Trabalho Final INF1022 2024.1

Profs. Vitor e Hermann

2 de junho de 2024

1 Enunciado

O trabalho pode ser feito em dupla ou de forma individual. Neste trabalho deve ser desenvolvido um analisador sintático para a linguagem Provol-One. O analisador sintático deve ser capaz de compilar programas escritos utilizando a linguagem Provol-One para uma outra linguagem a sua escolha (como ilustrado na Figura 1). Ou seja, o analisador sintático recebe como entrada um programa na linguagem Provol-One e produz como saída um outro programa escrito em uma outra linguagem. A linguagem do programa de saída pode ser escolhida por você, ela pode ser qualquer linguagem, por exemplo, C, C++, Java, Lua, Pyhton ou Assembly.

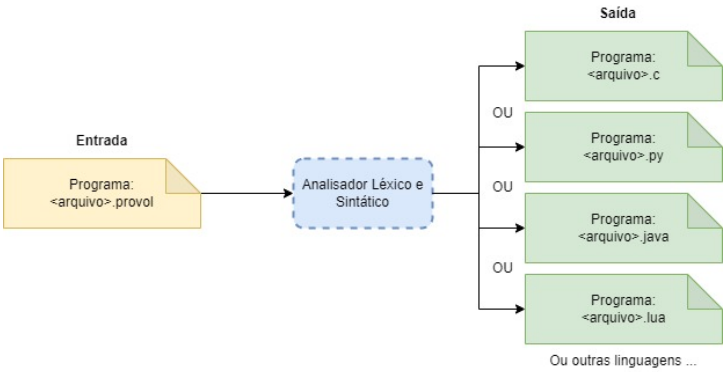

Figura 1: Analisador sintático para Provol-One.

Para a implementação do analisador sintático, deve-se usar um gerador de analisador sintático que implemente o método LaLR(1) ou outro ascendente, por exemplo, Yacc/Lex,Bison/Flex (usa LaLR(1)), JavaCC (que usa o descendente LL(1)), etc.

1.1 Desenvolvimento

A sintaxe da linguagem *Provol-One* é dada pela gramática abaixo:

$$
\begin{array}{lcl}
programa & \longrightarrow & INICIO\ varlist\ MONITOR\ varlist\ EXECUTE\ cmds\ TERMINO \\
varlist & \longrightarrow & id\ varlist\ \mid\ id \\
cmds & \longrightarrow & cmd\ cmds\ \mid\ cmd \\
cmd & \longrightarrow & ENQUANTO\ id\ FACA\ cmds\ FIM \\
cmd & \longrightarrow & id\ =\ id
\end{array}
$$

- Todas as variáveis são do tipo inteiro e não negativo.
- O não terminal *varlist* descreve uma lista de variáveis separados por virgúla.
- O não terminal $INICIO$ sinaliza o inicio de todo programa escrito em Provol-One e deve ser sucedido por uma lista completa das variáveis (*varlist*) que serão utilizadas no programa.
- O não terminal $MONITOR$ sinaliza que logo após ele será descrito uma lista de variávies que serão monitoradas durante a execução do programa. As variáveis que estão descritar em $MONITOR$ devem ser impressas no console sempre que forem inicializadas ou alteradas durante a execução do programa.
- O não terminal $EXECUTE$ sinaliza que após dele virá uma sequencia de comandos (*cmds*) que devem ser executados.
- Por fim, o não terminal $TERMINO$ sinaliza o fim do programa.

Além disso, a linguagem Provol-One será capaz de executar um loop que é definido pela regra:

$$cmd \longrightarrow ENQUANTO\ id\ FACA\ cmds\ FIM$$

A regra acima, que define o loop, diz que enquanto o valor que estiver em *id* for diferente de zero, execute os comandos que estão descritos em *cmds*. O *id* é um número inteiro que quando seu valor é zero siginifica (falso) e quando é qualquer coisa diferente de zero signifca (verdadeiro).

Além disso, a gramática descrita acima deve ser complementada para que a linguagem Provol-One seja capaz de executar comandos do tipo:

- IF-THEN e IF-THEN-ELSE: Comandos de seleção do tipo SE-ENTÃO. Por exemplo: IF condicao THEN comando. Ou IF condicao THEN comando ELSE comando.
- EVAL(*num*1,*num*2,*cmds*): Recebe uma lista de comandos (*cmds*), um número (*num*1) e um número (*num*2). O EVAL executa a lista de comandos *cmds* exatamente até *num*1 ser igual ao *num*2. A cada execução do EVAL *num*1 é subtraido em 1. Ou seja, os comandos contigos em *cmds*

vão ser executados no total *num*1 - *num*2 vezes. Caso *num*1 seja igual a *num*2, ele não ira executar os comandos contidos em *cmds* nenhuma vez.

O formato alternativo para este comando é: EVAL *cmds* VEZES *num*3 FIM. Onde *num*3 é igual a *num*1-*num*2. Significa executar os comandos cmds exatamente *num*1-*num*2 vezes. Com *num*1,*num*2 e *num*3 sendo números inteiros positivos. **Só é obrigatório a implementação de um dos formatos.**

- Soma A com B: Funcao para somar A com B. Pode ser implementada em dois formatos: (1) SOMA(A,B) ou (2) A + B. **Só é obrigatório a implementação de um dos formatos.**
- Multiplica A por B: Pode ser implementada em dois formatos: (1) MULT(A,B) ou (2) A * B. **Só é obrigatório a implementação de um dos formatos.**
- ZERO(A): Função que zera a variável A.

Alguns exemplos[1] de programas em Provol-One são:

```
INICIO X, Y
MONITOR  Z
EXECUTE
Y = 2
X = 5
Z = Y
ENQUANTO X FACA
   Z = Z + 1
FIM
TERMINO
```

Exemplo 1

```
INICIO X, Y, B
MONITOR  Z
EXECUTE
X = 5
Y = 2
B = 0
EVAL(X, Y,
IF B THEN Z = Z + 2
ELSE Z = Z + 1)
TERMINO
```

Exemplo 2

```
INICIO X, Y, L
MONITOR  Z
EXECUTE
ZERO(Z)
X = 5
Y = 0
L = 1
IF Y THEN
Z = EVAL(X, L, Z=Z*L)
ELSE
Z = EVAL(X, L, Z=Z*X)
FIM
TERMINO
```

Exemplo 3

1.2 Observação

Você também tem a permissão de expandir a gramática para permitir comandos adicionais ou demais operações que julgar interessantes e/ou necessárias.
Deixe explícitas as alterações realizadas na gramática descrita neste documento (documente no relatório do trabalho). Tanto as alterações obrigatórias (solicitadas neste trabalho) quanto alterações que você julgar interessantes.

[1] Usando o EVAL na forma EVAL(A,B,CMDS).

2 Entrega

Você deve entregar um arquivo contendo seu relatório, no qual estarão descritos seu trabalho - o que foi implementado, como foi implementado, o que funciona, o que não funciona, quais os testes utilizados etc. - e quaisquer outras informações que você considere relevantes, como a maneira de executá-lo. Caso utilize alguma fonte na sua pesquisa do trabalho, indique também.
No seu relatório final deve estar contido a gramática final utilizada no trabalho. A gramática deve estar descrita de maneira similar ao que a gramatica inicial foi descrita neste documento na parte 1.1. Como uma gramática LLC. Deixe claro quais regras foram adicionadas na gramática descrita neste relatório para que seja possível permitir todas as funcionalidades solicitadas. Caso tenha implementado alguma funcionalidade adicional, ou seja, uma que não foi solicitada, por favor deixe explícito.
Além disso, entregue um .zip com os casos de teste utilizados.
Por fim, entregue seu analisadorléxico e sintático bem como quaisquer outros arquivos/módulos auxiliares que tenha criado para a execução do trabalho.
Indique nome e matrícula de **ambos os componentes da dupla**, se for o caso.
Caso tenha arguições, a ordem das arguições será definida com base na ordem das entregas no EAD.

Dica

- Comece pequeno: garanta que seu analisador funcione para um fragmento da linguagem, depois vá incrementando.