Semântica das Linguagens de Programação

Maria João Frade
mjf@di.uminho.pt

Bibliografia

- *Semantics with Applications: A formal introduction.*
  H. Nielson & F. Nielson. Wiley, 1992. [disponível online]
  (re-editado pela Springer em 2007)
- *Theories of Programming Languages.*
  J.C. Reynolds. Cambridge Univ. Press, 1998.
- *The Semantics of Programming Languages.*
  M. Hennessy. Wiley, 1990. [disponível online]
- *The Formal Semantics of Programming Languages.*
  G. Winskel. MIT Press, 1993.

Definição de uma Linguagem de Programação

- Definição da **sintaxe** da linguagem
  - focada estrutura gramatical do programa
  - definição das diversas categorias sintácticas (expressões, comandos, programas, …)
- Definição da **semântica** da linguagem
  - descrição do *significado* das várias construções da linguagem
  - permite prever o seu comportamento em tempo de *execução*
  - *Qual o significado de um programa? Como é que ele é executado? Que garantias podemos ter a sua execução?*
- **Ferramentas:** *parsers*, interpretadores, compiladores, *debuggers*, *profilers*…

Semântica Informal

Qual o significado da seguinte construção?

$$\textbf{while } B \textbf{ do } C$$

Segundo Kernighan & Ritchie, em *The C Programming Language*,

"O comando C é executado repetidamente enquanto o valor da expressão B for verdadeiro, sendo o teste feito antes da execução do comando."

Semântica Informal

- As diversas características das linguagens de programação são muitas vezes apenas descritas de modo informal.
- Mas é importante ter uma descrição *precisa* do significado das várias construções de uma linguagem de programação.
  - *Quando é que dois programas são equivalentes?*
  - *Quando é que um programa satisfaz a sua especificação?*
  - *O compilador da linguagem é correcto?*
- Precisamos de uma **semântica formal** que especifique rigorosamente o comportamento dos programas.

Semântica Formal

- É a construção de um modelo matemático.
- Preocupa-se em especificar rigorosamente o comportamento dos programas.
- Especificação do comportamento é *independente da máquina.*
- Lança as bases para a implementação, análise e verificação dos programas.
- Permite revelar vários tipos de subtilizas de que é importante estar ciente.

Benefícios da Semântica Formal

- **No desenho da linguagem:** pode ser útil na clarificação de aspectos subtis e à descoberta de melhores formas de organizar a informação.
- **Na implementação:** permite processar, analisar e optimizar os programas de forma correcta. Lança as bases para a definição de máquinas abstractas e código intermédio.
- **Na verificação:** permite raciocinar acerca dos programas, especificações e outras propriedades.

Abordagens à Semântica Formal

- **Operacional:** descreve os passos de execução de um programa como uma máquina abstracta. Foca-se em *como* o efeito de uma computação é produzido.
- **Denotacional:** descreve o significado de um programa num domínio matemático abstracto. Foca-se apenas *no* efeito da computação não em como ele é obtido.
- **Axiomática:** descreve o comportamento de programas através de uma lógica. As propriedades do efeito de executar um programa são expressas por *asserções*.

Cada estilo tem os seus usos…

- **Operacional:** útil na implementação das linguagens.
- **Axiomática:** útil na verificação dedutiva dos programas.
- **Denotacional:** útil para trabalhar a um nível mais abstracto.

Na realidade os vários estilos usam-se uns aos outros.

Por exemplo:

- A prova de correção de um sistema de inferência axiomático é feita face à semântica operacional ou denotacional.

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.
- O objectos compostos tem uma forma única de ser decompostos nos seus constituintes imediatos.

Na abordagem formal à semântica as definições indutivas e as provas por indução estão por todo o lado…

Indução Matemática

Para qualquer propriedade $\Phi(x)$ sobre números naturais $x \in \mathbb{N} \stackrel{\text{def}}{=} \{0,1,2,\ldots\}$

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

basta provar

$\Phi(0)$ (caso base)

e $\forall x \in \mathbb{N}.\ \Phi(x) \Rightarrow \Phi(x+1)$ (passo indutivo)

Boa fundação

Uma relação binária $\prec$ sobre uma conjunto A diz-se uma **relação bem fundada** se não existir nenhuma sequência infinita decrescente

$$\ldots \prec a_3 \prec a_2 \prec a_1$$

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

$t' \prec t$ sse t' é um sub-termo estrito de t

Indução bem fundada

Para qualquer propriedade $P(a)$ sobre elementos de A

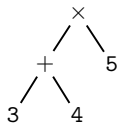
para provar $\forall a \in A\,.\,P(a)$

basta assumir que $P(b)$ para todo o elemento $b \in A$ tal que $b \prec a$

e com isso provar que $P(a)$ se verifica.

Sintaxe Abstracta

- A semântica de uma linguagem de programação só se preocupa com a sintaxe abstracta da linguagem.
- De facto estamos a lidar com árvores de sintaxe abstracta.
- Por exemplo:



em vez de $3 + 4 \times 5$

embora usemos uma sintaxe linear como $((3 + 4) \times 5)$

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.

Um pequeno exemplo

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

$$n ::= 0 \mid 1 \mid n\ 0 \mid n\ 1$$

usamos $n \in \mathbf{Num}$ como meta-variável.

Para determinar o número inteiro representado pelo numeral definimos uma função semântica

$$\mathcal{N}\colon \mathbf{Num} \to \mathbf{Z}$$

definida indutivamente da seguinte forma

$$\begin{aligned}
\mathcal{N}[\![0]\!] &= \mathbf{0} \\
\mathcal{N}[\![1]\!] &= \mathbf{1} \\
\mathcal{N}[\![n\ 0]\!] &= \mathbf{2} \star \mathcal{N}[\![n]\!] \\
\mathcal{N}[\![n\ 1]\!] &= \mathbf{2} \star \mathcal{N}[\![n]\!] + \mathbf{1}
\end{aligned}$$

Uma prova por indução estrutural

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

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

Prova: Por indução na estrutura de n

- **Casos de base**

  Provar a propriedade quando n é 0 ou 1.

- **Casos indutivos**

  Provar a propriedade quando n é $n'0$ ou $n'1$.

A linguagem **While**

Categorias sintáticas e respectivas *meta-variáveis*.

n will range over numerals, $\mathbf{Num}$, (em notação decimal)
x will range over variables, $\mathbf{Var}$,
a will range over arithmetic expressions, $\mathbf{Aexp}$,
b will range over boolean expressions, $\mathbf{Bexp}$, and
S will range over statements, $\mathbf{Stm}$. (os comandos)

A linguagem **While**

Sintaxe abstracta

$$
\begin{array}{lcl}
a & ::= & n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2 \\
b & ::= & \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\
S & ::= & x := a \mid \texttt{skip} \mid S_1 \,;\, S_2 \mid \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \\
 & \mid & \texttt{while } b \texttt{ do } S
\end{array}
$$

Semântica das expressões

Os numerais em notação decimal têm o significado esperado. Por exemplo,

$$\mathcal{N}[\![137]\!] = \mathbf{137} \in \mathbf{Z}$$

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**

- terá o valor 4 se o valor de **x** for 3
- terá o valor 3 se o valor de **x** for 2

Surge 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).

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

Seja s um estado.

$s\ x$ representa o valor da variável x no estado s.

$s[y \mapsto v]$ representa a seguinte função:

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

Semântica das expressões aritméticas

É dada pela seguinte função semântica

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

definida indutivamente do seguinte modo

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

Semântica das expressões booleanas

É dada pela seguinte função semântica

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

T é o conjunto dos valores de verdade

definida indutivamente por

$$\begin{array}{lcl}
\mathcal{B}[\![\mathtt{true}]\!]s & = & \mathbf{tt} \\
\mathcal{B}[\![\mathtt{false}]\!]s & = & \mathbf{ff} \\
\mathcal{B}[\![a_1 = a_2]\!]s & = & \begin{cases} \mathbf{tt} & \text{if } \mathcal{A}[\![a_1]\!]s = \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & \text{if } \mathcal{A}[\![a_1]\!]s \neq \mathcal{A}[\![a_2]\!]s \end{cases} \\
\mathcal{B}[\![a_1 \leq a_2]\!]s & = & \begin{cases} \mathbf{tt} & \text{if } \mathcal{A}[\![a_1]\!]s \leq \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & \text{if } \mathcal{A}[\![a_1]\!]s > \mathcal{A}[\![a_2]\!]s \end{cases} \\
\mathcal{B}[\![\neg\ b]\!]s & = & \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff} \\ \mathbf{ff} & \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt} \end{cases} \\
\mathcal{B}[\![b_1 \wedge b_2]\!]s & = & \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[\![b_1]\!]s = \mathbf{tt} \text{ and } \mathcal{B}[\![b_2]\!]s = \mathbf{tt} \\ \mathbf{ff} & \text{if } \mathcal{B}[\![b_1]\!]s = \mathbf{ff} \text{ or } \mathcal{B}[\![b_2]\!]s = \mathbf{ff} \end{cases}
\end{array}$$

Variáveis livres

$\mathrm{FV}(a)$ denota o conjunto das *variáveis livres* de uma expressão aritmética a

Define-se indutivamente do seguinte modo

$$\begin{array}{lcl}
\mathrm{FV}(n) & = & \emptyset \\
\mathrm{FV}(x) & = & \{\ x\ \} \\
\mathrm{FV}(a_1 + a_2) & = & \mathrm{FV}(a_1) \cup \mathrm{FV}(a_2) \\
\mathrm{FV}(a_1 \star a_2) & = & \mathrm{FV}(a_1) \cup \mathrm{FV}(a_2) \\
\mathrm{FV}(a_1 - a_2) & = & \mathrm{FV}(a_1) \cup \mathrm{FV}(a_2)
\end{array}$$

De modo similar define-se $\mathrm{FV}(b)$ para expressões booleanas.

Substituições

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

Define-se indutivamente do seguinte modo

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

Algumas propriedades

Sejam s e s' estados tais que $s\ x = s'\ x$ para todo x em $\mathrm{FV}(a)$. Então, $\mathcal{A}[\![a]\!]s = \mathcal{A}[\![a]\!]s'$.

Prova: por indução na estrutura de a .

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 booleadas.

Semântica Operacional

A linguagem **While**

Categorias sintáticas e respectivas *meta-variáveis*.

n will range over numerals, **Num**, (em notação decimal)
x will range over variables, **Var**,
a will range over arithmetic expressions, **Aexp**,
b will range over boolean expressions, **Bexp**, and
S will range over statements, **Stm**. (os comandos)

A linguagem **While**

Sintaxe abstracta

$$
\begin{array}{lcl}
a & ::= & n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2 \\
b & ::= & \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\
S & ::= & x := a \mid \texttt{skip} \mid S_1 \,;\, S_2 \mid \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2 \\
 & \mid & \texttt{while}\ b\ \texttt{do}\ S
\end{array}
$$

Semântica Operacional

- A semântica já dada às expressões aritméticas e booleanas apenas *consultam* o estado.
- O papel de um comando da linguagem é alterar o estado. A semântica dos comandos irá portanto *modificar* o estado.
- A semântica operacional preocupa-se em *como* os programas são executados e não apenas nos resultados da sua execução.

Semântica Operacional

Duas abordagens:

- **Semântica Natural** (*Semântica de Avaliação* ou *Big-step*)

  O seu propósito é descrever como os ***resultados globais*** das execuções são obtidos.

- **Semântica Operacional Estrutural** (*Semântica de Transições* ou *Small-step*)

  O seu propósito é descrever como os ***passos individuais*** das computações são executados.

Sistemas de Transições

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

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

$\langle S, s \rangle$ indica que o comando S é para ser executado no estado s.

s representa um estado final (é uma ***configuração terminal***)

A relação de transição descreve como a execução é feita.