

Interpretadores

Construção de compiladores I

Objetivos

Objetivos

- Apresentar o conceito de semântica operacional para especificar interpretadores.
- Mostrar a equivalência entre definições semântica e implementação de interpretadores.

Introdução

Introdução

- Nas aulas anteriores, vimos como construir a árvore de sintaxe abstrata a partir do texto do programa.
 - Análise léxica e sintática

Introdução

- A partir da árvore de sintaxe abstrata, podemos:
 - Fazer análise semântica.
 - Interpretar o código.
 - Gerar código

Introdução

- Antes de lidar com a análise semântica, vamos estudar sobre como construir intepretadores.
 - Motivo: tornar evidente a necessidade da análise semântica.

Noções de semântica

Noções de semântica

- Semântica formal: estudo de formalismos matemáticos para determinar o significado de programas.
- Três abordagens principais: denotacional, axiomática e operacional.

Noções de semântica

- Semântica denotacional.
 - Modelar o significado do programa usando funções e domínios semânticos.
 - Vantagens: composicionalidade
 - Desvantagens: difícil modelar estado.

Noções de semântica

- Semântica axiomática.
 - Significado de um programa é o que pode ser provado sobre ele.
 - Utilizada para demonstrar propriedades de um programa.

Noções de semântica

- Semântica operacional.
 - Semântica de programas expressa por meio de relações.
 - Dois estilos: big-step e small-step

Noções de semântica

- Semântica big-step
 - Definição como relações entre programas e seu resultado.
 - Associa um programa completo e seu respectivo resultado.
- Ideal para especificar interpretadores.

Noções de semântica

- Semântica small-step
 - Definição como relações que mostram a execução passo-a-passo.
- Útil para especificar provas.

Noções de semântica

- Nosso foco no curso será no uso de semântica operacional big-step.
- Porém, vamos mostrar a diferença entre big and small-step, usando uma linguagem simples e sua semântica.

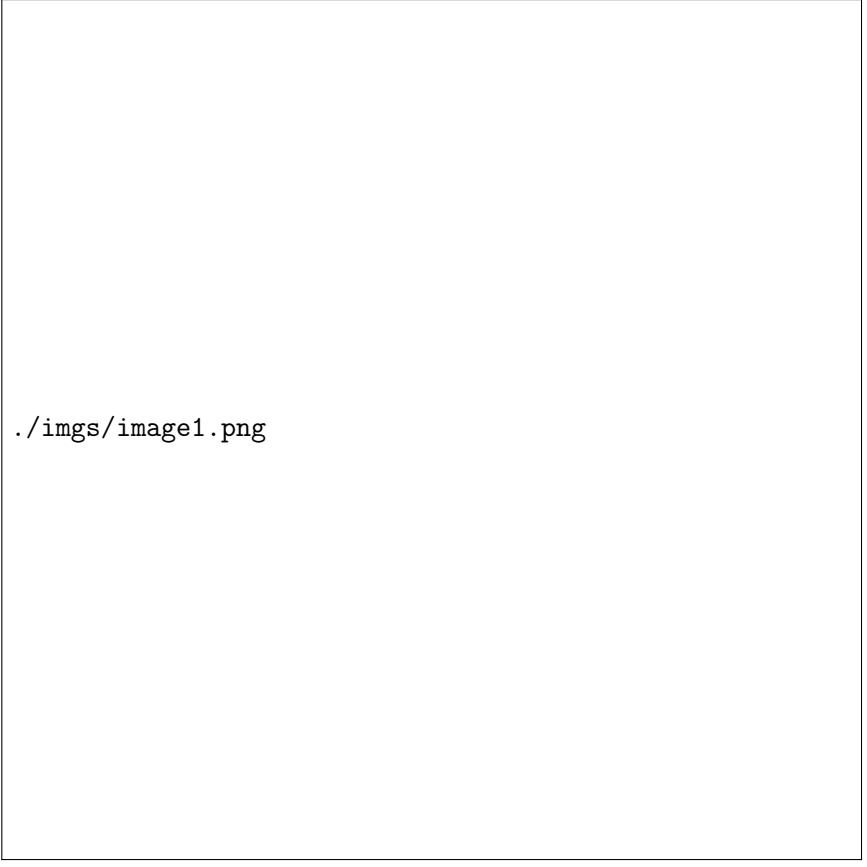
Noções de semântica

- Linguagem considerada:

$$e \rightarrow n \mid e + e \mid e * e$$

Noções de semântica

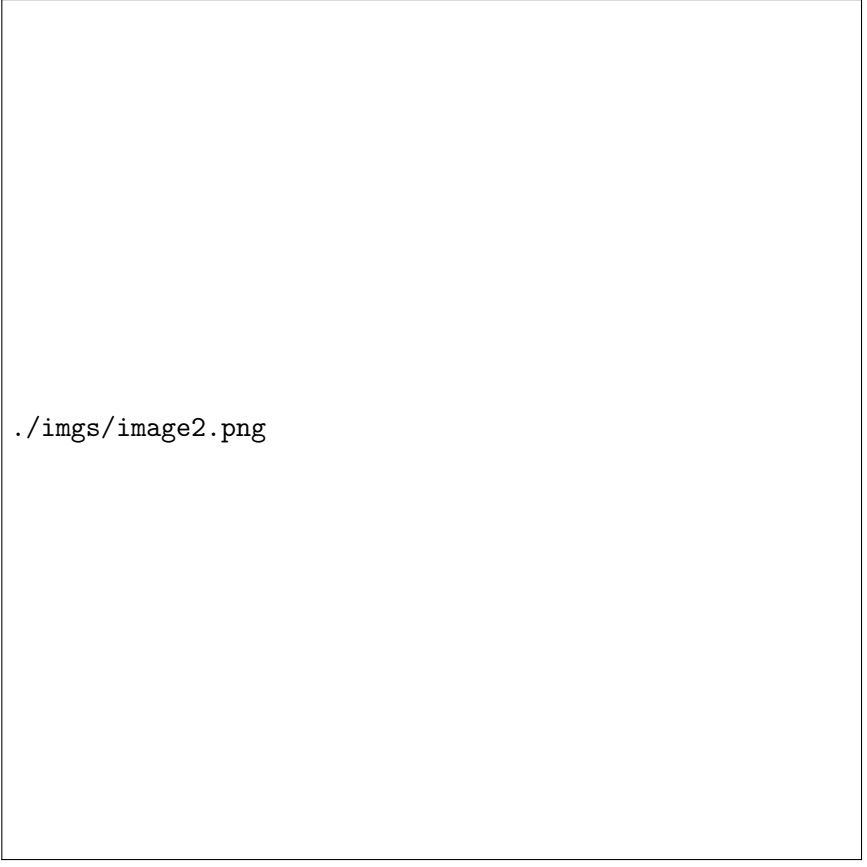
- Semântica big-step



`./imgs/image1.png`

Noções de semântica

- Semântica small-step



`./imgs/image2.png`

Implementação

Implementação

- Sintaxe em Haskell

```
data Exp
  = Const Int
  | Exp :+: Exp
  | Exp *: Exp
  deriving Show

newtype Value = VInt Int deriving Show
```

Implementação

- Semântica big-step

```
bigStep :: Exp -> Value
bigStep (Const n) = VInt n
bigStep (e1 :+: e2)
  = case (bigStep e1, bigStep e2) of
      (VInt n1, VInt n2) -> VInt (n1 + n2)
bigStep (e1 **: e2)
  = case (bigStep e1, bigStep e2) of
      (VInt n1, VInt n2) -> VInt (n1 * n2)
```

Implementação

- Semântica small-step

```
step :: Exp -> [Exp]
step (Const _) = []
step ((Const n1) :+: (Const n2))
  = [Const $ n1 + n2]
step ((Const n1) **: (Const n2))
  = [Const $ n1 * n2]
step (e1 :+: e2)
  = [e1' :+: e2 | e1' <- step e1] ++
    [e1 :+: e2' | e2' <- step e2]
step (e1 **: e2)
  = [e1' **: e2 | e1' <- step e1] ++
    [e1 **: e2' | e2' <- step e2]
```

Implementação

- Semântica small-step

```
data Tree a = Node a [Tree a] deriving Show

multiStep :: Exp -> Tree Exp
multiStep e = Node e [multiStep e' | e' <- step e]
```

Concluindo

Concluindo

- Nesta aula apresentamos uma introdução à construção de interpretadores e semântica formal.
- Próximas aulas: interpretadores para linguagens imperativas.

Exercícios

Exercícios

- Construa um analisador sintático para o código de exemplo para produzir um interpretador completo para a linguagem de expressões.