Semântica das Linguagens de Programação

Maria João Frade

HASLab - INESC TEC
Departamento de Informática, Universidade do Minho

2025/2026

Apresentação da UC

Aspectos de uma Linguagem de Programação

- **Sintaxe:** dedicada à estrutura gramatical dos programas.
  - Descrição dos símbolos e das expressões que constituem a linguagem, e das demais categorias sintácticas (comandos, programas, . . .)
- **Semântica:** dedicada ao significado dos programas gramaticalmente correctos.
  - Descrição do significado das várias construções da linguagem.
  - Permite prever o seu comportamento em tempo de execução.
- **Ferramentas:** parsers, interpretadores, compiladores, debuggers, verificadores, ...

Semântica das Linguagens de Programação

- Esta UC é uma **introdução à semântica formal** das linguagens de programação.
  - Uma semântica formal é a construção de um modelo matemático.
- Uma **descrição rigorosa** do significado dos programas:
  - revela ambiguidades e subtilezas;
  - é independente da máquina;
  - fundamenta compiladores e interpretadores;
  - lança as bases para a verificação formal de programas.
- Apresentaremos **várias abordagens** à semântica, expondo:
  - as ideias fundamentais (tendo por base uma linguagem simples)
  - o seu inter-relacionamento;
  - a sua aplicabilidade.

Benefícios da Semântica Formal

- No desenho da linguagem:
  - É útil na clarificação de aspectos subtis da linguagem.
  - Contribui para descoberta de melhores formas de organizar a informação.
- Na implementação de compiladores ou interpertadores:
  - Permite processar, analisar e optimizar os programas de forma correcta.
  - Lança as bases para a definição de máquinas abstractas e código intermédio.
- Na verificação de programas:
  - Permite raciocinar acerca dos programas, especificações e outras propriedades.
  - Permite definir e provar correção de programas.

Abordagens à Semântica Formal

- **Semântica Operacional**
  - Foco em *como é produzido o efeito* de uma computação.
  - Descreve como um programa executa, passo a passo, definindo, para cada construção da linguagem, regras de transição entre estados.
  - Útil na implementação das linguagens.
- **Semântica Axiomática**
  - Foco nas *propriedades do efeito* de executar um programa, e não em *como*.
  - Descreve o comportamento por meio de propriedades lógicas.
  - Útil na verificação dedutiva dos programas.
- **Semântica Denotacional**
  - Foco apenas *no efeito* da computação, não em *como* ela é obtida.
  - Descreve o significado de um programa num domínio matemático abstracto.
  - Útil para trabalhar a um nível teórico mais abstracto.

Syllabus

- Parte I - **Programação Imperativa**
  - Semânticas operacionais
    - ⋆ Semântica Natural ou Semântica de Avaliação (*big-step*)
    - ⋆ Semântica Estrutural ou Semântica de Trânsições (*small-step*)
  - Semântica axiomática
  - Semântica denotacional
  - Relações entre semânticas
  - Máquinas abstractas
- Parte II - **Programação Funcional**
  - Lambda-calculus (puro e com tipos)
  - Semântica de avaliação "call-by-value"
  - Semântica de avaliação "call-by-name"
  - Sistema de tipos

Bibliografia

- **Semantics with Applications: An Appetizer.**
  H. Nielson and F. Nielson. Springer 2007.
  (disponível online)
- **Theories of Programming Languages.**
  John C. Reynolds. Cambridge Univ. Press, 1998.
  (disponível online)

Avaliação

Dois testes escritos, 50% cada (nota mínima: 5 valores)

- 1º teste: 7 de Abril
- 2º teste: 21 de Maio

Qualquer aluno pode ser chamado a uma prova oral para defender a nota.

Indução

Na abordagem formal à semântica das linguagens de programação, as definições indutivas e as provas por indução estão por todo o lado. . .

Indução

- A indução é uma técnica que nos permite definir e raciocinar com objectos que são:
  - *estruturados*, de uma forma bem fundada;
  - *finitos*, embora arbitrariamente grandes.
- A indução explora a natureza finita e estruturada desses objectos para superar a sua complexidade arbitrária.
- Os objectos compostos tem uma forma única de ser decompostos nos seus constituintes imediatos.

Indução matemática

Seja Φ uma qualquer propriedade sobre números naturais, $\mathbb{N}$.

Para provar

$$\forall x \in \mathbb{N}.\, \Phi(x)$$

basta provar:

- (caso de base) $\quad \Phi(0)$
- (passo indutivo) $\quad \forall x \in \mathbb{N}.\, \Phi(x) \Rightarrow \Phi(x+1)$

Boa fundação

Relação bem fundada

Uma relação binária $\prec$ sobre um conjunto A diz-se *bem fundada*, se não existir nenhuma sequência infinita decrescente ... $\prec a_3 \prec a_2 \prec a_1$

A seguinte relação definida sobre um conjunto definido indutivamente é bem fundada

$$t' \prec t \quad \text{sse} \quad t' \text{ é um sub-termo estrito de } t$$

Indução bem fundada

Seja Φ uma propriedade sobre elementos de um conjunto A definido indutivamente. Para provar $\forall a \in A.\, \Phi(a)$
basta assumir $\Phi(a')$ para todo o $a' \prec a$ e, com isso, provar que $\Phi(a)$ se verifica.

Indução estrutural

Prova de uma propriedade por **indução estrutural** numa determinada categoria sintática.

- **Casos de base**
  Provar que a propriedade se verifica para todos os elementos de base da categoria sintática.
- **Casos indutivos**
  Para cada elemento composto da categoria sintática, assumir que a propriedade se verifica para todos os constituintes imediatos do elemento (estas são as *hipóteses de indução*) e provar que esta também se verifica para o elemento composto.

Exemplo

Considere a representação de números no sistema binário dada pela categoria sintática $\mathbf{BNum}$ de acordo com a seguinte sintaxe abstracta:

$$\mathbf{BNum} \ni n ::= 0 \mid 1 \mid n\,0 \mid n\,1$$

Para determinar o número inteiro representado pelo numeral definimos uma função semântica $\mathcal{N} : \mathbf{BNum} \to \mathbf{Z}$ definida indutivamente da seguinte forma

$$\begin{array}{rcl} \mathcal{N}[\![0]\!] & = & 0 \\ \mathcal{N}[\![1]\!] & = & 1 \\ \mathcal{N}[\![n\,0]\!] & = & 2 * \mathcal{N}[\![n]\!] \\ \mathcal{N}[\![n\,1]\!] & = & 2 * \mathcal{N}[\![n]\!] + 1 \end{array}$$

Exemplo (prova por indução estrutural)

Provar que $\mathcal{N} : \mathbf{BNum} \to \mathbf{Z}$ é uma função total.

$\mathcal{N}$ é uma função total se para todo o argumento $n \in \mathbf{BNum}$ existir um único número $\mathrm{n} \in \mathbf{Z}$ tal que $\mathcal{N}[\![n]\!] = \mathrm{n}$.

Prova por indução na estrutura de n.

- **Casos de base**
  Provar a propriedade quando n é 1 e quando n é 0.
- **Casos indutivos**
  Provar a propriedade quando n é da forma $n'\,0$ e $n'\,1$.

A linguagem **While**

A linguagem **While**

Categorias sintáticas / Sintaxe abstracta

$$
\begin{array}{rcl}
\mathbf{Num} \ni n & & \text{(numerais em notação decimal)} \\
\mathbf{Var} \ni x & & \text{(variáveis)} \\
\mathbf{Aexp} \ni a & ::= & n \mid x \mid a_1 + a_2 \mid a_1 * a_2 \mid a_1 - a_2 \\
\mathbf{Bexp} \ni b & ::= & \texttt{false} \mid \texttt{true} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\
\mathbf{Stm} \ni C & ::= & x := a \mid \texttt{skip} \mid C_1 \,;\, C_2 \mid \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2 \\
& \mid & \texttt{while}\ b\ \texttt{do}\ C
\end{array}
$$

Semântica das expressões

- O número de operadores das expressões (aritméticas e booleanas) foi, sem perda de expressividade, reduzido a um mínimo, para simplificar as provas.
- Os operadores habituais podem ser vistos como macros definidas às custa dos operadores nativos.

$$
\begin{array}{rcl}
a_1 > a_2 & = & \neg(a_1 \leq a_2) \\
a_1 \geq a_2 & = & a_1 > a_2 \vee a_1 = a_2 \\
a_1 < a_2 & = & a_1 \leq a_2 \wedge \neg(a_1 = a_2) \\
b_1 \vee b_2 & = & \neg(\neg b_1 \wedge \neg b_2)
\end{array}
$$

Semântica das expressões

- Os numerais em notação decimal têm o significado esperado. Por exemplo, $\mathcal{N}[\![23]\!] = 23 \in \mathbf{Z}$.

  Note que um numeral é uma entidade sintática e um número é um valor semântico.
- O significado de uma expressão depende do valor atribuído às variáveis que nela ocorrem.

  Por exemplo, a expressão $x + 1$ depende do valor do x.

  Surge assim a necessidade de introduzir a noção de **estado**.

Estado

- O **estado** associa a cada variável o seu valor corrente.
- O estado será representado por uma função de variáveis para valores do domínio de interpretação (neste caso os inteiros).

$$\mathbf{State} = \mathbf{Var} \to \mathbf{Z}$$

Seja s um estado.

- $s\,x$ representa o valor da variável x no estado s.
- $s[y \mapsto v]$ representa a seguinte função:

$$(s[y \mapsto v])\,x = \begin{cases} v & \text{se } x = y \\ s\,x & \text{se } x \neq y \end{cases}$$

Semântica das expressões aritméticas

A interpretação das expressões aritméticas é feita pela seguinte função semântica definida indutivamente.

$$\mathcal{A} : \mathbf{Aexp} \to (\mathbf{State} \to \mathbf{Z})$$

$$\begin{array}{rcl}
\mathcal{A}[\![n]\!]\,s & = & \mathcal{N}[\![n]\!] \\
\mathcal{A}[\![x]\!]\,s & = & s\,x \\
\mathcal{A}[\![a_1 + a_2]\!]\,s & = & \mathcal{A}[\![a_1]\!]\,s + \mathcal{A}[\![a_2]\!]\,s \\
\mathcal{A}[\![a_1 * a_2]\!]\,s & = & \mathcal{A}[\![a_1]\!]\,s * \mathcal{A}[\![a_2]\!]\,s \\
\mathcal{A}[\![a_1 - a_2]\!]\,s & = & \mathcal{A}[\![a_1]\!]\,s - \mathcal{A}[\![a_2]\!]\,s
\end{array}$$

Semântica das expressões booleanas

A interpretação das expressões booleanas é feita, no conjunto dos valores lógicos $\mathbf{T} = \{\mathbf{tt}, \mathbf{ff}\}$, pela seguinte função semântica.

$$\mathcal{B} : \mathbf{Bexp} \to (\mathbf{State} \to \mathbf{T})$$

$$\begin{array}{rcl}
\mathcal{B}[\![\texttt{true}]\!]\,s & = & \mathbf{tt} \\
\mathcal{B}[\![\texttt{false}]\!]\,s & = & \mathbf{ff} \\
\mathcal{B}[\![a_1 = a_2]\!]\,s & = & \begin{cases} \mathbf{tt} & \text{se } \mathcal{A}[\![a_1]\!]\,s = \mathcal{A}[\![a_2]\!]\,s \\ \mathbf{ff} & \text{se } \mathcal{A}[\![a_1]\!]\,s \neq \mathcal{A}[\![a_2]\!]\,s \end{cases} \\
\mathcal{B}[\![a_1 \leq a_2]\!]\,s & = & \begin{cases} \mathbf{tt} & \text{se } \mathcal{A}[\![a_1]\!]\,s \leq \mathcal{A}[\![a_2]\!]\,s \\ \mathbf{ff} & \text{se } \mathcal{A}[\![a_1]\!]\,s > \mathcal{A}[\![a_2]\!]\,s \end{cases} \\
\mathcal{B}[\![\neg b]\!]\,s & = & \begin{cases} \mathbf{tt} & \text{se } \mathcal{B}[\![b]\!]\,s = \mathbf{ff} \\ \mathbf{ff} & \text{se } \mathcal{B}[\![b]\!]\,s = \mathbf{tt} \end{cases} \\
\mathcal{B}[\![b_1 \wedge b_2]\!]\,s & = & \begin{cases} \mathbf{tt} & \text{se } \mathcal{B}[\![b_1]\!]\,s = \mathbf{tt} \text{ e } \mathcal{B}[\![b_2]\!]\,s = \mathbf{tt} \\ \mathbf{ff} & \text{se } \mathcal{B}[\![b_1]\!]\,s = \mathbf{ff} \text{ ou } \mathcal{B}[\![b_2]\!]\,s = \mathbf{ff} \end{cases}
\end{array}$$

Variáveis livres

$\mathsf{FV}(a)$ denota o conjunto das *variáveis livres* da expressão aritmética a.

Variáveis livres de a

$$\begin{array}{rcl}
\mathsf{FV}(n) & = & \emptyset \\
\mathsf{FV}(x) & = & \{x\} \\
\mathsf{FV}(a_1 + a_2) & = & \mathsf{FV}(a_1) \cup \mathsf{FV}(a_2) \\
\mathsf{FV}(a_1 * a_2) & = & \mathsf{FV}(a_1) \cup \mathsf{FV}(a_2) \\
\mathsf{FV}(a_1 - a_2) & = & \mathsf{FV}(a_1) \cup \mathsf{FV}(a_2)
\end{array}$$

Exercício

Defina o conjunto $\mathsf{FV}(b)$ das variáveis livres da expressões booleana b.

Substituições

$a[y \mapsto a_0]$ denota a *substituição*, na expressão a, de cada ocorrência livre da variável y pela expressão a_0.

Substituição sobre uma expressão aritmética

$$
\begin{array}{rcl}
n[y \mapsto a_0] & = & n \\
x[y \mapsto a_0] & = & \begin{cases} a_0 & \text{se } x = y \\ x & \text{se } x \neq y \end{cases} \\
(a_1 + a_2)[y \mapsto a_0] & = & (a_1[y \mapsto a_0]) + (a_2[y \mapsto a_0]) \\
(a_1 * a_2)[y \mapsto a_0] & = & (a_1[y \mapsto a_0]) * (a_2[y \mapsto a_0]) \\
(a_1 - a_2)[y \mapsto a_0] & = & (a_1[y \mapsto a_0]) - (a_2[y \mapsto a_0])
\end{array}
$$

Exercício

Defina a função de substituição sobre expressões booleanas.

Propriedades

Lema

Sejam s e s' estados tais que $s\,x = s'\,x$ para todo o $x \in \mathsf{FV}(a)$. Então

$$\mathcal{A}[\![a]\!]\,s = \mathcal{A}[\![a]\!]\,s'$$

Prova: por indução na estrutura de a.

Lema

Para qualquer estado s,

$$\mathcal{A}[\![a[y \mapsto a_0]]\!]\,s = \mathcal{A}[\![a]\!]\,(s[y \mapsto \mathcal{A}[\![a_0]\!]\,s])$$

Prova: por indução na estrutura de a.

Exercício

Defina (e prove) resultados similares para expressões booleanas.

Semântica da linguagem **While**

- A semântica que foi definida para as expressões aritméticas e booleanas apenas consultam o estado, não o alteram.
- O papel de um comando da linguagem é alterar o estado. A semântica dos comandos poderá portanto modificar o estado.
- Para interpretar os comandos iremos usar um estilo operacional. A semântica operacional descreve como os programas são executados e não apenas os resultados da sua execução.
- A semântica dos comandos vai ser dada por um sistema de transição.
- Existem duas abordagens à semântica operacional (que se distinguem na forma como a relação de transição é especificada).

Semântica Operacional

Semântica operacional

Existem duas abordagens à semântica operacional:

- **Semântica Natural** (ou **Big-step**)
  - O seu propósito é descrever como os *resultados globais* das execuções são obtidos.
  - Também é chamada de Semântica de Avaliação.
- **Semântica Operacional Estrutural** (ou **Small-step**)
  - O seu propósito é descrever como os *passos individuais* das computações são executados.
  - Também é chamada de Semântica de Transições.

Sistemas de transição

Na semântica operacional o significado de um comando é dado por um sistema de transição.

Sistema de transição

Um sistema de transição é constituído por um conjunto de *configurações* e por uma *relação de transição* entre configurações. Há dois tipos de configurações:

- $\langle C, s\rangle$ – indica que o comando C vai ser executado no estado s.
- s – representa um estado final (é uma *configuração terminal*)

- A relação de transição descreve como a execução é feita.
- A diferença entre a abordagem *big-step* e *small-step* está na forma como a relação de transição é especificada.

Semântica Natural

(big-step)

Semântica natural

- A *relação de transição* tem a forma $\langle C, s\rangle \to s'$ e especifica, para cada comando, a relação entre o estado inicial e o estado final.
- A relação de transição é definida indutivamente por um conjunto de *regras*.
- As regras sem premissas chamam-se *axiomas*.

Semântica natural para **While**

$$[\mathrm{skip_{ns}}] \qquad \frac{}{\langle \texttt{skip}, s\rangle \to s}$$

$$[\mathrm{ass_{ns}}] \qquad \frac{}{\langle x := a, s\rangle \to s\,[x \mapsto \mathcal{A}[\![a]\!]\, s]}$$

$$[\mathrm{comp_{ns}}] \qquad \frac{\langle C_1, s\rangle \to s' \qquad \langle C_2, s'\rangle \to s''}{\langle C_1\,;\,C_2, s\rangle \to s''}$$

Semântica natural para **While**

$$[\mathrm{if^{tt}_{ns}}] \qquad \frac{\langle C_1, s\rangle \to s'}{\langle \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2, s\rangle \to s'} \quad \text{se} \quad \mathcal{B}[\![b]\!]\, s = \mathbf{tt}$$

$$[\mathrm{if^{ff}_{ns}}] \qquad \frac{\langle C_2, s\rangle \to s'}{\langle \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2, s\rangle \to s'} \quad \text{se} \quad \mathcal{B}[\![b]\!]\, s = \mathbf{ff}$$

$$[\mathrm{while^{tt}_{ns}}] \qquad \frac{\langle C, s\rangle \to s' \qquad \langle \texttt{while}\ b\ \texttt{do}\ C, s'\rangle \to s''}{\langle \texttt{while}\ b\ \texttt{do}\ C, s\rangle \to s''} \quad \text{se} \quad \mathcal{B}[\![b]\!]\, s = \mathbf{tt}$$

$$[\mathrm{while^{ff}_{ns}}] \qquad \frac{}{\langle \texttt{while}\ b\ \texttt{do}\ C, s\rangle \to s} \quad \text{se} \quad \mathcal{B}[\![b]\!]\, s = \mathbf{ff}$$

Árvores de derivação

- As transições são derivadas construindo *árvores de derivação*.
- A *raiz* da derivação é a transição que queremos derivar e as *folhas* são instâncias dos axiomas.
- Os *nodos internos* são conclusões de instâncias de regras.
- As condições instanciadas das regras têm que ser satisfeitas.

Terminação

A execução de um programa C num estado s

- termina se e só se existir um estado s' tal que $\langle C, s\rangle \to s'$
- diverge (ou entra em ciclo) se e só se não existir nenhum estado s' tal que $\langle C, s\rangle \to s'$

Exercício

Como é que se comportam os seguintes programas?

1. $\texttt{while}\ \neg(x = 1)\ \texttt{do}\ \{y := y * x;\ x := x-1\}$
2. $\texttt{while}\ 1 \leq x\ \texttt{do}\ \{y := y * x;\ x := x-1\}$
3. `while true do skip`

Equivalência semântica

O sistema de transição permite-nos argumentar acerca dos programas e das suas propriedades.

Equivalência semântica

Dois programas, C_1 e C_2, são *semanticamente equivalentes* se, para quaisquer estados s e s',

$$\langle C_1, s\rangle \to s' \qquad \text{sse} \qquad \langle C_2, s\rangle \to s'$$

Exercício

Prove que os programas seguintes são semanticamente equivalentes.

1. `while` b `do` C
2. `if` b `then` $\{C;$ `while` b `do` $C\}$ `else skip`

Indução na estrutura da derivação

Para provar uma propriedade para todas as árvores de derivação:

- **Casos de base** (os axiomas)

  Provar que a propriedade se verifica para todos os axiomas.

- **Casos indutivos** (as regras)

  Para cada regra assumir que a propriedade se verifica para as premissas da regra (são as hipóteses de indução) e provar que a propriedade também se verifica para a conclusão, desde que as condições da regra sejam satisfeitas.

Determinismo

A semântica natural aqui apresentada é *determinista*.

Teorema

Para quaisquer estados s, s', s'' e programa C,

$$\text{se} \ \ \langle C, s\rangle \to s' \ \text{ e } \ \langle C, s\rangle \to s'' \ \text{ então } \ s' = s''.$$

Prova: por indução na estrutura da derivação de $\langle C, s\rangle \to s'$.

(Assume-se que $\langle C, s\rangle \to s'$ e demonstra-se que, se $\langle C, s\rangle \to s''$, então $s' = s''$.)

A função semântica $\mathcal{S}_{\text{ns}}$

O significado de um programa pode ser visto como uma função parcial de $\mathbf{State}$ para $\mathbf{State}$.

Função semântica $\mathcal{S}_{\text{ns}}$

$$\mathcal{S}_{\text{ns}} : \mathbf{Stm} \to (\mathbf{State} \hookrightarrow \mathbf{State})$$

$$\mathcal{S}_{\text{ns}}[\![C]\!]\, s = \begin{cases} s' & \text{se } \ \langle C, s\rangle \to s' \\ \underline{\text{undef}} & \text{caso contrário} \end{cases}$$

Para qualquer programa C, $\mathcal{S}_{\text{ns}}[\![C]\!] \in \mathbf{State} \hookrightarrow \mathbf{State}$ é uma função parcial.

Exercício

Qual é resultado de $\mathcal{S}_{\text{ns}}[\![\texttt{while true do skip}]\!]$?