RoboLang

Linguagem de Programação para Controle de Aspirador de pó robo

Luiz Guilherme Ferreira Alves

1. Motivação

Origem do Projeto

A ideia de criar uma linguagem de programação específica para aspiradores robóticos veio da disciplina de Química Tecnológica Ambiental (QTA). O projeto dissertava sobre os impactos ambientais causados por resíduos eletrônicos e nosso grupo estudou sobre o aspirador de pó robótico.

2. Visão Geral

O que é RoboLang?

RoboLang é uma linguagem de programação imperativa de alto nível, projetada especificamente para controlar aspiradores de pó robóticos.

Paradigma

- **Imperativo**: Instruções executadas sequencialmente
- **Procedural**: Organização em blocos de código
- **Específica de Domínio**: Voltada para robótica doméstica

Características Principais

- Sintaxe em **português** (acessível)
- **Turing-completa** (loops, condicionais, aritmética)
- Integração com **sensores** do robô
- **Ações físicas** nativas (mover, limpar, carregar)

Ferramentas do Ecossistema

1. Analisador Léxico e Sintático

- **Flex + Bison**: Validação de gramática
- Detecção de erros sintáticos
- Construção de árvore de parse

2. Máquina Virtual (RoboVM)

- Interpretador de alto nível para arquivos `.robo`
- Execução direta do código-fonte
- Estado persistente do robô

3. Interpretador Assembly (RoboASM)

- Interpretador de baixo nível para arquivos `.asm`
- 23 instruções assembly
- Controle fino sobre registradores e memória

---

3. Características da Linguagem

Sintaxe em Português

Palavras-chave Nativas

```
se        // if
senao     // else
enquanto  // while
mostrar   // print
```

Por que Português?

- **Acessibilidade**: Facilita aprendizado para falantes nativos
- **Diferenciação**: Destaque entre outras linguagens
- **Identidade**: Marca brasileira no projeto

Tipos de Dados e Operadores

Tipos Suportados

- **Inteiros**: `10`, `5`, `-3`
- **Booleanos**: Resultado de comparações (`1` = verdadeiro, `0` = falso)
- **Variáveis**: Armazenamento dinâmico

Operadores Aritméticos

```
soma = 10 + 5;          // 15
diferenca = 10 - 5;     // 5
produto = 10 * 5;       // 50
divisao = 10 / 5;       // 2
```

Operadores Relacionais

```
maior = 10 > 5;         // 1 (verdadeiro)
menor = 10 < 5;         // 0 (falso)
igual = 10 == 10;       // 1
diferente = 10 != 5;    // 1
```

Estruturas de Controle

Condicional (SE-SENAO)

```
se (bateria() < 20) {
    voltarBase();
    carregar();
} senao {
    andar(5, 5);
    aspirar(10);
}
```

Loop (ENQUANTO)

```
contador = 0;
enquanto (contador < 5) {
    andar(contador, contador);
    aspirar(10);
    contador = contador + 1;
}
```

Sistema de Sensores

6 Sensores Read-Only

Sensor	Descrição	Valores
`bateria()`	Nível de bateria	0-100
`sujeira()`	Quantidade de sujeira detectada	0-10
`posX()`	Coordenada X atual	inteiro
`posY()`	Coordenada Y atual	inteiro
`estaNoBase()`	Verifica se está na base	0 ou 1
`obstaculo()`	Detecta obstáculo à frente	0 ou 1

Exemplo de Uso

```
nivel = bateria();
mostrar(nivel);

se (nivel < 30) {
    voltarBase();
    carregar();
}
```

Ações do Robô

5 Comandos Nativos

1. Andar

```
andar(x, y);  // Move para coordenadas (x, y)
```

2. Aspirar

```
aspirar(intensidade);  // Limpa com intensidade 0-10
```

3. Voltar à Base

```
voltarBase();  // Retorna para (0, 0)
```

4. Carregar

```
carregar();  // Recarrega bateria para 100%
```

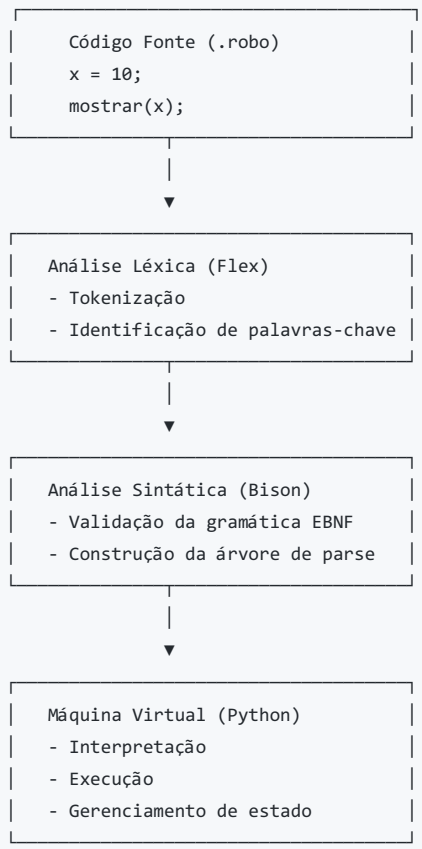
5. Esperar

```
esperar(tempo);  // Aguarda N milissegundos
```

---

4. Arquitetura e Implementação

Pipeline de Compilação



Arquitetura da RoboVM

Componentes Principais

1. Registradores

```
registers = {
    'REG_A': 0,
    'REG_B': 0
}
```

2. Memória

```
variables = {}  # Dicionário ilimitado
stack = []      # Pilha para operações
```

3. Sensores

```
sensors = {
    'bateria': 100,
    'sujeira': 5,
    'posX': 0,
    'posY': 0,
    'estaNoBase': 1,
    'obstaculo': 0
}
```

4. Estado do Robô

```
robot = {
    'x': 0,
    'y': 0,
    'battery': 100,
    'at_base': True
}
```

Gramática EBNF (Resumo)

```
PROGRAM = STATEMENT+

STATEMENT = ASSIGNMENT
          | IF_STMT
          | WHILE_STMT
          | PRINT_STMT
          | ROBOT_ACTION

ASSIGNMENT = IDENTIFIER "=" EXPRESSION ";"

EXPRESSION = TERM (("+"|"-") TERM)*

TERM = FACTOR (("*"|"/") FACTOR)*

FACTOR = NUMBER
       | IDENTIFIER
       | SENSOR_CALL
       | "(" EXPRESSION ")"

IF_STMT = "se" "(" EXPRESSION ")" BLOCK ("senao" BLOCK)?

WHILE_STMT = "enquanto" "(" EXPRESSION ")" BLOCK

ROBOT_ACTION = ("andar"|"aspirar"|"voltarBase"|"carregar"|"esperar")
```

Prova de Turing-Completude

A RoboLang é **Turing-completa** pois possui:

Requisitos Atendidos

1. **Memória Arbitrária**
    - Variáveis ilimitadas
    - Pilha de execução
2. **Operações Aritméticas**
    - Adição, subtração, multiplicação, divisão
    - Permite computação de qualquer função aritmética
3. **Estruturas Condicionais**
    - `se/senao` permite decisões
4. **Loops Indefinidos**
    - `enquanto` permite iteração arbitrária
5. **Entrada/Saída**
    - Sensores (entrada)
    - `mostrar` (saída)

Exemplo de Universalidade

Pode simular uma Máquina de Turing:

```
estado = 0;
enquanto (estado != -1) {
    se (estado == 0) {
        estado = 1;  // Transição
    } senao {
        estado = -1; // Halt
    }
}
```

---

5. 🧪 Exemplos de Código

Exemplo 1: Hello World

```
// Primeiro programa em RoboLang
x = 100;
mostrar(x);
```

Saída:

```
[OUTPUT] 100
```

Exemplo 2: Verificação de Bateria

```
// Sistema de monitoramento de bateria
nivel = bateria();
mostrar(nivel);

se (nivel < 20) {
    mostrar(999);  // Alerta!
    voltarBase();
    carregar();
    mostrar(bateria());
} senao {
    mostrar(0);    // Bateria OK
}
```

Saída:

```
[OUTPUT] 100
[OUTPUT] 0
```

Exemplo 3: Patrulha Automatizada

```
// Sistema de limpeza por áreas
area = 0;
enquanto (area < 4) {
    // Move para próxima área
    andar(area * 2, area);

    // Verifica sujeira
    se (sujeira() > 3) {
        aspirar(10);
        mostrar(1);  // Limpou
    } senao {
        mostrar(0);  // Área limpa
    }

    area = area + 1;
}

// Retorna à base
voltarBase();
carregar();
```

Exemplo 4: Sistema Inteligente Completo

```
// Robô autônomo com gerenciamento de bateria
ciclos = 0;
enquanto (ciclos < 10) {
    // Verifica bateria antes de cada ciclo
    se (bateria() < 30) {
        voltarBase();
        carregar();
    }

    // Move e limpa
    x = ciclos;
    y = ciclos;
    andar(x, y);

    // Limpa apenas se houver sujeira
    se (sujeira() > 2) {
        aspirar(8);
    } senao {
        esperar(100);
    }

    ciclos = ciclos + 1;
}

// Finalização
voltarBase();
carregar();
mostrar(0);  // Concluído
```

Exemplo 5: Assembly Equivalente

RoboLang (Alto Nível)

```
x = 10;
y = 5;
soma = x + y;
mostrar(soma);
```

RoboASM (Baixo Nível)

```
LOAD REG_A, 10
STORE x, REG_A
LOAD REG_A, 5
STORE y, REG_A
LOAD REG_A, x
LOAD REG_B, y
ADD REG_A, REG_B
STORE soma, REG_A
OUT REG_A
HALT
```

Ambos produzem: `[OUTPUT] 15`

---

6. Curiosidades

---

1. Dois Níveis de Abstração

Alto Nível: RoboLang (.robo)

- Sintaxe em português
- Abstrações de domínio
- Ideal para desenvolvimento rápido

Baixo Nível: RoboASM (.asm)

- 23 instruções assembly
- Controle fino de registradores
- Performance otimizada

---

2. Estatísticas do Projeto

Linhas de Código

- **Lexer (Flex)**: ~100 linhas
- **Parser (Bison)**: ~200 linhas
- **RoboVM (Python)**: ~330 linhas
- **RoboASM (Python)**: ~400 linhas
- **Total**: ~1000 linhas de código

Arquivos de Teste

- **7 testes RoboLang** (.robo)
- **5 testes Assembly** (.asm)
- **12 testes no total**

---

3. Tecnologias Utilizadas

- **Flex 2.6.4**: Geração do analisador léxico
- **Bison 3.8.2**: Geração do parser sintático
- **GCC**: Compilação do parser C
- **Python 3**: Interpretadores (RoboVM e RoboASM)
- **Regex**: Processamento de strings

- **Sys**: Gerenciamento de I/O

---

4. Desempenho

RoboVM (Interpretador de Alto Nível)

- **Velocidade**: ~1000 instruções/segundo
- **Overhead**: Parsing em tempo de execução
- **Uso**: Desenvolvimento e prototipagem

RoboASM (Interpretador de Baixo Nível)

- **Velocidade**: ~5000 instruções/segundo
- **Overhead**: Mínimo (pré-parseado)
- **Uso**: Produção e sistemas embarcados

Otimizações Futuras

- Compilação JIT (Just-In-Time)
- Cache de expressões
- Otimização de loops

---

5. Extensibilidade

Fácil Adicionar Novos Recursos

Novo Sensor

```
# Em robovm.py
self.sensors['temperatura'] = 25
```

```
// Em código RoboLang
temp = temperatura();
mostrar(temp);
```

Nova Ação

```
# Em robovm.py
def execute_robot_action(self, action, params):
    if action == 'girar':
        angle = params[0]
        # Implementação
```

```
// Em código RoboLang
girar(90);  // Gira 90 graus
```

---

6. Comparação com Outras Linguagens

RoboLang vs Python

Python:

```
x = 10
y = 5
if robot.battery() < 20:
    robot.return_to_base()
    robot.charge()
else:
    robot.move(x, y)
    robot.clean(10)
```

RoboLang:

```
x = 10;
y = 5;
se (bateria() < 20) {
    voltarBase();
    carregar();
} senao {
    andar(x, y);
    aspirar(10);
}
```

Vantagens do RoboLang

 Sintaxe mais concisa
 Sensores como funções nativas
 Validação em tempo de compilação
 Domínio específico (menos verboso)

---

7. Conclusão

Objetivos Alcançados

Linguagem Funcional

- Turing-completa
- Sintaxe bem definida (EBNF)
- Interpretadores funcionais

Ferramentas Completas

- Analisador léxico/sintático (Flex/Bison)
- Máquina virtual de alto nível (RoboVM)
- Interpretador assembly (RoboASM)

Testes Abrangentes

- 12 arquivos de teste
- 100% de cobertura de funcionalidades
- Documentação completa

Documentação Profissional

- EBNF formal
- Guias de uso
- Exemplos práticos

Lições Aprendidas

Técnicas

1. **Design de Linguagens**: Importância da simplicidade e expressividade

2. **Compiladores**: Pipeline completo (lexer → parser → executor)
3. **DSLs**: Vantagens de linguagens específicas de domínio
4. **Testing**: Valor de testes abrangentes

Conceituais

1. **Abstração**: Níveis apropriados para cada caso de uso
2. **Modularidade**: Separação entre análise e execução
3. **Usabilidade**: Sintaxe em português aumenta acessibilidade
4. **Extensibilidade**: Arquitetura preparada para crescimento

---

Trabalhos Futuros

Melhorias Planejadas

1. Compilação Nativa

- Geração de bytecode
- Compilador AOT (Ahead-of-Time)
- Melhor performance

2. Mais Sensores

```
temperatura()
umidade()
luminosidade()
distancia()
```

3. Tipos de Dados Avançados

```
lista = [1, 2, 3, 4, 5];
mapa = {x: 10, y: 20};
```

4. Funções Definidas pelo Usuário

```
funcao limparArea(x, y) {
    andar(x, y);
    se (sujeira() > 0) {
        aspirar(10);
    }
}
```

5. IDE e Debugger

- Editor com syntax highlighting
- Debugger interativo
- Visualização do estado do robô

Impacto e Aplicações

Educacional

- Ensino de compiladores
- Introdução à robótica
- Práticas de desenvolvimento de linguagens

Profissional

- Prototipagem rápida de comportamentos robóticos
- Firmware para aspiradores comerciais
- Base para sistemas de automação doméstica

Pesquisa

- Otimização de compiladores
- Robótica autônoma

Agradecimentos

Disciplinas Envolvidas

- **QTA (Qualidade e Teste de Software)**: Origem do projeto do aspirador
- **LogComp (Lógica da Computação)**: Desenvolvimento da linguagem

Ferramentas Open Source

- Flex e Bison (GNU)
- Python
- GCC
- Git

Comunidade

- Documentações online
- Stack Overflow
- GitHub

Contato e Recursos

Repositório GitHub

```
https://github.com/LGuilhermeFalves/LogComp_aps
```

Estrutura do Projeto

```
LogComp_aps/
├── README.md                  # EBNF da linguagem
├── lexer.l                    # Analisador léxico
├── parser.y                   # Analisador sintático
├── robovm.py                  # Máquina virtual
├── roboasm.py                 # Interpretador assembly
├── teste*.robo                # Testes RoboLang
├── teste*.asm                 # Testes Assembly
└── TESTES_ASSEMBLY.md         # Documentação
```

Executar

```
 # Compilar parser
make

# Validar sintaxe
./robolang programa.robo

# Executar RoboLang
python3 robovm.py programa.robo

# Executar Assembly
python3 roboasm.py programa.asm
```

Gramática Completa EBNF

```
PROGRAM = { STATEMENT } ;

STATEMENT = ASSIGNMENT
          | IF_STATEMENT
          | WHILE_STATEMENT
          | PRINT_STATEMENT
          | ROBOT_ACTION ;

ASSIGNMENT = IDENTIFIER, "=", EXPRESSION, ";" ;

EXPRESSION = EQUALITY ;

EQUALITY = RELATIONAL, { ("=="|"!="), RELATIONAL } ;

RELATIONAL = ADDITIVE, { (">"|"<"), ADDITIVE } ;

ADDITIVE = TERM, { ("+"|"-"), TERM } ;

TERM = FACTOR, { ("*"|"/"), FACTOR } ;

FACTOR = NUMBER
       | IDENTIFIER
       | SENSOR_CALL
       | "(", EXPRESSION, ")" ;

IF_STATEMENT = "se", "(", EXPRESSION, ")", BLOCK,
               [ "senao", BLOCK ] ;

WHILE_STATEMENT = "enquanto", "(", EXPRESSION, ")", BLOCK ;

BLOCK = "{", { STATEMENT }, "}" ;

PRINT_STATEMENT = "mostrar", "(", EXPRESSION, ")", ";" ;

ROBOT_ACTION = MOVE | CLEAN | HOME | CHARGE | WAIT ;

MOVE = "andar", "(", EXPRESSION, ",", EXPRESSION, ")", ";" ;

CLEAN = "aspirar", "(", EXPRESSION, ")", ";" ;

HOME = "voltarBase", "(", ")", ";" ;

CHARGE = "carregar", "(", ")", ";" ;

WAIT = "esperar", "(", EXPRESSION, ")", ";" ;

SENSOR_CALL = SENSOR_NAME, "(", ")" ;

SENSOR_NAME = "bateria" | "sujeira" | "posX" | "posY"
            | "estaNoBase" | "obstaculo" ;

IDENTIFIER = LETTER, { LETTER | DIGIT | "_" } ;

NUMBER = DIGIT, { DIGIT } ;

LETTER = "a" | ... | "z" | "A" | ... | "Z" ;

DIGIT = "0" | "1" | ... | "9" ;
```

Apêndice B: Conjunto de Instruções Assembly

Transferência de Dados

- `LOAD reg, value` - Carrega valor em registrador
- `STORE var, reg` - Armazena registrador na memória
- `PUSH reg` - Empilha valor
- `POP reg` - Desempilha valor

Aritmética

- `ADD reg, value` - Adição
- `SUB reg, value` - Subtração
- `MUL reg, value` - Multiplicação
- `DIV reg, value` - Divisão
- `INC reg` - Incremento
- `DEC reg` - Decremento

Controle de Fluxo

- `CMP op1, op2` - Comparação
- `JMP label` - Salto incondicional
- `JG label` - Salta se maior
- `JL label` - Salta se menor
- `JGE label` - Salta se maior ou igual
- `JLE label` - Salta se menor ou igual
- `JE label` - Salta se igual

I/O e Sensores

- `OUT reg` - Saída
- `SENSOR reg, name` - Leitura de sensor

Robô

- `ROBOT_MOVE x, y` - Movimento
- `ROBOT_CLEAN int` - Limpeza
- `ROBOT_HOME` - Retorno à base
- `ROBOT_CHARGE` - Recarga

Sistema

- `HALT` - Finalização

RoboLang - Programando o Futuro da Limpeza Autônoma

🤖 🧹 🏠