1. Árvore Sintática para uma entrada simples

TABELA DE SIMBOLOS			
IDENTIFICADOR	LINHA:COLUNA	TYPE	EH FUNCAO?
val	2:9	SET	0
el	12:14	FLOAT	0
a	16:16	SET	0
subsum	1:5	SET	1

Acima é possível visualizar a tabela de símbolos para o arquivo `success2.c`.

5 Descrição dos arquivos testes

Os testes se encontram na pasta `tests/`, os arquivos `success1.c` e `success2.c`, são testes que contém código correto, já os arquivos `error1.c` e `error2.c` contém códigos incorretos, sendo os seus erros, respectivamente:

1. SYNTAX ERROR 2:13 - syntax error, unexpected ';'
 - LEXICAL ERROR 3:20 - Unidentified character: #
 - LEXICAL ERROR 4:9 - Incorrect token format: 321varivel
 - SYNTAX ERROR 4:9 - syntax error, unexpected ';', expecting ID
 - SYNTAX ERROR 9:5 - syntax error, unexpected INT_TYPE, expecting ';'

```

2. SYNTAX ERROR 5:21 - syntax error, unexpected ';',
   expecting ',', ' or ')'
SYNTAX ERROR 12:11 - syntax error, unexpected ';'

```

Basicamente, como é possível observar acima, os erros são apresentados e possuem a linha e coluna na qual eles pertencem. O primeiro arquivo de erro - `error1.c` - possui tanto erros léxicos, quando erros sintáticos, já o segundo arquivo - `error2.c` - possui apenas erros sintáticos. Quando um arquivo possui erros não são apresentadas a tabela de símbolos nem a árvore sintática. Além disso, cada erro é apresentado seguido de uma descrição do mesmo.

6 Instruções para compilação e execução do programa

O programa foi criado e testado em um sistema operacional Linux - Ubuntu 20.04.1 LTS. É necessária a instalação do FLEX [Est] e do BISON [DS] para a compilação do programa. Ao executar o programa também deverá ser passado o arquivo que será analisado.

Comandos para compilação e execução:

```

$ bison syntax.y
$ flex lexical.l
$ gcc syntax.tab.c lex.yy.c tabela.c arvore.c
$ ./a.out tests/<nome-arquivo>.c

```

Outra alternativa para facilitar a compilação seria utilizar o comando `make`.

Referências

- [ALSU07] A.V. Aho, M.S. Lam, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques, & Tools*. Pearson/Addison Wesley, 2nd edition, 2007.
- [DS] C. Donnelly and R. Stallman. Bison - the yacc-compatible parser generator. Online; Acessado 18 de março de 2021.
- [Est] W. Estes. Flex: Fast lexical analyzer generator. Online; Acessado 21 de fevereiro de 2021.

A Gramática

$$\begin{aligned}
\langle \text{program} \rangle &::= \langle \text{declaration_list} \rangle \\
\langle \text{declaration_list} \rangle &::= \langle \text{declaration} \rangle \langle \text{declaration_list} \rangle \mid \langle \text{declaration} \rangle \\
\langle \text{declaration} \rangle &::= \langle \text{function_declaration} \rangle \mid \langle \text{var_declaration} \rangle \\
\langle \text{var_declaration} \rangle &::= \langle \text{type} \rangle \langle \text{id} \rangle \text{' ; ' } \\
\langle \text{function_declaration} \rangle &::= \langle \text{type} \rangle \langle \text{id} \rangle \text{' (' } \langle \text{params_list} \rangle \text{') ' } \langle \text{brackets_stmt} \rangle \\
&\quad \mid \langle \text{type} \rangle \langle \text{id} \rangle \text{' (' ' ' } \langle \text{brackets_stmt} \rangle \\
\langle \text{params_list} \rangle &::= \langle \text{type} \rangle \langle \text{id} \rangle \text{' , ' } \langle \text{param_list} \rangle \mid \langle \text{type} \rangle \langle \text{id} \rangle \\
\langle \text{stmts} \rangle &::= \langle \text{stmt} \rangle \langle \text{stmts} \rangle \mid \langle \text{stmt} \rangle \\
\langle \text{stmt} \rangle &::= \langle \text{for_stmt} \rangle \mid \langle \text{if_else_stmt} \rangle \mid \langle \text{return_stmt} \rangle \mid \langle \text{io_stmt} \rangle \\
&\quad \mid \langle \text{brackets_stmt} \rangle \\
&\quad \mid \langle \text{exp_stmt} \rangle \\
&\quad \mid \langle \text{set_stmt} \rangle \\
&\quad \mid \langle \text{var_declaration} \rangle \\
&\quad \mid \langle \text{assignment} \rangle \text{' ; ' } \\
\langle \text{assignment} \rangle &::= \langle \text{id} \rangle \text{' = ' } \langle \text{exp} \rangle \\
\langle \text{brackets_stmt} \rangle &::= \text{' { ' } \langle \text{stmts} \rangle \text{' } ' } \\
\langle \text{io_stmt} \rangle &::= \text{read ' (' } \langle \text{id} \rangle \text{') ' ' ; ' } \\
&\quad \mid \text{write ' (' } \langle \text{string} \rangle \mid \langle \text{exp} \rangle \text{') ' ' ; ' } \\
&\quad \mid \text{writeln ' (' } \langle \text{string} \rangle \mid \langle \text{exp} \rangle \text{') ' ' ; ' } \\
\langle \text{for_stmt} \rangle &::= \text{for ' (' } \langle \text{assignment} \rangle \text{' ; ' } \langle \text{exp} \rangle \text{' ; ' } \langle \text{assignment} \rangle \text{') ' } \langle \text{stmt} \rangle \\
\langle \text{if_else_stmt} \rangle &::= \text{if ' (' } \langle \text{exp} \rangle \text{') ' } \langle \text{stmt} \rangle \\
&\quad \mid \text{if ' (' } \langle \text{exp} \rangle \text{') ' } \langle \text{brackets_stmt} \rangle \text{ else } \langle \text{stmt} \rangle \\
\langle \text{return_stmt} \rangle &::= \text{return ' ; ' } \mid \text{return } \langle \text{exp} \rangle \text{' ; ' } \\
\langle \text{set_stmt} \rangle &::= \text{forall ' (' } \langle \text{id} \rangle \text{ in } \langle \text{set_exp} \rangle \text{') ' } \langle \text{stmt} \rangle \\
&\quad \mid \text{is_set ' (' } \langle \text{id} \rangle \mid \langle \text{set_exp} \rangle \text{') ' } \\
\langle \text{exp_stmt} \rangle &::= \langle \text{exp} \rangle \text{' ; ' } \mid \text{' ; ' } \\
\langle \text{exp} \rangle &::= \langle \text{or_exp} \rangle \mid \langle \text{set_exp} \rangle \\
\langle \text{set_exp} \rangle &::= \text{add ' (' } \langle \text{set_in_exp} \rangle \text{') ' } \\
&\quad \mid \text{remove ' (' } \langle \text{set_in_exp} \rangle \text{') ' } \\
&\quad \mid \text{exists ' (' } \langle \text{set_in_exp} \rangle \text{') ' } \\
\langle \text{set_in_exp} \rangle &::= \langle \text{or_exp} \rangle \text{ in } \langle \text{set_exp} \rangle \\
\langle \text{or_exp} \rangle &::= \langle \text{or_exp} \rangle \text{' || ' } \langle \text{and_exp} \rangle \mid \langle \text{and_exp} \rangle
\end{aligned}$$

$$\begin{aligned}
\langle and_exp \rangle &::= \langle and_exp \rangle \text{'\&\&'} \langle relational_exp \rangle \mid \langle relational_exp \rangle \\
\langle relational_exp \rangle &::= \langle relational_exp \rangle \langle relational_op \rangle \langle sum_exp \rangle \mid \langle sum_exp \rangle \\
\langle relational_op \rangle &::= \text{'<'} \mid \text{'>'} \mid \text{'>='} \mid \text{'<='} \mid \text{'=='} \mid \text{'!='} \\
\langle sum_exp \rangle &::= \langle sum_exp \rangle \text{'+'} \langle mul_exp \rangle \\
&\mid \langle sum_exp \rangle \text{'-'} \langle mul_exp \rangle \\
&\mid \langle mul_exp \rangle \\
\langle mul_exp \rangle &::= \langle mul_exp \rangle \text{'*'} \langle primal_exp \rangle \\
&\mid \langle mul_exp \rangle \text{'/'} \langle primal_exp \rangle \\
&\mid \langle unary_exp \rangle \\
\langle unary_exp \rangle &::= \langle primal_exp \rangle \mid \text{'!'} \langle primal_exp \rangle \mid \text{'-'} \langle primal_exp \rangle \\
&\mid \langle id \rangle \text{'(' arg_list ')'} \\
&\mid \langle id \rangle \text{'(' ' ')'} \\
\langle primal_exp \rangle &::= \langle id \rangle \mid \langle const \rangle \mid \text{'('} \langle exp \rangle \text{')'} \\
\langle arg_list \rangle &::= \langle exp \rangle, \langle arg_list \rangle \mid \langle exp \rangle \\
\langle type \rangle &::= \langle basic_type \rangle \mid \langle elem_type \rangle \mid \langle set_type \rangle \\
\langle const \rangle &::= \langle int_const \rangle \mid \langle float_const \rangle \mid \langle empty_const \rangle \\
\langle int_const \rangle &::= \langle digit \rangle + \\
\langle float_const \rangle &::= \langle digit \rangle + \text{'.'} \langle digit \rangle + \\
\langle empty_const \rangle &::= \text{'EMPTY'} \\
\langle elem_type \rangle &::= \text{elem} \\
\langle set_type \rangle &::= \text{set} \\
\langle basic_type \rangle &::= \text{int} \mid \text{float} \\
\langle string \rangle &::= \text{'.'*} \mid \text{'.'*} \\
\langle id \rangle &::= [\text{a-zA-Z_}][\text{a-zA-Z_}]^* \\
\langle digit \rangle &::= [0-9]
\end{aligned}$$