# Especificação Semântica e Geração de Código - Linguagem E-moji

**Discentes:** Fernando Seiji Onoda Inomata, Lucas Duarte, Vitor Mayorca Camargo.

## 1. Introdução

O presente documento especifica as os aspectos semânticos e a estratégia de geração de código intermediário para a linguagem de programação E-moji. Aqui são especificadas regras, e erros semânticos que não podem ser capturados durante as etapas de compilação anteriores.

Após a validação semântica, o compilador gera um código de três endereços (TAC) como representação intermediária.

## 2. Sistema de Tipos

A linguagem E-moji utiliza um sistema de Tipagem Forte e Estática. Ou seja:

1. Toda variável deve ter um tipo declarado explicitamente antes de ser usada;
2. O tipo de uma variável não pode ser alterado durante a execução;
3. Não há conversão implícita de tipos;
4. Operações entre tipos incompatíveis resultarão em erros de compilação.

### 2.1 Tipos Primitivos Suportados

De acordo com o documento de especificação léxica da linguagem entregue anteriormente, os tipos básicos suportados são:

| Tipo | Token | Descrição | Valor Padrão (Geração de Código) |
|---|---|---|---|
| **Inteiro** | 🔢 (INT) | Números inteiros com sinal. | 0 |
| **Booleano** | 🤥 (BOOL) | Valores lógicos 👍 (TRUE) ou 👎 (FALSE). | 0 (False) |
| **String** | 🔤 (STRING) | Cadeia de caracteres literais. | "" (Vazio) |

### 2.2 Regras de Compatibilidade

As operações só são permitidas entre operandos do mesmo tipo, conforme a tabela abaixo:

- **Aritmética (➕, ➖, ✖️, ➗):** Ambos os operandos devem ser INT. O resultado é INT.
- **Relacional (🐓, 🐣, 🥚):**
  - Para 🐓 (Maior) e 🐣 (Menor): Ambos os operandos devem ser INT. O resultado é BOOL.
  - Para 🥚 (Igualdade): Os operandos devem ser do mesmo tipo (INT vs INT ou BOOL vs BOOL). O resultado é BOOL.
- **Lógica (🤏, ✌️):** Ambos os operandos devem ser BOOL. O resultado é BOOL.
- **Atribuição (🎁):** O tipo da expressão à direita deve ser idêntico ao tipo da variável à esquerda.

## 3. Tabela de Símbolos e Regras de Escopo

A tabela de símbolos será a estrutura central para armazenar informações sobre os identificadores (variáveis).

### 3.1 Estrutura do Escopo

A linguagem adota o Escopo Estático (Léxico), similar às linguagens C e Java.

- **Delimitadores:** O escopo é delimitado pelos tokens de bloco: 🤜 (Início) e 🤛 (Fim).
- **Hierarquia:** Blocos podem ser aninhados. Um bloco interno tem acesso às variáveis declaradas nos blocos externos (ancestrais).
- **Visibilidade:** Uma variável declarada em um bloco interno não é visível nos blocos externos.

### 3.2 Regras de Declaração

1. **Declaração Obrigatória:** Toda variável deve ser declarada antes de ser utilizada em uma expressão ou atribuição.
   - *Erro Semântico:* "Erro: Variável 'x' não declarada."
2. **Unicidade no Escopo:** Não é permitido declarar duas variáveis com o mesmo nome no **mesmo escopo imediato**.
   - *Erro Semântico:* "Erro: Variável 'x' já declarada neste escopo."

### 3.3 Inicialização

- A linguagem permite a declaração de variáveis sem inicialização imediata (ex: 🔢 a;).
- **Comportamento:** Semanticamente é válido. Na geração de código, assume-se que a alocação de memória reserva o espaço, mas o valor é indefinido até a primeira atribuição 🎁.

## 4. Verificação de Tipos (Type Checking)

O analisador semântico percorrerá a Árvore Sintática realizando as seguintes validações:

### 4.1 Expressões

Para cada nó de operação binária (Soma, Subtração, AND, OR, etc.), o compilador verifica:

1. Se o operando da esquerda (Esq) e o da direita (Dir) possuem tipos compatíveis com o operador.
2. O tipo resultante da expressão é propagado para o nó pai.

### 4.2 Comandos de Controle de Fluxo

- **Condicional (🤨 / 🤔) e Repetição (😑 / 😮):**
  - A expressão que define a condição (ex: a 🐓 b) **deve** resultar estritamente no tipo BOOL.
  - *Erro Semântico:* "Erro: A condição do comando IF/WHILE deve ser booleana. Tipo encontrado: INT".

## 5. Especificação da Geração de Código Intermediário

O compilador gerará código linear na forma de **Código de Três Endereços**. Esta representação utiliza variáveis temporárias geradas pelo compilador para desmembrar expressões complexas.

### 5.1 Formato das Instruções

O formato geral é result = arg1 op arg2.

| **Operação E-moji** | **Instrução IR (Exemplo)** | **Significado** |
|---|---|---|
| a 🎁 10 | a = 10 | Atribuição simples |
| a 🎁 b ➕ c | t1 = b + c<br><br>a = t1 | Operação aritmética usando temporário |

| | | |
|---|---|---|
| a 🐓 b | t1 = a > b | Operação relacional |
| 🤨 (cond) | if_false t1 goto L1 | Desvio condicional |
| goto | goto L1 | Desvio incondicional |
| Labels | L1: | Rótulo de destino |

## 5.2 Estratégia de Tradução

### 5.2.1 Atribuições e Expressões

Expressões aninhadas serão "achatadas".

- Fonte:

```
x 🎁 (a ➕ b) ❌ c;
```

- Código Gerado:

```
t1 = a + b
t2 = t1 * c
x = t2
```

### 5.2.2 Controle de Fluxo (IF / ELSE)

Serão gerados rótulos (Label) para controlar os saltos.

- Fonte:

```
🧐 (a 🐓 b) 👎
   cmd1;
👎 🖐️ 👎
   cmd2;
👎
```

- Código Gerado:

```
t1 = a > b
if_false t1 goto L1  # Salta para o Else se falso
cmd1...              # Bloco IF
goto L2              # Pula o Else
L1:                  # Rótulo do Else
cmd2...              # Bloco Else
L2:                  # Fim do IF
```

### 5.2.3 Estruturas de Repetição (WHILE)

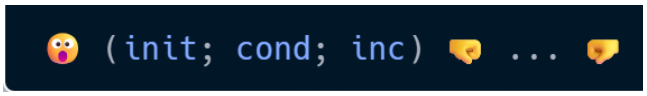
- Fonte:


- Código Gerado:

```
L1:                    # Início do loop
t1 = a < 10
if_false t1 goto L2    # Sai do loop se falso
... (bloco) ...
goto L1                # Volta para testar
L2:                    # Saída
```

### 5.2.4 Estruturas de Repetição (FOR)

O comando:




Será traduzido de forma similar ao While, mas com a inicialização antes e o incremento ao final do bloco.

- Código Gerado:

```
... (código da inicialização) ...
L1:                    # Rótulo Teste
t1 = ... (código da condição) ...
if_false t1 goto L2
... (bloco de código) ...
... (código do incremento) ...
goto L1
L2:                    # Fim
```