Design e Implementação de um Protótipo de Interpretador para uma Linguagem de Programação Orientada ao Entity Component System

Autor Francisco Sebastiany Junior

Orientador Prof. Me. Luciano Santos Cardoso

Projeto

Pivotagem

- Linguagem de Domínio Específico;
- Utilização de Bibliotecas de *Lexing* e *Parsing*;
- Avaliação de Resultados após as Fases de Design e Implementação.

Etapas

1. **Design da Linguagem:** como será a sintaxe, tipagem e tratamento de erros? Qual será o paradigma? Quais *features* do ECS abstrair?
2. **Análise do Design:** quais problemas o design resolve? Quão possível é a implementação dele?
3. **Implementação do Interpretador:** qual será a metodologia de desenvolvimento? Como o ECS será representado internamente? Como será feito o *lexer* e o *parser*?
4. **Análise do Interpretador:** ele pode cumprir com o design proposto? Quais foram as dificuldades enfrentadas?

Fundamentos

Padrão de Design de Software

Definição

- Solução reutilizável para um problema recorrente no design de software;
- É uma descrição geral de como resolver o problema, e não uma solução específica;
- Dividido em padrões *criacionais*, *estruturais* e *comportamentais*;
- Exemplos: *Singleton*, *Decorator* e *Observer*.

Padrão de Arquitetura de Software

Definição

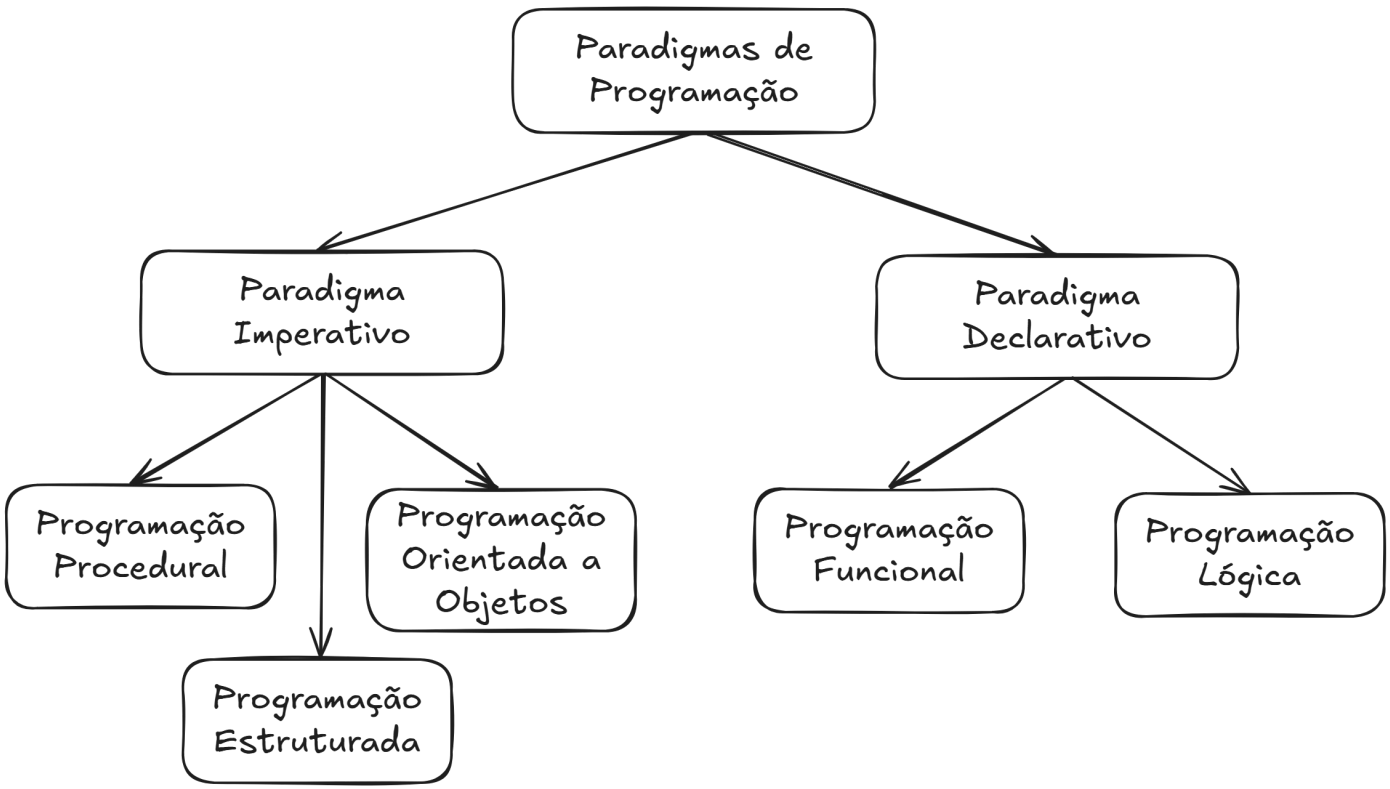
- Muito parecido com um padrão de design, só que mais abrangente, vendo a aplicação como um todo;
- Exemplos: *Model-View-Controller*, *Microservices* e *Entity Component System*.

Paradigma de Programação

Definição

- Conjunto de conceitos e princípios que orientam o desenvolvimento de software;
- Pode ser visto como uma hierarquia, onde há paradigmas mais abrangentes e outros mais específicos, que herdam dos mais abrangentes;
- Os dois paradigmas mais abrangentes são o *imperativo* e *declarativo*;
- Paradigmas mais específicos incluem o *funcional*, *orientado a objetos*, entre outros.

Hierarquia



Fonte: adaptado de ARIO LIYAN (2023).

Entity Component System

Definição

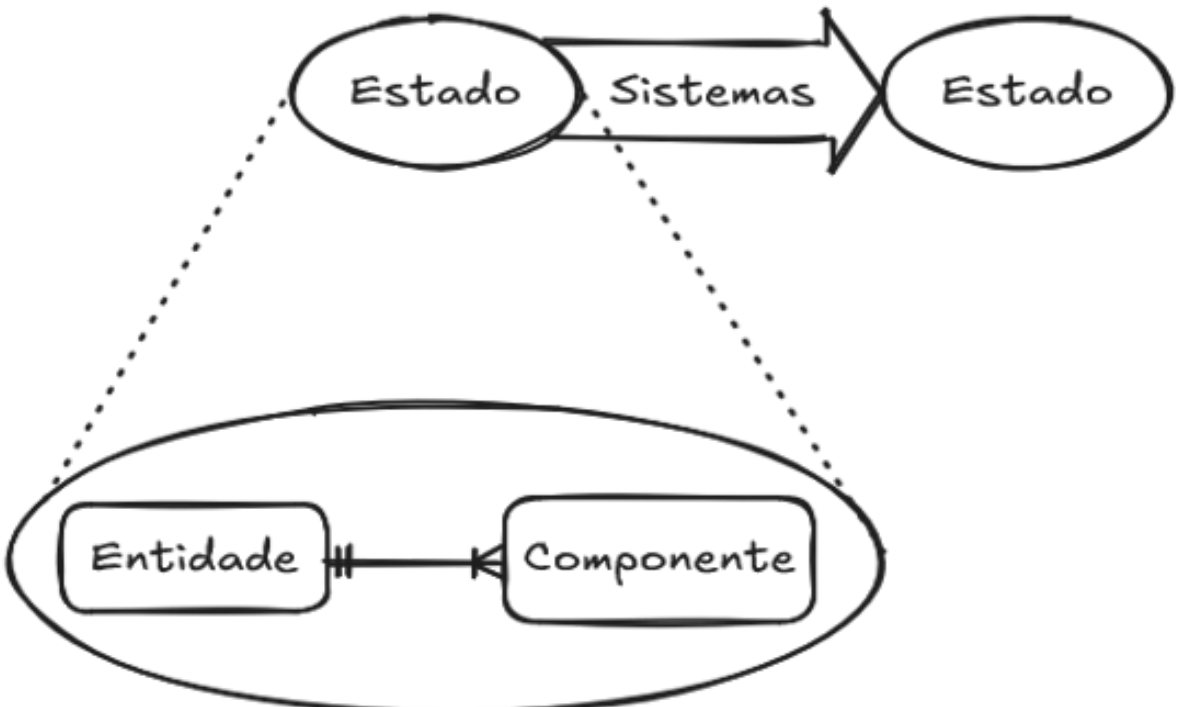
- Padrão de arquitetura baseado no design orientado a dados;
- Surgiu na área de desenvolvimento de jogos devido a sua flexibilidade e desempenho;
- Composto de três elementos fundamentais: *entidades*, *componentes* e *sistemas*;
- Ainda está em fase de formalização.

Entity Component System

Os Três Elementos Fundamentais

- **Entidade:** identificador único sem dado ou lógica;
- **Componente:** estrutura de dados que representa um conceito;
- **Sistema:** lógica que opera em um conjunto de entidades com componentes específicos.

Entity Component System



Fonte: adaptado de SANDER MERTENS (2019).

Entity Component System

```
struct Position {
  x: f32,
  y: f32,
}

struct Velocity {
  dx: f32,
  dy: f32,
}
```

Entity Component System

```rust
fn main() {
    let entities = [0, 1];

    let mut positions = [
        Position { x: 0.0, y: 0.0 }, // Entidade 0.
        Position { x: 1.0, y: 1.0 }, // Entidade 1.
    ];

    let velocities = [
        Velocity { dx: 1.0, dy: 1.0 }, // Entidade 0.
        Velocity { dx: 2.0, dy: 2.0 }, // Entidade 1.
    ];
```

```
    loop { apply_velocity(&entities, &mut positions, &velocities); }
}
```

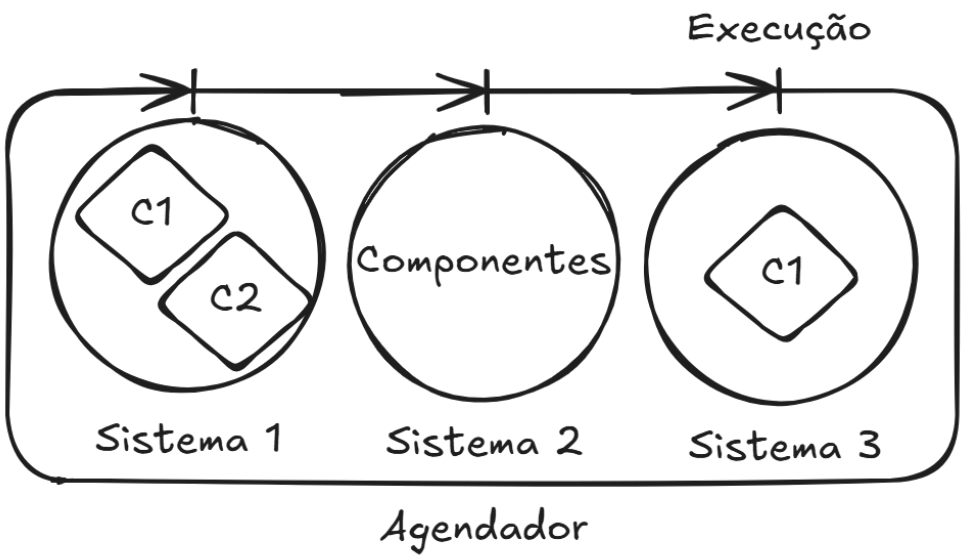
Entity Component System

```
fn apply_velocity(
    entities: &[usize],
    positions: &mut [Position],
    velocities: &[Velocity]
) {
    for &entity in entities {
        positions[entity].x += velocities[entity].dx;
        positions[entity].y += velocities[entity].dy;
    }
}
```

Entity Component System

Agendador

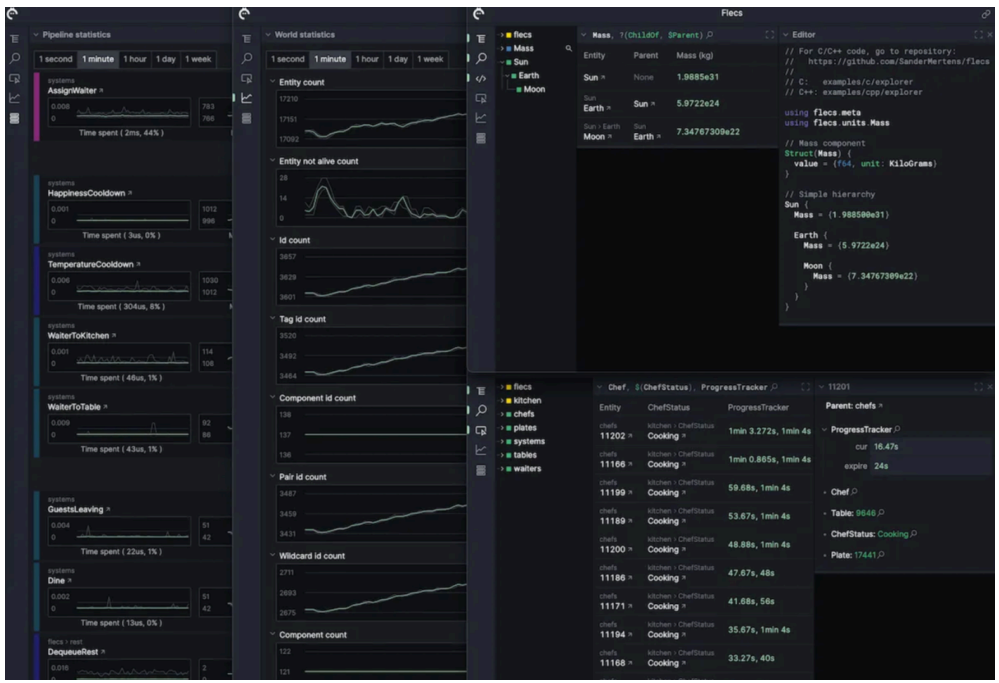
Construto com a finalidade de executar os sistemas da aplicação, também determinando a ordem e a frequência de execução.



Entity Component System

Depurador de ECS

Interface, gráfica ou não, que permite visualizar e manipular as entidades, componentes e sistemas da aplicação.



Fonte: SANDER MERTENS (2021).

Entity Component System

Relacionamentos

Conceito que permite o relacionamento entre entidades e componentes, tornando possível a representação de estruturas mais complexas, como hierarquias.

Um caso de uso é a representação de um sistema de arquivos, onde pastas podem conter arquivos — uma relação pai-filho.

Entity Component System

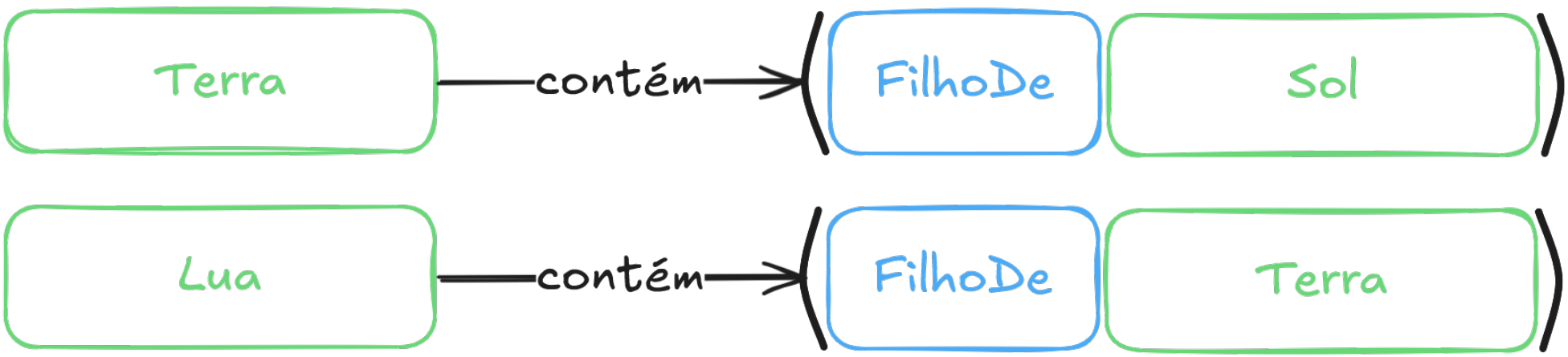
Relacionamentos


```
Entidade.set<Posição>({0, 0});
```

Fonte: elaborado com base em SANDER MERTENS (2022).

Entity Component System

Relacionamentos




```
Terra.add(FilhoDe, Sol); Lua.add(FilhoDe, Terra);
```

Fonte: elaborado com base em SANDER MERTENS (2022).

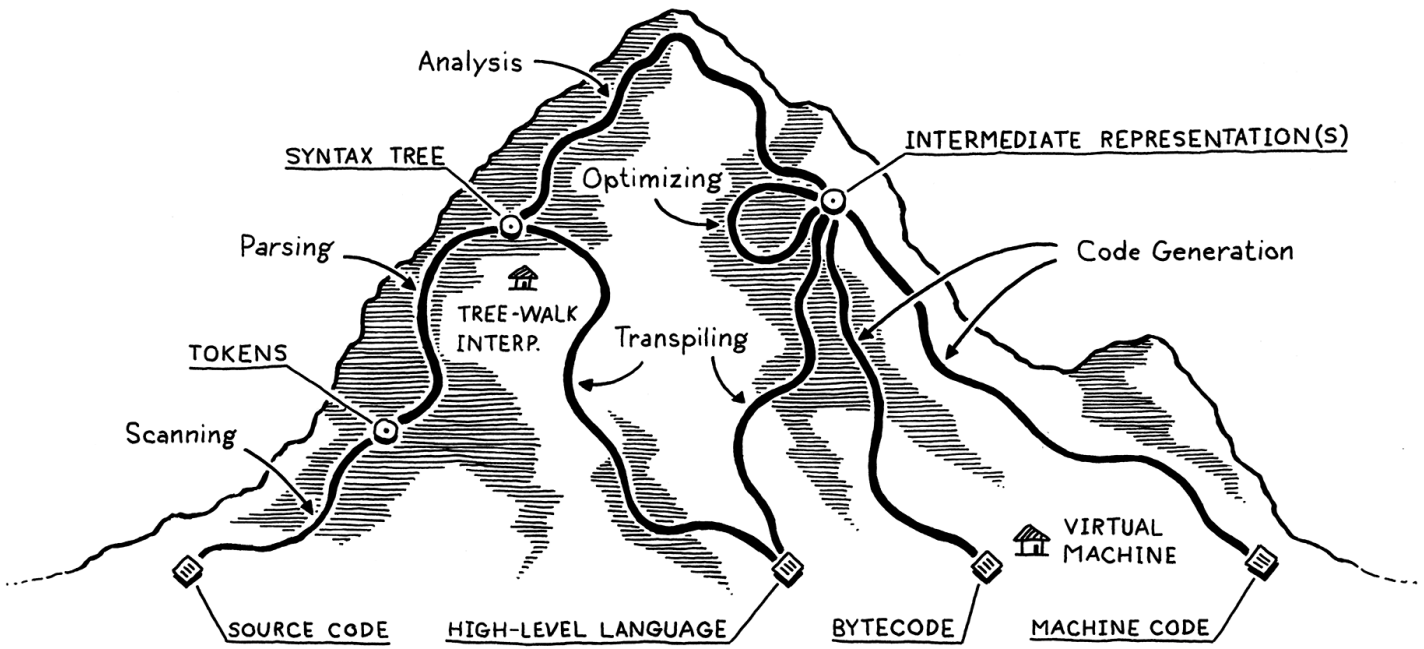
Interpretador *Tree-Walking*

Definição

- Interpretador é um programa que executa diretamente o código fonte de uma linguagem de programação, linha por linha.
- *Tree-Walking* é uma variante de interpretador com foco em simplicidade (a custo de desempenho).
- É separado em três fases: *análise léxica*, *análise sintática* e *interpretação*.

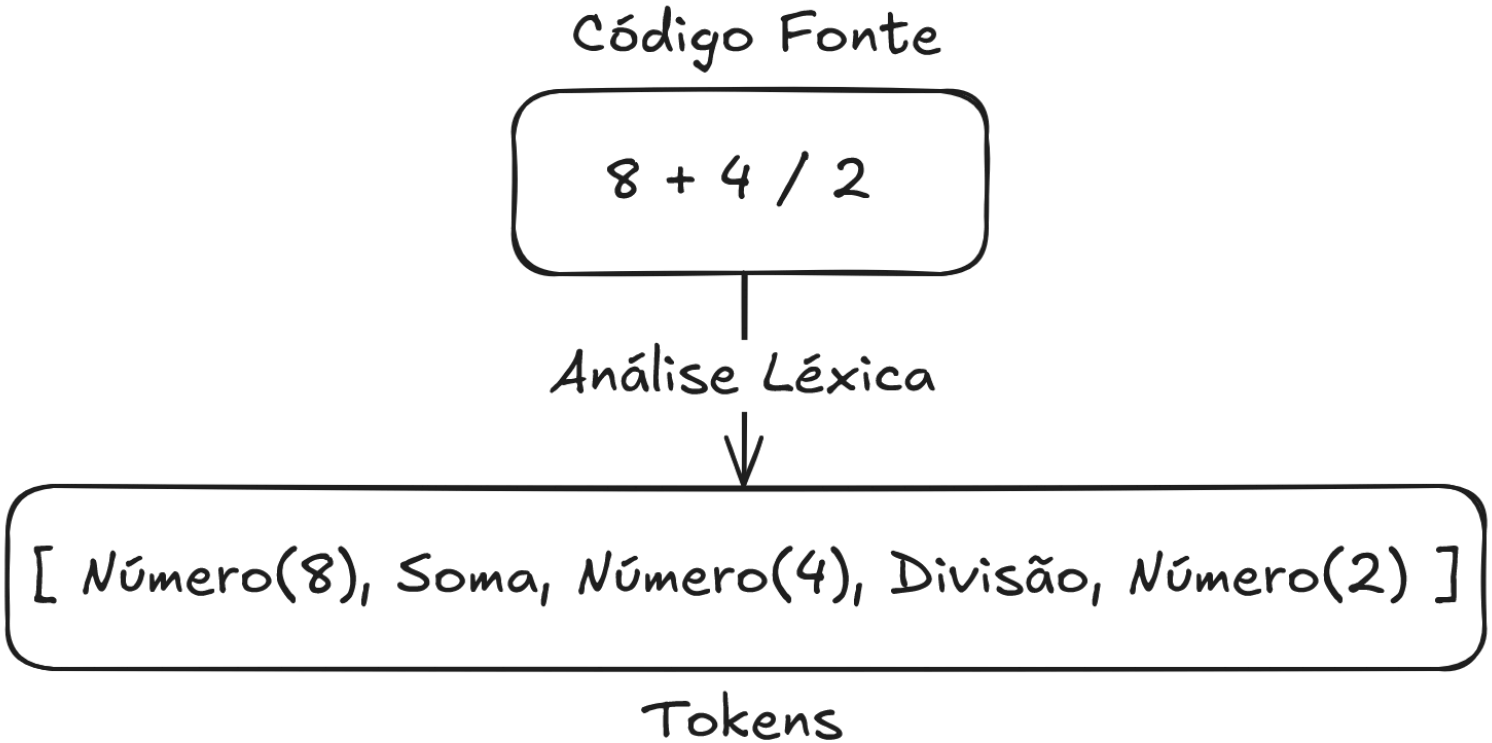
Interpretador *Tree-Walking*

Visão Geral



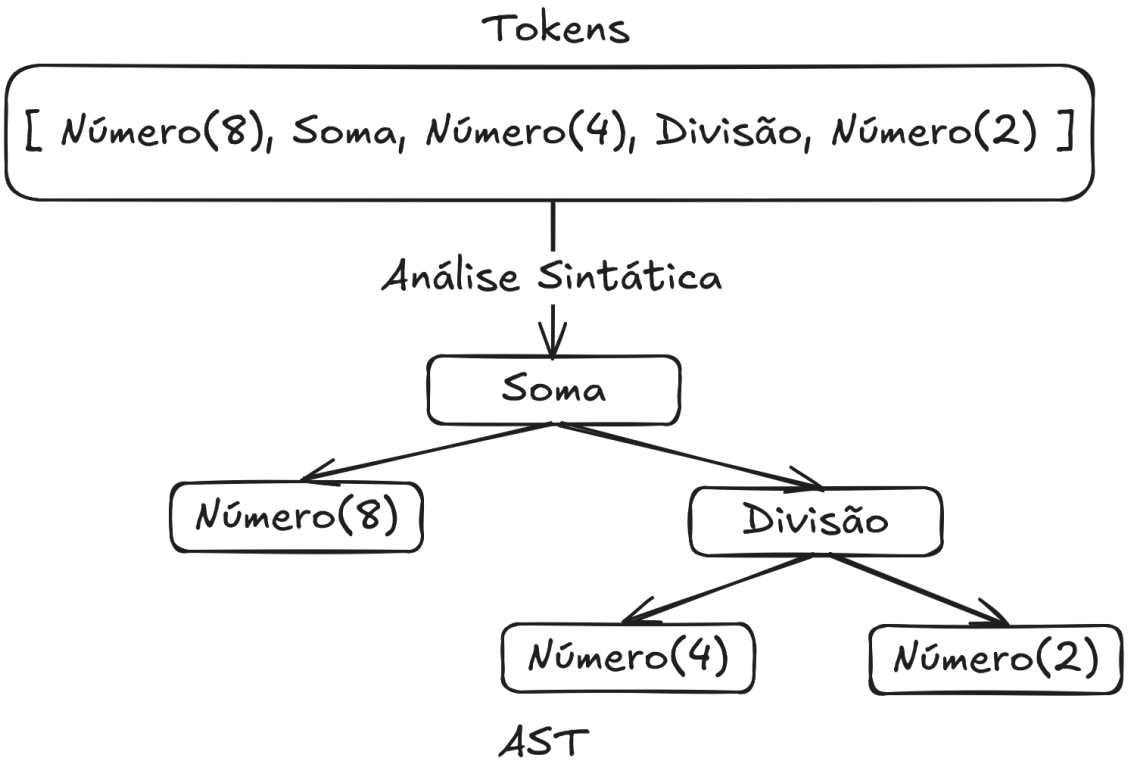
Fonte: ROBERT NYSTROM (2021).

Análise Léxica



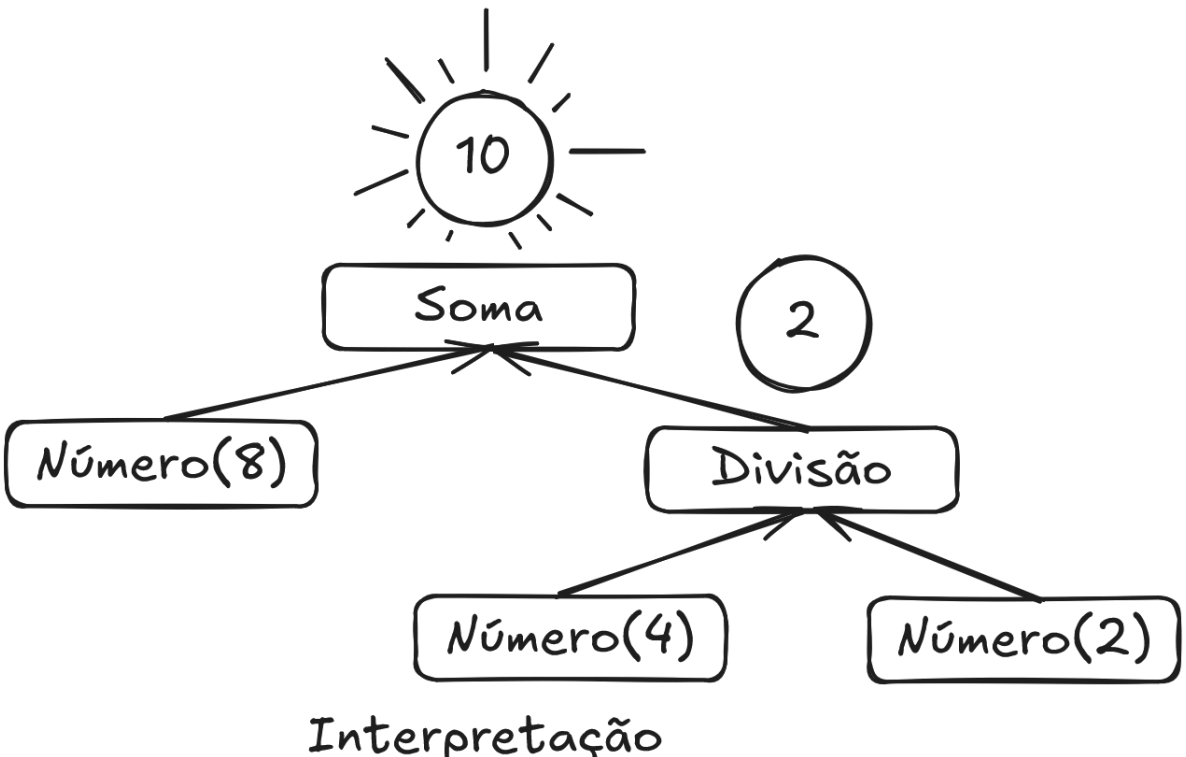
Fonte: baseado em ROBERT NYSTROM (2021).

Análise Sintática



Fonte: baseado em ROBERT NYSTROM (2021).

Interpretação



Fonte: baseado em ROBERT NYSTROM (2021).

Tecnologias

Rust

Motivação

- **Tipagem forte:** gera garantias em tempo de compilação. Uso de enum e match é ótimo na manipulação dos tokens;
- **Tratamento de erros explícito:** toda etapa de um interpretador está sujeita a erros — tratamento explícito garante que os erros sejam tratados de acordo;
- **Alto desempenho:** desempenho anda lado a lado com ECS e interpretadores.

Cargo

- Gerenciador de pacotes da linguagem Rust;
- Será utilizado para gerenciar as bibliotecas *Logos* e *Chumsky*.

Logos

Definição

Biblioteca de análise léxica baseada em *regex* para Rust.

Motivação

A escolha se deve ao fato de que *Logos* é atualmente uma das bibliotecas mais maturas e completas para análise léxica na linguagem Rust, além da alta compatibilidade com *Chumsky*.

Como bônus, ela é também a biblioteca mais rápida segundo o *benchmark* disponível no repositório oficial.

Lexer para uma Linguagem de Aritmética com *Logos*

```
#[derive(Logos)]
#[logos(skip r"[\s]+")]
enum Token {
  #[token("-")]
  Sub,

  #[regex("[0-9]+", |lex| lex.slice().parse().unwrap())]
  Int(i64),
}
```

Fonte: adaptado de MACIEJ HIRSZ (2018).

Chumsky

Definição

Biblioteca de análise sintática baseada no paradigma declarativo para Rust.

Motivação

A escolha se deve aos mesmos motivos que levaram à escolha de *Logos*: maturidade, compatibilidade e desempenho.

AST e *Parser* para uma Linguagem de Aritmética com *Chumsky*

```
enum Expr<'a> {
  Int(i64),
  Neg(Box<Expr<'a>>),
}
```

```
fn parser<'a>() -> impl Parser<'a, &'a str, Expr<'a>> {
  let op = |c| just(c).padded()

  // int -> regex([0-9]+)
  let int = text::int(10)
    .map(|s: &str| Expr::Int(s.parse().unwrap()));

  // unary -> int | '-' unary
  let unary = op('-')
    .repeated()
    .foldr(int, |_op, rhs| Expr::Neg(Box::new(rhs)));

  unary
}
```

Fonte: adaptado de JOSHUA BARRETTO (2021).

Referências

ARIO LIYAN. **What is a Programming Paradigm?**. Disponível em: <https://medium.com/@Ariobarxan/what-is-a-programming-paradigm-ec6c5879952b>. Acesso em: 18 may. 2025.

JOSHUA BARRETTO. **Chumsky**. , 2021. Disponível em: <https://github.com/zesterer/chumsky>. Acesso em: 17 may. 2025

MACIEJ HIRSZ. **Logos**. , 2018. Disponível em: <https://github.com/maciejhirsz/logos>. Acesso em: 17 may. 2025

ROBERT NYSTROM. **Crafting Interpreters**. [s.l.] Genever Benning, 2021.

SANDER MERTENS. **Entity Component System FAQ**. , 2019. Disponível em: <https://github.com/SanderMertens/ecs-faq>. Acesso em: 15 may. 2025

SANDER MERTENS. **Flecs Explorer**. , 2021. Disponível em: <https://github.com/flecs-hub/explorer>. Acesso em: 15 may. 2025

SANDER MERTENS. **Building Games in ECS with Entity Relationships**. Disponível em: <https://ajmmertens.medium.com/building-games-in-ecs-with-entity-relationships-657275ba2c6c>. Acesso em: 19 may. 2025.