Trabalho Final Analisadores Lexicos e sintaticos

Livian Essvein 2211667
Luiza Régnier 2211931

Neste trabalho foi implementado um compilador para a linguagem ObsAct:

O compilador foi feito na linguagem Python usando o PLY (Python Lex-Yacc), que basicamente é uma versão do Lex/Yacc para Python. O objetivo era pegar um arquivo .obs, interpretar suas instruções e gerar um arquivo C equivalente (saida.c).

O código do programa compilador.py foi feito aos poucos: primeiro o léxico, depois o parser, depois a AST, e por último a geração de código em c.

como rodar: para executar o compilador.py, basta usar o Python passando o arquivo `.obs` de teste como argumento no próprio terminal:

python compilador.py exemplo.obs

- **Organização do compilador:**

1. Analisador léxico: descobre os tokens
2. Analisador sintático: valida a gramática
3. AST (árvore sintática): organiza o que foi lido
4. Análise semântica: verificação extra (por exemplo: usar só dispositivos válidos).
5. Geração de código C: transforma o programa ObsAct em um programa C.

- **Parte léxica:**

A parte léxica usa várias funções **t_*** do PLY. Elas são chamadas automaticamente pelo PLY.

- t_ID, t_NUM, t_BOOL, t_MSG, t_OP_LOGIC, t_ARROW
- palavras reservadas (dispositivos, def, alerta, etc.)

O PLY lê o arquivo .obs e vai chamando essas funções para transformar o texto em uma sequência de tokens. Também tem t_ignore para ignorar espaços e t_COMMENT para ignorar comentários.

- **Parte Sintática:**

Cada regra da gramática foi transformada em uma função p_alguma coisa seguindo o padrão do Yacc. Exemplos do código:

```
def p_programa(p):
    """programa : sec_dev sec_cmd"""
    p[0] = (p[1], p[2])

def p_sec_dev(p):
    """sec_dev : DISPOSITIVOS ':' lista_devices FIMDISPOSITIVOS"""
    p[0] = p[3]

def p_lista_devices_rec(p):
    """lista_devices : linha_device lista_devices"""
    p[0] = [p[1]] + p[2]
```

A partir daí, cada regra cria um pedaço da AST (uma tupla).

Por exemplo:

- Atribuição: ("atribuicao", var, valor)
- Ligar: ("ligar", dispositivo)
- If : ("if", condicao, acao)
- Difundir: ("diff1", msg, lista_ids)

O PLY chama automaticamente essas funções durante o parsing.
Se uma regra da gramática não casar, o compilador chama p_error.
Essa parte cobre tudo de gramática, derivações, não-terminais, terminais, regras de produção, etc.

- **Árvore Sintática:**

A AST foi construída usando tuplas e listas simples.

Exemplos de nós:

- Condição: ("cond", id, op, valor)
- And lógico: ("and", cond1, cond2)
- Alerta variável: ("alerta2", id, mensagem, var)

- **Geração do código em c:**

Foram criadas funções para cada tipo de nó da Arvore:

- ❖ gerar_acao_ligar
- ❖ gerar_acao_desligar
- ❖ gerar_alerta1
- ❖ gerar_alerta2
- ❖ gerar_if
- ❖ gerar_ifelse
- ❖ gerar_diff1
- ❖ gerar_diff2
- ❖ gerar_variavel

E por fim uma função principal gerar_codigo(lista_cmd) que chama cada uma dependendo do comando.

O arquivo final saida.c contém:

- Includes necessários para rodar o código em c (stdio.h, string.h)
- As funções (ligar, desligar, alerta, alerta_var)
- A função main()
- Todas as instruções traduzidas dentro da main usando as funções e variaveis traduzidas geradas.

- **Testes:**

Todos os arquivos com a linguagem ObsAct usados para testar o compilador foram tirados do próprio enunciado, por exemplo:

(exemplo 2 do enunciado)

```
dispositivos:
  Termometro[temperatura]
  Ventilador[potencia]
fimdispositivos

def temperatura = 40;
def potencia = 90;

quando temperatura > 30 : execute ligar em Ventilador;
```

- **O que foi feito a mais:**

Função que valida o dispositivo (validar_dispositivos(lista_dispositivos, lista_cmd)):

Se o dispositivo não foi iniciado depois de dispositivos:, a função emite uma mensagem de que o dispositivo não existe.

Inicio da função:

```
def validar_dispositivos(lista_dispositivos, lista_cmd):
    # cria um conjunto com nomes de dispositivos declarados
    nomes = {nome for (nome, attr) in lista_dispositivos}

    erros = []

    for cmd in lista_cmd:
        tipo = cmd[0]

        # AÇÕES SIMPLES
        if tipo in ("ligar", "desligar"):
            _, dispositivo = cmd
            if dispositivo not in nomes:
                erros.append(f"Dispositivo '{dispositivo}' nao existe.")
```