SCC0605 Teoria da Computação e Compiladores

SAMIRA CRISTINA VASCONCELOS MENDES
MATHEUS ARATAQUE UEMA
NELSON CALSOLARI NETO

Desenvolvimento de um analisador sintático para linguagem P-- e sua integração com o analisador léxico anteriormente desenvolvido

SÃO CARLOS
2020

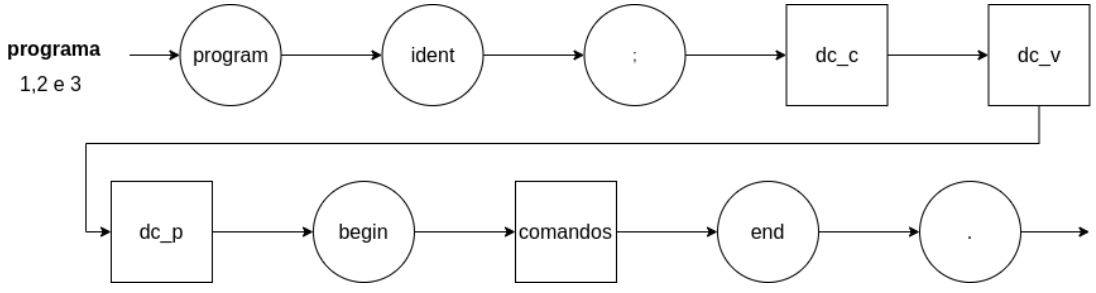
1. ALTERAÇÕES NO ANALISADOR LÉXICO

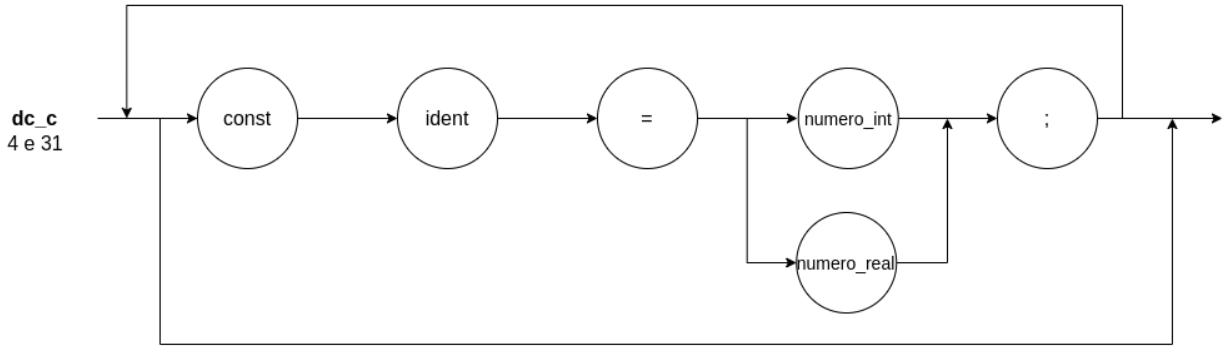
Para que o analisador léxico funcione corretamente em conjunto com o analisador sintático, foram realizadas algumas mudanças e correções de erros identificados ao longo da implementação. Configuramos o analisador léxico para que a chamada da sua função realize a leitura de apenas uma cadeia por vez do arquivo de entrada, retornando o token correspondente para função que o chamou.

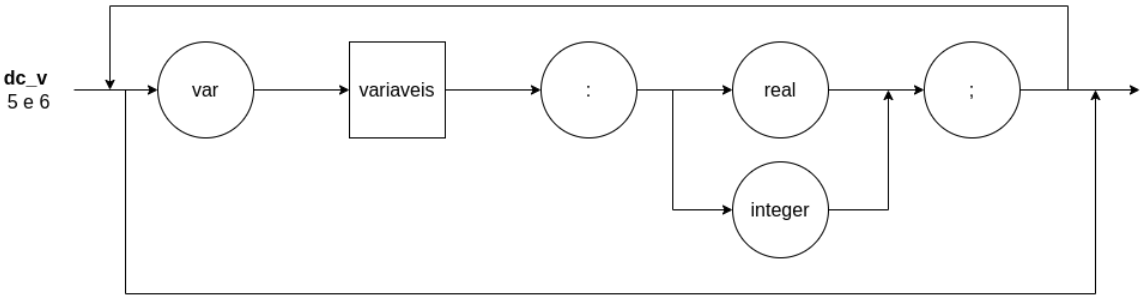
Além disso, foi necessário implementar o reconhecimento para os comandos reservados ‘then’ e ‘for’ que não se encontravam na função do analisador léxico anteriormente, possibilitando assim a análise sintática completa dos comandos ‘if’ e ‘else’. Por fim, foi adicionada uma nova regra no reconhecimento de cadeias de símbolos reservados, visto que anteriormente não havia a separação na leitura de uma cadeia, por exemplo, ‘);’ retornava o token ‘simbolo_nao_reconhecido’, quando deveria retornar, primeiramente o token ‘comando_reservado_fecha_parenteses’ e, na chamada seguinte da função, o token ‘comando_reservado_ponto_virgula’.

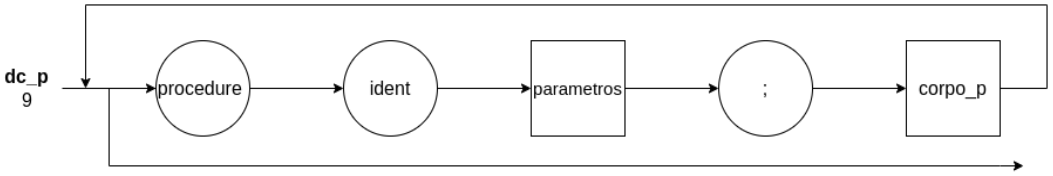
2. IMPLEMENTAÇÃO DO ANALISADOR SINTÁTICO

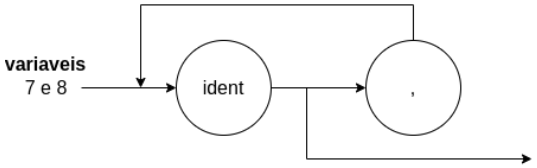
Para implementação do analisador léxico descendente preditivo recursivo, adotamos a abordagem recomendada em aula de desenvolver primordialmente os grafos sintáticos a partir das regras gramaticais. Os grafos resultantes desse processo e as respectivas regras gramaticais usadas para sua construção se encontram abaixo:

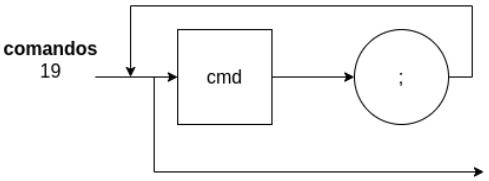


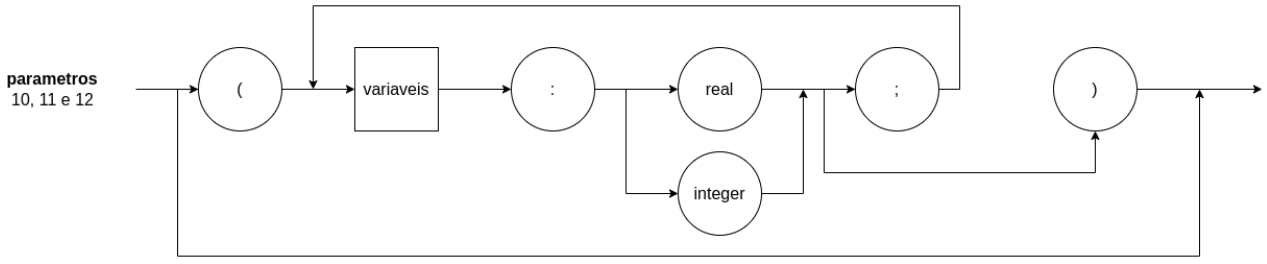

dc_c
4 e 31
const
ident
=
numero_int
;
numero_real

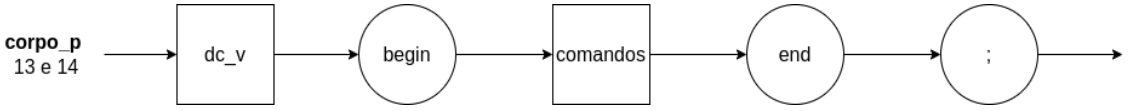

dc_v
5 e 6
var
variaveis
:
real
;
integer

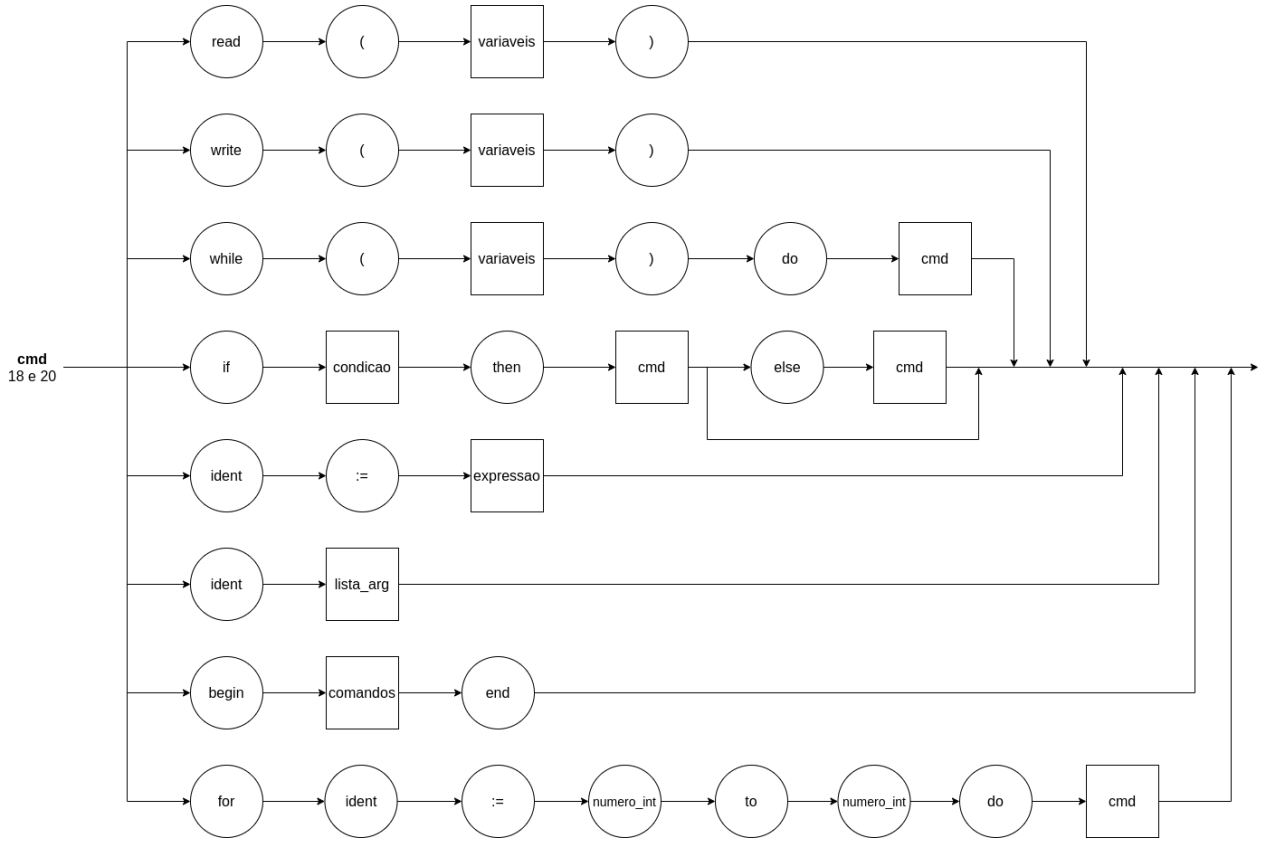

dc_p
9
procedure
ident
parametros
;
corpo_p

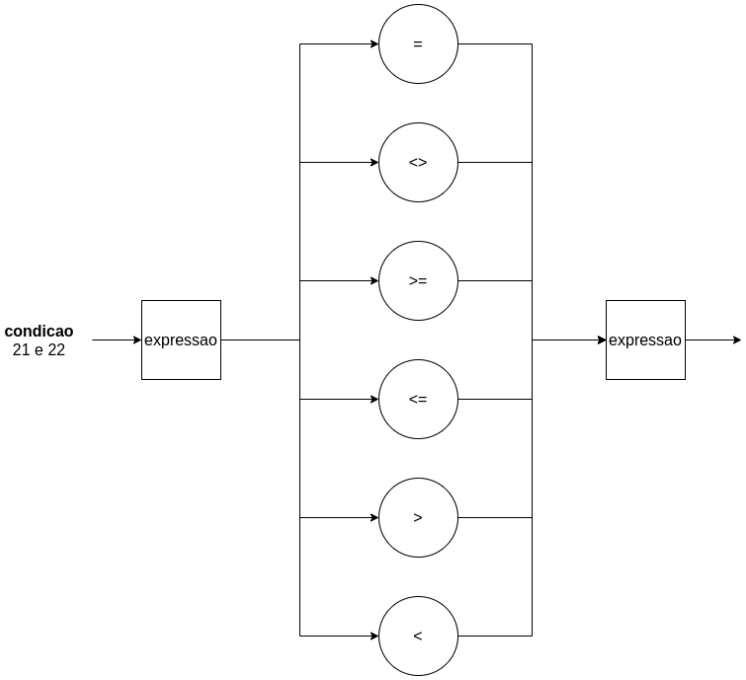

variaveis
7 e 8
ident
,

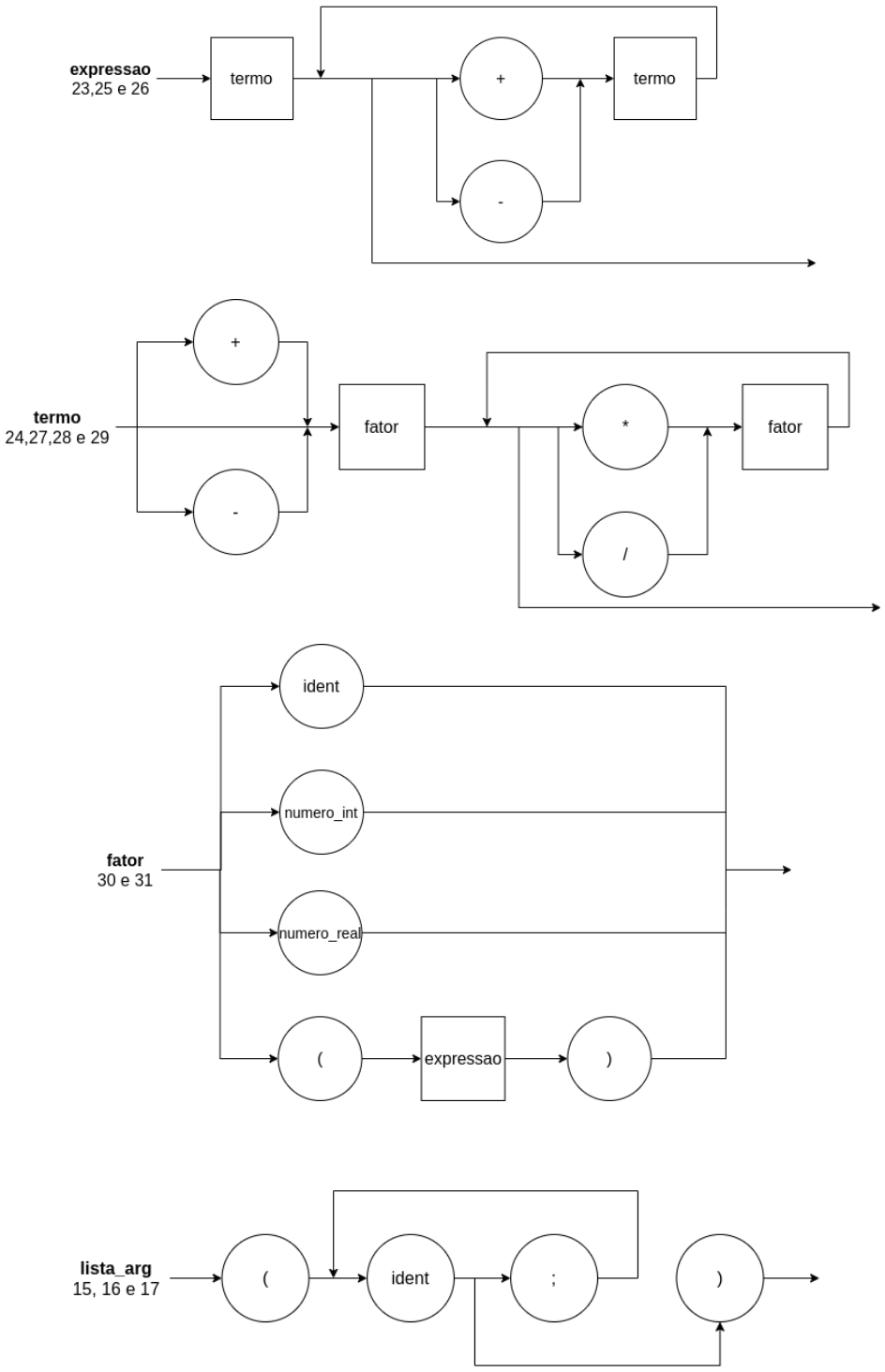

comandos
19
cmd
;


parametros
10, 11 e 12
(
variaveis
:
real
;
)
integer


corpo_p
13 e 14
dc_v
begin
comandos
end
;


cmd
18 e 20
read
(
variaveis
)
write
(
variaveis
)
while
(
variaveis
)
do
cmd
if
condicao
then
cmd
else
cmd
ident
:=
expressao
ident
lista_arg
begin
comandos
end
for
ident
:=
numero_int
to
numero_int
do
cmd


condicao
21 e 22
expressao
=
<>
>=
<=
>
<
expressao



Antes mesmo de implementar o código final, optamos por desenvolver o pseudocódigo a partir dos grafos a fim de facilitar a implementação.

3. TRATAMENTO DE ERROS

O tratamento de erros do analisador sintático foi realizado através da implementação do modo pânico, como sugerido em aula. Nessa implementação, caso seja encontrado um erro sintático, o código retorna o tipo do erro e a linha em que ocorreu, em seguida o modo pânico é ativado e a função ‘erro’ é chamada.

O modo pânico foi implementado de uma maneira em que, se o token esperado é reconhecido, o modo pânico é desativado, se o token esperado não é reconhecido e o modo pânico está desativado, o modo pânico é ativado e o tratamento de erros é realizado.

A função de tratamento de erros funciona da seguinte maneira: ela recebe um conjunto dos possíveis tokens, que são os tokens correspondentes aos seguidores imediatos do estado em que ocorreu o erro e aos seguidores do pai, e então realiza a leitura do arquivo de entrada até encontrar um desses possíveis tokens. Caso não encontre, a função retorna o token ‘erro’, que significa que a leitura foi realizada até o final do arquivo e não foi possível encontrar um dos tokens esperados, caso o tratamento de erros o encontre, o token é retornado.

Com relação a passagem de argumentos do seguidor pai, houve dois casos ao longo da implementação: as funções em que o seguidor do pai é sempre o mesmo, como no caso de ‘cmd’ que será sempre seguido por um ‘end’, e as funções em que é necessário, na chamada da função correspondente a este diagrama pelo diagrama pai, passar os seguidores do pai também, como no caso de ‘variaveis’ que, se chamada por ‘dc_v’, teria como seguidor pai o token ‘:’, mas se chamada pelo comando ‘write’, teria como seguidor pai o token ‘)’.

Além disso, foi adicionada uma verificação antes de determinados diagramas sintáticos a fim de identificar se o token lido é o token esperado. Por exemplo, na função ‘programa’, logo após o ‘;’ espera-se que o próximo token seja correspondente ao primeiro de ‘dc_c’, ‘dc_v’, ‘dc_p’ ou ‘begin’ e, caso não seja nenhum desses, um erro deve ser reportado e a leitura do arquivo até que se encontre esses tokens deve ser feita. Essa verificação foi feita nos casos em que uma função realiza a chamada de outra função, de forma que é possível saber qual o token esperado, como nas chamadas da função ‘comando’, por exemplo.

Uma falha identificada ao longo da implementação do tratamento de erros é a linha reportada no erro ocorrido. Em casos em que o erro ocorre no final da linha, a linha reportada no erro é a linha posterior - caso o erro tenha ocorrido na 9, reporta-se a linha 10 - , o que é decorrente do fato de que se reporta a linha em que o ponteiro de leitura do analisador léxico se encontra e não a linha do token retornado.

4. INSTRUÇÕES PARA COMPILAR O CÓDIGO-FONTE

Para compilar o código fonte:

```
gcc main.c −o main
```

Já para execução do código:

```
./main [arquivo de entrada]
```

O arquivo de entrada é o arquivo correspondente ao arquivo que se deseja ler. Foram deixados alguns exemplos de arquivo no trabalho. A implementação e os testes realizados no projeto foram feitos através do Visual Studio Code.