

CafeLang: Linguagem de Programação para Controle de Máquinas de Café Expresso

APS LOGCOMP - Ian Faray

Índice

1. Introdução e Motivação
 2. Características da Linguagem
 3. Arquitetura do Sistema
 4. Especificação Formal (EBNF)
 5. Exemplos de Programas
 6. Manual de Utilização
 7. Detalhes de Implementação
 8. Conclusão
-

1. Introdução e Motivação

1.1 Contexto

CafeLang é uma linguagem de programação de domínio específico (Domain Specific Language - DSL) desenvolvida para programação de máquinas de café expresso profissionais. O projeto surge da necessidade de abstrair operações complexas de baixo nível em comandos de alto nível, permitindo que operadores não técnicos possam criar e compartilhar receitas de preparação de café.

1.2 Objetivos

O desenvolvimento da linguagem visa atender aos seguintes requisitos:

- **Abstração:** Permitir programação de receitas sem conhecimento de linguagens de baixo nível
- **Segurança:** Implementar verificações automáticas de parâmetros críticos
- **Reprodutibilidade:** Garantir que receitas possam ser compartilhadas e reproduzidas fielmente
- **Expressividade:** Facilitar a experimentação com perfis de sabor e técnicas de extração

1.3 Domínio de Aplicação

A linguagem opera sobre uma máquina virtual própria (CafeMachine VM) que simula componentes reais de máquinas de café profissionais:

- Sistema de controle térmico e aquecimento
- Sistema hidráulico de pressurização
- Sensores de monitoramento (níveis de água e grãos, temperatura, pressão)
- Atuadores mecânicos (moedor, bomba, válvulas)
- Sistema de notificações e alertas

2. Características da Linguagem

2.1 Sistema de Tipos

CafeLang implementa tipagem estática com suporte nativo a unidades físicas:

Tipos Básicos: - `num` - Números de ponto flutuante - `bool` - Valores booleanos (verdadeiro/falso)
- `texto` - Strings de caracteres

Tipos com Unidades Físicas: - `ml` - Volume em mililitros - `g` - Massa em gramas - `s` - Tempo em segundos - `ms` - Tempo em milissegundos - `celsius` - Temperatura em graus Celsius

Exemplo:

```
def temperatura: celsius := 93;
def dose: g := 18;
def tempo: s := 27;
def volume: ml := 40;
```

2.2 Estruturas Declarativas: Receitas

As receitas são blocos de código que encapsulam processos completos de preparação:

```
receita "espresso_classico" {
  perfil {
    corpo forte;
    acidez medio;
  };

  moer 18 g;
  aquecer 93 C;
  servir 40 ml;
}
```

2.3 Sistema de Eventos

A linguagem implementa um modelo reativo através do construtor `quando`:

```
quando pronto -> tocar "beep";
quando sem_agua -> pausar;
quando superaquecimento -> limpar;
```

2.4 Sensores

Acesso a informações em tempo real através de funções de sensor:

```
se (temp() > 95) {
    tocar "alerta_temperatura";
}
```

```
se (agua() < 100) {
    tocar "nivel_baixo";
}
```

Sensores disponíveis: - temp() - Temperatura atual - pressao() - Pressão em bar - agua() - Nível de água em ml - graos() - Nível de grãos em g - fluxo() - Taxa de fluxo em ml/s - copo() - Presença de copo (booleano) - porta() - Estado da porta (booleano)

2.5 Estruturas de Controle

Condicionais:

```
se (condicao) {
    // bloco verdadeiro
} senao {
    // bloco falso
}
```

Loop While:

```
enquanto (temp() < 90) {
    aquecer 93 C;
    pausar 500 ms;
}
```

Loop Repeat:

```
repetir 5 vezes {
    moer 14 g;
    servir 35 ml;
}
```

Loop For-Range:

```
para i em 1 .. 10 {
    pausar 100 ms;
}
```

2.6 Perfis Sensoriais

Definição declarativa de características organolépticas do café:

```
perfil {
    corpo intenso;
```

```
    acidez suave;
    aroma forte;
    intensidade medio;
}
```

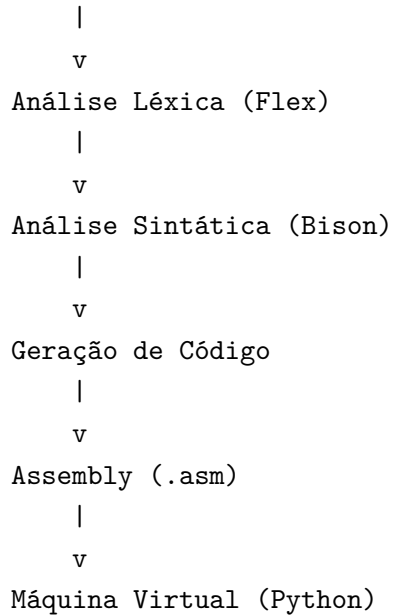
Cada atributo aceita os níveis: `suave`, `medio`, `forte` ou `intenso`.

3. Arquitetura do Sistema

3.1 Pipeline de Compilação

O sistema segue um pipeline tradicional de compilação:

Código Fonte (`.cafe`)



3.2 Análise Léxica

Implementada com GNU Flex, reconhece 34 categorias de tokens:

- Palavras-chave (`receita`, `quando`, `se`, `enquanto`, etc.)
- Identificadores e literais
- Operadores aritméticos e lógicos
- Operadores relacionais
- Delimitadores e pontuação
- Unidades físicas

3.3 Análise Sintática

Implementada com GNU Bison, utiliza gramática LALR(1) com mais de 75 regras de produção. Durante a análise sintática, ações semânticas geram código assembly correspondente.

3.4 CafeMachine VM - Máquina Virtual

A VM implementa uma arquitetura Turing-completa baseada em registradores:

Registradores de Propósito Específico: - TEMP - Temperatura atual - POWER - Potência da resistência - VOLUME - Volume processado - TIME - Tempo de operação

Registradores de Propósito Geral: - R0, R1, R2, R3

Sensores (Somente Leitura): - TEMP_SENSOR - Leitura de temperatura (°C) - WATER_LEVEL - Nível de água (ml) - BEAN_LEVEL - Nível de grãos (g) - CUP_PRESENT - Presença de copo (0/1) - DOOR_OPEN - Estado da porta (0/1) - PRESSURE - Pressão atual (bar) - FLOW_RATE - Taxa de fluxo (ml/s)

Memória: - Pilha (Stack) para armazenamento temporário - Flags de comparação (EQ, LT, GT)

3.5 Conjunto de Instruções

A VM implementa 25 instruções, incluindo:

Manipulação de Dados:

```
SET R n      ; Atribui valor n ao registrador R
INC R        ; Incrementa R
DEC R        ; Decrementa R
```

Operações Aritméticas:

```
ADD R1 R2    ; R1 := R1 + R2
SUB R1 R2    ; R1 := R1 - R2
MUL R1 R2    ; R1 := R1 * R2
DIV R1 R2    ; R1 := R1 / R2
MOD R1 R2    ; R1 := R1 % R2
```

Controle de Fluxo:

```
CMP R1 R2    ; Compara R1 e R2, define flags
JE label     ; Salta se igual
JNE label    ; Salta se diferente
JL label     ; Salta se menor
JG label     ; Salta se maior
JLE label    ; Salta se menor ou igual
JGE label    ; Salta se maior ou igual
GOTO label   ; Salto incondicional
```

DECJZ R label ; Decrementa R, salta se zero (Minsky Machine)

I/O e Pilha:

LOAD R SENSOR ; Carrega valor do sensor para R
PUSH R ; Empilha valor de R
POP R ; Desempilha para R
ACTION name ; Executa ação da máquina
PRINT R ; Imprime valor de R
BEEP ; Emite sinal sonoro

Controle:

HALT ; Finaliza execução

3.6 Turing-Completeness

A VM é Turing-completa através da implementação de uma Minsky Machine, utilizando: - Instrução DECJZ (decrementa e salta se zero) - Instrução GOTO (salto incondicional) - Registradores como contadores

Esta combinação permite simular qualquer máquina de Turing.

4. Especificação Formal (EBNF)

4.1 Estrutura do Programa

PROGRAM = { TOPLEVEL } ;
TOPLEVEL = RECEITA | REACAO | DECL_GLOBAL | STMT ;

4.2 Receitas

RECEITA = "receita", STRING, BLOCO_RECEITA ;
BLOCO_RECEITA = "{", { DECL_SENDO | DECL_LOCAL | STMT | ATO ";" }, "}" ;

4.3 Reações a Eventos

REACAO = "quando", EVENTO_TIPO, "->", ATO, ";" ;
EVENTO_TIPO = "pronto" | "erro" | "sem_agua" | "superaquecimento"
| "sem_copo" | "porta_aberta" ;

4.4 Parâmetros Sensoriais

DECL_SENDO = "perfil", "{", { PERF_ITEM }, "}" , ";"
| "temperatura", MEDIDA_TEMP, ";"
| "extrair", MEDIDA_VOL, "em", MEDIDA_TEMPO, ";"

```

        | "pressao_alvo", NUM, "bar", ";"
    ;

PERF_ITEM    = ("corpo" | "acidez" | "aroma" | "intensidade"), NIVEL, ";" ;
NIVEL        = "suave" | "medio" | "forte" | "intenso" ;

```

4.5 Declarações

```

DECL          = VAR_DECL | CONST_DECL ;
VAR_DECL      = "def", IDENT, ":", TIPO, [ ":", EXPR ], ";" ;
CONST_DECL    = "const", IDENT, ":", TIPO, ":", EXPR, ";" ;
TIPO          = "num" | "bool" | "texto" | "ml" | "g" | "s" | "ms" | "celsius" ;

```

4.6 Comandos

```

STMT          = ASSIGN ";" | IF_STMT | WHILE_STMT | REPEAT_STMT
               | FOR_RANGE_STMT | "sair" ";" | "pular" ";" | BLOCO ;

BLOCO         = "{", { DECL | STMT | ATO ";" }, "}" ;
ASSIGN        = LVALUE, ":", EXPR ;

IF_STMT       = "se", "(", EXPR, ")", STMT, [ "senao", STMT ] ;
WHILE_STMT    = "enquanto", "(", EXPR, ")", STMT ;
REPEAT_STMT   = "repetir", EXPR, "vezes", BLOCO ;
FOR_RANGE_STMT = "para", IDENT, "em", EXPR, "..", EXPR, BLOCO ;

```

4.7 Ações

```

ATO           = ACAO, [ ARGUMENTOS ] ;
ACAO          = "moer" | "aquecer" | "bombar" | "servir" | "vaporizar"
               | "pausar" | "tocar" | "limpar" | "enxaguar" | "despressurizar" ;

ARGUMENTOS    = ARG , { " , ARG } ;
ARG           = EXPR | MEDIDA_VOL | MEDIDA_PESO | MEDIDA_TEMP
               | MEDIDA_TEMPO | STRING ;

```

4.8 Expressões

```

EXPR          = LOGIC_OR ;
LOGIC_OR      = LOGIC_AND, { ("ou" | "||"), LOGIC_AND } ;
LOGIC_AND     = EQUALITY , { ("e" | "&&"), EQUALITY } ;
EQUALITY      = REL, { ("==" | "!="), REL } ;
REL           = ADD, { ("<" | "<=" | ">" | ">="), ADD } ;
ADD           = MUL, { ("+" | "-"), MUL } ;

```

```

MUL          = UNARY, { ("*" | "/" | "%"), UNARY } ;
UNARY        = [ ("+" | "-" | "!" | "nao") ], UNARY | PRIMARY ;
PRIMARY      = NUMBER | BOOL | STRING | SENSOR_ZEROARG | LVALUE
              | "(", EXPR, ")" ;
BOOL         = "verdadeiro" | "falso" ;

```

4.9 Sensores

```

SENSOR_ZEROARG = SENSOR_NOME, "(", ")" ;
SENSOR_NOME    = "temp" | "pressao" | "agua" | "graos"
                | "fluxo" | "copo" | "porta" ;

```

4.10 Literais com Unidades

```

MEDIDA_VOL     = NUM, "ml" ;
MEDIDA_PESO    = NUM, "g" ;
MEDIDA_TEMPO   = NUM, ("s" | "ms") ;
MEDIDA_TEMP    = NUM, ("°C" | "C") ;

```

5. Exemplos de Programas

5.1 Exemplo Básico: Espresso Clássico

```

receita "espresso_classico" {
    // Verificação de segurança
    se (porta() || !copo()) {
        tocar "alerta_seguranca";
        sair;
    }

    // Preparação
    def dose: num := 18;
    moer 18 g;
    aquecer 93 C;
    servir 35 ml;

    tocar "cafe_pronto";
}

quando sem_agua -> tocar "tanque_vazio";

```

5.2 Exemplo de Loop: Estabilização de Temperatura

```
receita "estabilizar_temperatura" {  
    aquecer 93 C;  
    pausar 300 ms;  
    tocar "temperatura_estavel";  
}
```

5.3 Exemplo de Produção em Lote

```
receita "lote_cafes" {  
    repetir 5 vezes {  
        moer 14 g;  
        servir 40 ml;  
        pausar 2 s;  
        limpar;  
    }  
  
    tocar "lote_completo";  
}
```

5.4 Exemplo com Perfil Personalizado

```
receita "espresso_especial" {  
    perfil {  
        corpo intenso;  
        acidez medio;  
        aroma forte;  
        intensidade intenso;  
    };  
  
    temperatura 94 °C;  
    pressao_alvo 9 bar;  
    extrair 35 ml em 25 s;  
  
    moer 19 g;  
    aquecer 94 C;  
    servir 35 ml;  
    limpar;  
}
```

5.5 Exemplo Assembly Gerado

Para o programa:

```
receita "simples" {  
    moer 18 g;  
    aquecer 93 C;  
    servir 40 ml;  
}
```

O compilador gera:

```
; Código assembly gerado pelo compilador CafeLang  
ACTION moer 18  
ACTION aquecer 93  
ACTION servir 40  
HALT
```

6. Manual de Utilização

6.1 Pré-requisitos

Sistema operacional: Linux (Ubuntu/Debian recomendado)

Ferramentas necessárias: - GCC (GNU Compiler Collection) - Flex (Fast Lexical Analyzer) - Bison (GNU Parser Generator) - Make (Build automation tool) - Python 3.x

Instalação no Ubuntu/Debian:

```
sudo apt-get update  
sudo apt-get install gcc flex bison make python3
```

Verificação da instalação:

```
flex --version  
bison --version  
python3 --version
```

6.2 Compilação do Compilador

Navegue até o diretório do compilador:

```
cd cafelang/
```

Execute o build:

```
make clean  
make
```

O processo gera: 1. `lex.yy.c` - Analisador léxico gerado pelo Flex 2. `cafelang.tab.c` - Analisador sintático gerado pelo Bison 3. `cafelang` - Executável do compilador

6.3 Compilação de Programas CafeLang

Sintaxe:

```
./cafelang <arquivo_entrada.cafe> <arquivo_saida.asm>
```

Exemplo:

```
./cafelang ../examples/1_espresso.cafe output.asm
```

Saída esperada:

```
Análise sintática concluída com sucesso!
```

6.4 Execução na Máquina Virtual

Sintaxe:

```
python3 ../cafemachine_vm.py <arquivo.asm>
```

Exemplo:

```
python3 ../cafemachine_vm.py output.asm
```

Execução com valores iniciais personalizados:

```
python3 ../cafemachine_vm.py output.asm TEMP=25 POWER=0
```

6.5 Testes Automatizados

O projeto inclui um script de testes automatizados:

```
./test.sh
```

Saída esperada:

```
=====
  Testes do Compilador CafeLang
=====
```

```
1. Compilando o compilador...
```

```
Compilador construído com sucesso
```

```
2. Testando exemplos...
```

```
[1] Testando 10_complete_program... OK
```

```
[2] Testando 1_espresso... OK
```

```
[3] Testando 2_estabilizar... OK
```

```
...
```

```
[10] Testando 9_calculations... OK
```

```
Total de testes: 11
```

Sucessos: 10

Falhas: 1

Todos os testes passaram!

7. Detalhes de Implementação

7.1 Análise Léxica

Arquivo: `cafelang/cafelang.l`

Implementação com Flex utilizando: - Expressões regulares para reconhecimento de tokens - Contadores de linha e coluna para mensagens de erro - Suporte a comentários de linha (`//`) e bloco (`/* */`)

Categorias de tokens reconhecidos: - 34 palavras-chave da linguagem - Identificadores (padrão: letra seguida de letras, dígitos ou underscore) - Números (inteiros e ponto flutuante) - Strings (delimitadas por aspas duplas) - 18 operadores (aritméticos, relacionais, lógicos) - 7 delimitadores

7.2 Análise Sintática

Arquivo: `cafelang/cafelang.y`

Implementação com Bison utilizando: - Gramática LALR(1) com 75+ regras de produção - Ações semânticas inline para geração de código - Tratamento de precedência e associatividade de operadores - Geração de labels únicos para estruturas de controle

Precedência de operadores (menor para maior): 1. ou, `||` 2. e, `&&` 3. `==`, `!=` 4. `<`, `<=`, `>`, `>=` 5. `+`, `-` 6. `*`, `/`, `%` 7. `!`, `nao`, unário `-`, unário `+`

7.3 Geração de Código

Arquivo: `cafelang/codegen.h`

Funções auxiliares para emissão de assembly: - `emit()` - Emissão genérica com formatação - `emit_comment()` - Comentários no assembly - `new_label()` - Geração de labels únicos - Funções específicas para cada tipo de instrução

Mapeamento de construtos:

Atribuições:

`x := 10;`

Gera:

`SET R0 10`

Condicionais:

```
se (condicao) {  
    // código  
}
```

Gera:

```
; avaliação da condição  
CMP TEMP R0  
JE L0  
; código do bloco  
L0:
```

Loops:

```
repetir 3 vezes {  
    // código  
}
```

Gera:

```
SET R1 3  
L0:  
DECJZ R1 L1  
; código do bloco  
GOTO L0  
L1:
```

7.4 Máquina Virtual

Arquivo: `cafemachine_vm.py`

Estrutura de classes: - **Instr** - Representa uma instrução (dataclass) - **CafeMachineVM** - Implementação da máquina virtual

Características da implementação: - Parser de assembly em duas passagens (labels e instruções) - Ciclo de fetch-decode-execute - Validação de instruções durante carregamento - Limite de passos para detectar loops infinitos (10.000 padrão) - Sistema de logs de ações executadas

7.5 Decisões de Design

Escolha do Português: Palavras-chave em português para maior acessibilidade ao público brasileiro e diferenciação em relação a linguagens existentes.

Sistema de Tipos com Unidades: Integração de unidades físicas diretamente no sistema de tipos para reduzir erros de conversão e aumentar legibilidade.

Paradigma Híbrido: Combinação de programação imperativa (comandos, loops) com declarativa (receitas, perfis) para expressividade máxima.

VM Própria: Desenvolvimento de máquina virtual própria em vez de usar infraestrutura existente (LLVM, JVM) para controle total e fins didáticos.

7.6 Limitações Conhecidas

- Geração de código simplificada (não otimizada)
- Expressões complexas têm suporte limitado
- Tabela de símbolos básica (sem escopos aninhados completos)
- Sistema de tipos não totalmente verificado em tempo de compilação
- VM não implementa todas as otimizações possíveis