Trabalho Final INF1022 2025.2

Prof. Vitor Pinheiro

14 de novembro de 2025

1 Enunciado

O trabalho pode ser feito em dupla ou de forma individual. Neste trabalho deve ser desenvolvido um analisador sintático para a linguagem ObsAct. O analisador sintático deve ser capaz de compilar programas escritos utilizando a linguagem ObsAct para uma outra linguagem a sua escolha (como ilustrado na Figura 1). Ou seja, o analisador sintático recebe como entrada um programa na linguagem ObsAct 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 a sua escolha. Por exemplo, mas não restrita a essas: C, C++, Java, Lua, Pyhton ou Assembly.

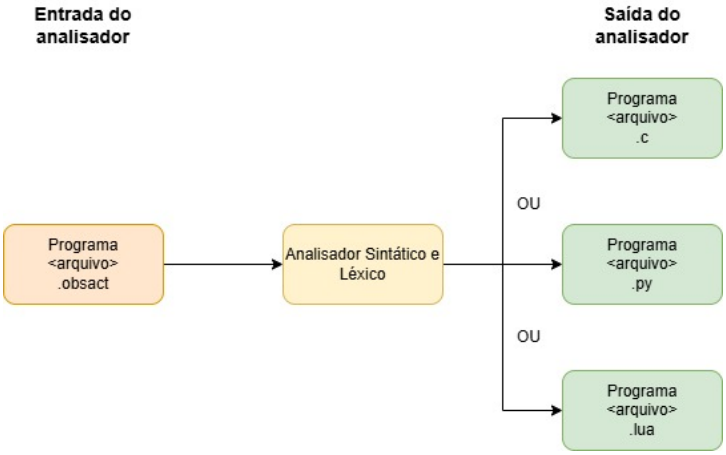

Figura 1: Analisador sintático para a linguagem ObsAct.

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 *ObsAct* é dada pela gramática abaixo:

$$
\begin{array}{lcl}
PROGRAM & \longrightarrow & DEV\_SEC\ \ CMD\_SEC \\
DEV\_SEC & \longrightarrow & dispositivos:\ \ DEV\_LIST\ \ fimdispositivos \\
DEV\_LIST & \longrightarrow & DEVICE\ \ DEV\_LIST \mid DEVICE \\
DEVICE & \longrightarrow & ID\_DEVICE \\
DEVICE & \longrightarrow & ID\_DEVICE\ \ [ID\_OBS] \\
CMD\_SEC & \longrightarrow & CMD\_LIST \\
CMD\_LIST & \longrightarrow & CMD;\ \ CMD\_LIST \mid CMD; \\
CMD & \longrightarrow & ATTRIB \mid OBSACT \mid ACT \\
ATTRIB & \longrightarrow & def\ \ ID\_OBS\ \ =\ \ VAL \\
OBSACT & \longrightarrow & quando\ \ OBS\ \ :\ \ ACT \\
OBSACT & \longrightarrow & quando\ \ OBS\ \ :\ \ ACT\ \ senao\ \ ACT \\
OBS & \longrightarrow & ID\_OBS\ \ OPLOGIC\ \ VAL \\
OBS & \longrightarrow & ID\_OBS\ \ OPLOGIC\ \ VAL\ \ AND\ \ OBS \\
VAL & \longrightarrow & NUM \mid BOOL \\
ACT & \longrightarrow & execute\ \ ACTION\ \ em\ \ ID\_DEVICE \\
ACT & \longrightarrow & alerta\ \ para\ \ ID\_DEVICE:\ \ MSG \\
ACT & \longrightarrow & alerta\ \ para\ \ ID\_DEVICE:\ \ MSG\ \ ,\ \ ID\_OBS \\
ACT & \longrightarrow & difundir:\ \ MSG\ \ ->\ \ [DEV\_LIST\_N] \\
ACT & \longrightarrow & difundir:\ \ MSG\ \ \ ID\_OBS\ \ ->\ \ [DEV\_LIST\_N] \\
ACTION & \longrightarrow & ligar \mid desligar \\
DEV\_LIST\_N & \longrightarrow & ID\_DEVICE \mid ID\_DEVICE\ \ ,\ \ DEV\_LIST\_N
\end{array}
$$

- Todas as variáveis númericas são do tipo inteiro e não negativo.
- Todas os terminais (NUM, ID_DEVICE, ID_OBS, VAL, $OPLOGIC$, $BOOL$, AND e MSG) não podem ser vazios.
- NUM representa um número inteiro não negativo.
- $BOOL$ representa TRUE ou FALSE.
- ID_OBS é uma string que representa o nome do sensor a ser observado. Pode conter letras e números, porém só pode começar com letra.
- O terminal ID_DEVICE só pode conter letras.
- O terminal AND força que as duas sentenças precisam ser verdadeiras para que a condição logica seja verdadeira. Por exemplo: se movimento == True AND luz < 100 entao ligar lampada.
- Perceba que o terminal ";" sinaliza o fim dos comandos $ATTRIB$ e $OBSACT$.
- $OPLOGIC$: é um terminal que representa um operador lógico. Eles podem ser: >, <, >=, <=, == ou ! =

- As variaveis de *msg* e *id_device* podem conter no máximo 100 caracteres.
- A regra:

$$ACT \longrightarrow alerta\ para\ ID\_DEVICE:\ MSG\ ,\ ID\_OBS$$

diz para enviar um alerta para o device com nome ID_DEVICE. A mensagem de alerta contém duas strings que devem ser concatenadas: MSG e ID_OBS. A concatenação resultante deve ser MSG + (espaço em branco) + ID_OBS. Por exemplo, a linha de código:

```
alerta para Termometro: "Temperatura medida esta em",
    temperatura ;
```

vai concatenar a string "Temperatura medida está em"com a variável *temperatura* e vai resultar na string "Temperatura medida está em 37", sendo que 37 seria o valor da variável *temperatura*.

- A regra:

$$ACT \longrightarrow difundir:\ MSG\ ->\ [DEV\_LIST\_N]$$
$$ACT \longrightarrow difundir:\ MSG\ \ ID\_OBS\ ->\ [DEV\_LIST\_N]$$

diz para enviar um alerta para todos os devices que estão na DEV_LIST_N. A primeira regra do difundir só envia o conteúdo de MSG. A segunda regra do difunfir envia o conteúdo de MSG concatenado com o condeúdo de ID_OBS.

A gramática acima não esta completa, neste trabalho você vai precisar definir algumas regras a mais para incluir algumas funcionalidades. **Mais importante que isso, fique a vontade para alterar a gramática caso ache necessário para poder gerar a linguagem que esta sendo pedida.** Por exemplo, você pode retirar recursão a esquerda, fatorar ou até alterar as regras acima. Desde que a linguagem final gerada seja a mesma.

1.2 Exemplos de código na linguagem Obs

Alguns exemplos de programas em ObsAct são:

```
dispositivos:
  Termometro[temperatura]
  Ventilador[potencia]
fimdispositivos
def temperatura = 40;
def potencia = 90;
quando temperatura > 30 : execute ligar em Ventilador;
```

```
dispositivos:
  Monitor
  Celular
  Termometro[temperatura]
fimdispositivos

quando temperatura > 30 :
difundir: "Temperatura em " temperatura -> [Monitor, Celular
    ];
```

```
dispositivos:
  Monitor
  Celular[movimento]
  Higrometro[umidade]
  Lampada[potencia]
fimdispositivos

def potencia = 100 ;
quando umidade < 40 : alerta para Monitor: "Ar seco
    detectado" ;
quando movimento == True : execute ligar em Lampada senao
    execute desligar em Lampada ;
```

```
dispositivos:
  Monitor
  Higrometro[umidade]
fimdispositivos

quando umidade < 40 : alerta para Monitor: "Ar seco
    detectado" ;
```

```
dispositivos:
  Lampada[potencia]
fimdispositivos
def potencia = 100;
execute ligar em Lampada;
```

```
dispositivos:
  Ventilador
fimdispositivos
execute desligar Ventilador ;
```

```
dispositivos:
  Celular
fimdispositivos
alerta para Celular: "Hora de acordar!" ;
```

```
dispositivos:
  Termometro[temperatura]
fimdispositivos
alerta para Termometro: "Temperatura esta em", temperatura ;
```

1.3 Sobre os dispositivos

Veja que no inicio de todo programa ObsAct o programador precisa primeiro declarar todos os dispositivos que serão utilizados ao longo do programa. Como por exemplo:

```
dispositivos:
  Monitor
  Celular[movimento]
  Higrometro[umidade]
  Lampada[potencia]
fimdispositivos
```

Todo programa gerado a partir de um programa escrito em ObsAct vai conter 4 funções: uma para ligar o dispositivo, outra para desligar o dispositivo, outras duas para enviar um alerta para o dispositivo. Estas funções devem ser implementadas como a seguir:

```
function ligar(id_device)
{
    print(id_device+" ligado!")
}

function desligar(id_device)
{
    print(id_device+" desligado!")
}

function alerta(id_device, msg)
{
    print(id_device+" recebeu o alerta:\n")
    print(msg)
}

function alerta(id_device, msg, var)
{
    print(id_device+" recebeu o alerta:\n")
    print(msg + " "+ var)
}
```

Veja que estas 4 funções definidas acima estão em um pseudocódigo. *id_device* é a variavel que identifica o dispositivo.

Elas devem ser implementadas na linguagem de programação escolhida pelo aluno. Por exemplo, caso o aluno queria traduzir a linguagem ObsAct para Python, as funções acima devem ser geradas em Python.

Desta maneira, toda vez que alguma ação de *ligar*, *desligar* ou *alerta* for executada para algum *id_device*, são estas funções que devem ser chamadas passando como parametro o *id_device*.

Para facilitar o trabalho, o aluno pode escolher também já deixar estas 4 funções implementadas no código. Por exemplo, se o aluno for gerar código em Python, ele já pode ter um arquivo .py que vai conter estas 4 funções (ligar, desligar e as duas de alerta).

1.4 Suposições

Todo valor de ID_OBS que não for definido pelo programador através da linha de código:

```
1 def temperatura = 40 .
```

(onde temperatura é o ID_OBS), deve ser definido para zero.

1.5 Observação

Você 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.

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.

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. 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 o código do seu analisador 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.