



**UNIVERSIDADE ESTADUAL PAULISTA
“JÚLIO DE MESQUITA FILHO”**

RELATÓRIO ANALISADOR SINTÁTICO

DISCIPLINA: COMPILADORES

**CURSO DE BACHARELADO EM CIÊNCIA DA
COMPUTAÇÃO**

**DARLAN MURILO NAKAMURA DE ARÁUJO – RA:
151251207**

**RAFAEL BEZERRA DE MENEZES RODRIGUES – RA:
151251551**

PRESIDENTE PRUDENTE – SP

OUTUBRO DE 2018

Sumário

1 - Introdução.....	3
2 - Problemas Encontrados	3
3 – BNF Especificada.....	4
4 – Casos de Teste	4
5 – Tratamento de Erros	6

1 - Introdução

Para o desenvolvimento do Analisador Sintático e Léxico, foi utilizado as seguintes bibliotecas:

- jFlex versão 1.6.1 (disponível em <http://jflex.de/download.html>)

- jCup versão 11 (java-cup-11b) (disponível em <http://www2.cs.tum.edu/projects/cup/>)

O CUP funciona como um LALR Parser – Look-Ahead LR Parser, do qual realiza o *parse* (separa e analisa) o texto de acordo com as regras de produção especificadas por uma gramática formal. Foi necessário um estudo profundo da ferramenta, para entender como ela funciona e como utilizá-la, conhecendo suas vantagens e desvantagens.

Durante o desenvolvimento, percebeu-se que a gramática utilizada no CUP é diferente das linguagens LL aprendida na disciplina, sendo necessário o estudo sobre as gramáticas SLR. Portanto, foi necessário refatorar a BNF da LALG – Linguagem Pascal Simplificado para uma gramática SLR, com derivação mais à direita.

2 - Problemas Encontrados

Percebemos que alguns trechos entre o programa correto e a própria BNF da LALG não estavam corretos, de acordo com o padrão utilizado nas linguagens. Realizamos algumas correções na linguagem especificada pela BNF da LALR (disponível em <http://gege.fct.unesp.br/docentes/dmec/olivete/compiladores/arquivos/LALG.pdf>), como por exemplo:

No documento, está especificado que seção de parâmetros formais é descrita como:

<seção de parâmetros formais> ::= [**var**] <lista de identificadores> : <identificador>

Porém, no exemplo do programa correto está:

```
...  
procedure proc(var a1 : int);  
...
```

Como sabemos, int não é um identificador, mas um tipo, podendo ser “int” ou “boolean”.

Realizamos algumas correções também no “programa correto1.txt”, pois algumas linhas estavam em desacordo com a BNF apresentada no documento, como por exemplo:

A declaração “*if*” definida no documento, diz que obrigatoriamente, a declaração “*if* expressão” deve possuir a palavra reservada “*then*” em seguida.

<comando condicional 1> ::= if <expressão> then <comando> [else <comando>]

Trecho extraído da BNF LALG

E as declarações do comando condicional *if* encontrado no “programa correto1.txt” não possuíam o *then*:

```
...
write(b);
    if (d)
    begin
        a:=20;
        ...
```

Trecho de código do programa correto.txt

3 – BNF Especificada

As principais alterações foram:

- 1- Padronizar o ponto e vírgula (“;”) em todo final de comando, independente se não é o último do bloco.
- 2- Nenhum comando “*end*” terá ponto e vírgula (“;”).
- 3- Todo bloco deve iniciar com “*begin*” e finalizar com “*end*”.

O bloco interior ao comando “*if*”, não é necessário possuir as palavras reservadas “*begin*” e “*end*” desde que seja apenas uma única expressão. Do contrário, é necessário. O mesmo serve para o “*else*”.

A BNF do projeto pode ser encontrada através do arquivo “Parser.cup”, dentro da pasta “src\AnalizadorLexicoCalculadora\ui”.

4 – Casos de Teste

Na pasta raiz do projeto, há uma pasta chamada “programas corretos”, na qual possui diversos programas com uma sintaxe da qual a linguagem aceita. Após refatorar o “programa correto1.txt” para a linguagem aceita (do qual pode ser acessado pelo arquivo “programa correto inteiro.rad”), obtemos:

```

program correto;

int a, b, c;

boolean d, e, f;

procedure proc(var a1 : int);

int a, b, c;

boolean d, e, f;

begin
    a:=1;
    if (a<1) then
        a:=12;
    end

begin
    a:=2;
    b:=10;
    c:=11;
    a:=b+c;
    d:=true;
    e:=false;
    f:=true;
    read(a);

    if (d) then
        begin
            a:=20;
            b:=10*c;
            c:=a / b;
        end

    while (a>1) do
        begin
            if (b>10) then
                begin
                    b:=2;
                    a:=a-1;
                end
            end
        end
    end.

```

5 – Tratamento de Erros

A recuperação de erros é realizada através do método do pânico.

Através desse método, é necessário informar para o compilador, pontos de sincronização.

No projeto, foram adicionados pontos de sincronização específicos, baseados nos erros mais comuns dos usuários.

5.1 – Comando de Atribuição

```
...  
c:=11;  
a:=  
d:=true;  
e:=false;  
f:=true;  
read(a);
```

Na atribuição da variável “a” há um erro. Portanto, foi adicionado um ponto de sincronização na atribuição, e o analisador sintático informa que há um erro na atribuição e realiza se recupera a partir do ponto de sincronização.

5.2 – Atribuição de Variável

```
program correto;  
int a, b  
boolean d, e, f;
```

Neste caso, o usuário colocou vírgula “,” ao invés do ponto e vírgula “;”.

O analisador sintático será capaz de se recuperar deste tipo de erro que ocorrerá durante a atribuição de variável.