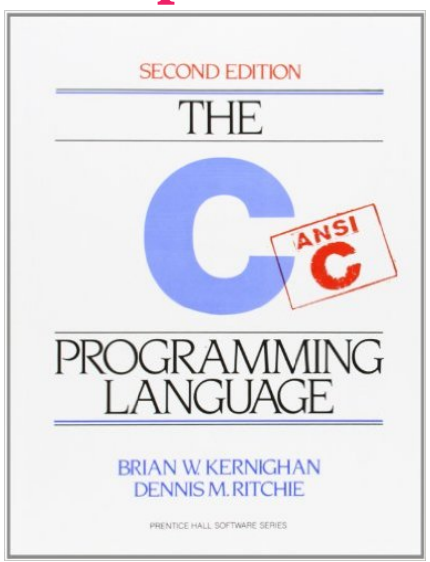
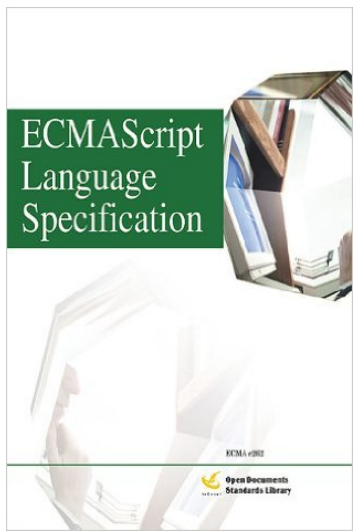
Semântica de linguagens de programação

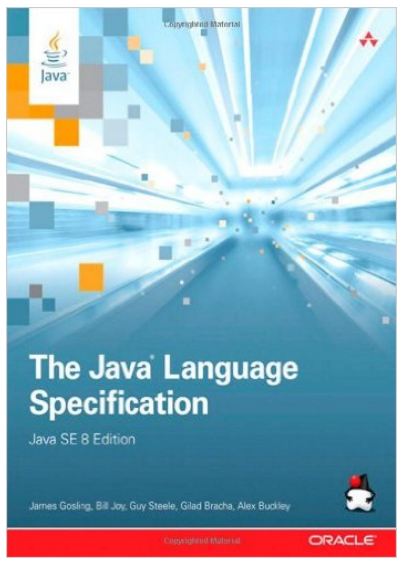
A semântica da maior parte das linguagens de programação é definida informalmente, isto é, utilizando-se linguagem natural (português, inglês).

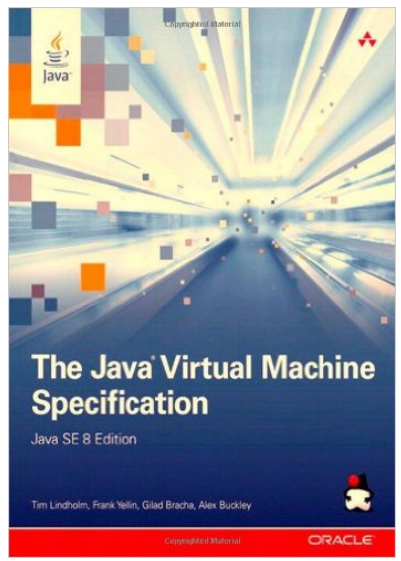
Essas especificações textuais podem ser registradas como um padrão (ISO, ANSI, ABNT, . . .) ou não.

Exemplo:









Semântica utilizando linguagem natural (informal)

Vantagens de se usar linguagem natural (para descrever linguagens):

- não requer conhecimento prévio (para ler a especificação)
- permite transmitir a intuição e propósito por trás de certas construções
- particularmente útil para o usuário da linguagem

Desvantagens da linguagem natural

- não é possível estudar a linguagem matematicamente
- não é particularmente útil para quem implementa o compilador
- bastante suscetível a ambiguidades

Limitações da semântica informal

Conhecer profundamente a semântica de uma linguagem de programação é requerido para

- implementação de compiladores
- análise de código fonte
- definir e estudar transformações de código e traduções entre linguagens

Uma descrição informal não permite a prova de **propriedades** da linguagem como um todo, assim como da **correção** de programas individuais escritos nela.

Pergunta: por que não utilizar um descrição rigorosa, formal e baseada em matemática para descrever a semântica de linguagens de programação?

Semântica formal

Dar **semântica formal** para uma linguagem de programação é determinar o significado de seus programas de forma precisa, utilizando para tal objetos matemáticos.

Por **significado** podemos entender

- o sentido operacional (efeito na máquina)
- o sentido denotacional (função computada)
- satisfação de uma especificação (correção de programas)

Por **objetos matemáticos** podemos entender

- conjuntos, funções, relações, estruturas algébricas, ...

Semântica formal vs semântica informal

Vantagens da semântica formal

- descrição precisa, menos suscetível a ambiguidades
- possibilidade de provar propriedades da linguagem, de programas e de transformações entre programas

Desvantagens da semântica formal

- notação pesada, incluindo letras gregas em excesso (quase todas . . .)
- requer uma maturidade matemática maior (indução matemática para definições e provas, técnicas de prova usando sistemas dedutivos, etc).

Abordagens para semântica formal

Semântica operacional o foco é descrever o **efeito do programa** na **execução** de uma máquina abstrata. Particularmente útil como referência para implementação de compiladores.

Semântica denotacional o foco é associar o programa a um objeto matemático que descreve o seu **efeito final**, de forma composicional. Interessante para determinar equivalência de programas.

Semântica axiomática o foco é determinar a correção de programas individuais com base em uma especificação de seu comportamento (usualmente pré- e pós-condições).

Conteúdo

Semântica operacional

Ideia fundamental:

- definir um **conjunto de estados** (configurações) Ω de uma máquina abstrata;
- definir uma **relação de transição** $R \subseteq \Omega \times \Omega$.

Dois estilos de semântica operacional:

- *small-step*: a relação R associa cada estado ao próximo na execução da máquina;
- *big-step*: a relação R associa cada estado ao estado final de parada da máquina (quando este existe).

Sintaxe

Para ilustrar os dois estilos de semântica operacional, utilizaremos uma linguagem de expressões simples, denominada L0 (*Types and Programming Languages, Chapter 3*)

Definição: O conjunto `Terms` (de termos/programas/expressões) é definido como o **menor** conjunto tal que as seguintes propriedades valem.

1. $\texttt{true} \in \texttt{Terms}$
2. $\texttt{false} \in \texttt{Terms}$
3. $\texttt{0} \in \texttt{Terms}$
4. se $\texttt{t} \in \texttt{Terms}$ então $\texttt{succ t} \in \texttt{Terms}$
5. se $\texttt{t}_1, \texttt{t}_2, \texttt{t}_3 \in \texttt{Terms}$ então $\texttt{if t}_1 \texttt{ then t}_2 \texttt{ else t}_3 \in \texttt{Terms}$
6. se $\texttt{t} \in \texttt{Terms}$ então $\texttt{iszero t} \in \texttt{Terms}$
7. se $\texttt{t} \in \texttt{Terms}$ então $\texttt{pred t} \in \texttt{Terms}$

Sintaxe: definição abreviada

$$
\begin{array}{lcl}
\mathtt{t}, \mathtt{t}_1, \ldots & \in & \mathtt{Terms} \\
\mathtt{t} & ::= & \mathtt{true} \\
 & | & \mathtt{false} \\
 & | & \mathtt{if\ t_1\ then\ t_2\ else\ t_3} \\
 & | & \mathtt{0} \\
 & | & \mathtt{succ\ t_1} \\
 & | & \mathtt{pred\ t_1} \\
 & | & \mathtt{iszero\ t_1}
\end{array}
$$

Devemos entender a notação acima como uma forma abreviada de apresentar a mesma definição indutiva de `Terms` vista no slide anterior.

Ao invés de $\mathtt{if\ t_1\ then\ t_2\ else\ t_3}$ poderiamos ter optado por $\mathtt{if}(\mathtt{t_1}, \mathtt{t_2}, \mathtt{t_3})$

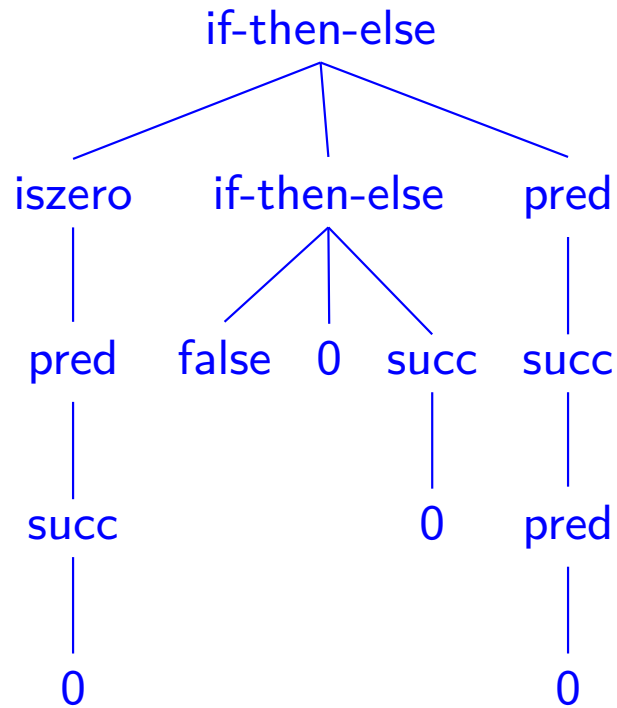
Termos como árvores de sintaxe

Nota: os termos de `Terms` devem representam árvores de sintaxe abstrata, portanto questões relacionadas à sintaxe concreta (ambiguidade de parsing, possibilidade ou não de parênteses, etc.) não serão tratadas.

Exemplo: o termo

```
if iszero (pred (succ 0))
  then if false then 0 else succ 0
   else pred (succ (pred 0))
```

é a forma textual da árvore de sintaxe abstrata apresentada à direita.



Valores

Consideraremos cada programa distinto $\mathtt{t} \in \mathtt{Terms}$ como um **estado** possível da máquina de estados.

Definimos um subconjunto $\mathtt{Values} \subseteq \mathtt{Terms}$ contendo termos que possuem sentido por si próprio. Elementos desse conjunto são considerados os **valores** da linguagem.

Definição: o conjunto $\mathtt{Values}$ é definido como segue:

$$
\begin{array}{lcl}
\mathtt{nv}, \mathtt{nv}_1, \ldots & \in & \mathtt{NV} \quad \text{(numeric Values)} \\
\mathtt{nv} & ::= & \mathtt{0} \mid \mathtt{succ}\ \mathtt{nv}_1
\end{array}
$$

$$
\begin{array}{lcl}
\mathtt{v}, \mathtt{v}_1, \ldots & \in & \mathtt{Values} \\
\mathtt{v} & ::= & \mathtt{true} \mid \mathtt{false} \mid \mathtt{nv}
\end{array}
$$

Semântica operacional *small-step*

Na semântica operacional *small-step*, definimos uma relação $\texttt{step} \subseteq \texttt{Terms} \times \texttt{Terms}$ indicando o próximo estado da máquina a partir do estado atual.

Notação: usaremos $\longrightarrow$ como um sinônimo para `step`. Quando quisermos indicar $(a, b) \in \texttt{step}$, escrevemos $a \longrightarrow b$. Similarmente, para $(a, b) \notin \texttt{step}$ escrevemos $a \not\longrightarrow b$

Exemplo: de alguns pares que devem estar na relação step.

- `iszero 0` $\longrightarrow$ `true`
- `if true then 0 else succ 0` $\longrightarrow$ `0`
- `if iszero (succ 0) then 0 else succ 0`
  $\longrightarrow$
  `if false then 0 else succ 0`

Semântica operacional *small-step* (cont.)

Exemplo: de pares que **não** devem estar na relação `step`

- $\texttt{iszero 0} \not\longrightarrow \texttt{false}$
- $\texttt{true} \not\longrightarrow \texttt{true}$
- $\texttt{if iszero (succ 0) then 0 else succ 0} \not\longrightarrow \texttt{true}$

Nota: não é desejável que valores sejam avaliados, visto que eles *já possuem significado próprio*.

Usaremos **indução** para determinar $\longrightarrow$ de forma precisa a partir de um **conjunto finito de regras**.

Semântica operacional *small-step*: regras I

Definição: $\longrightarrow \subseteq \mathtt{Terms} \times \mathtt{Terms}$ é a **menor** relação tal que as seguintes regras são válidas.

$$\frac{}{\mathtt{if\ true\ then\ t_2\ else\ t_3} \longrightarrow \mathtt{t_2}} \quad \text{(E-IfTrue)}$$

$$\frac{}{\mathtt{if\ false\ then\ t_2\ else\ t_3} \longrightarrow \mathtt{t_3}} \quad \text{(E-IfFalse)}$$

$$\frac{\mathtt{t} \longrightarrow \mathtt{t'}}{\mathtt{if\ t\ then\ t_2\ else\ t_3} \longrightarrow \mathtt{if\ t'\ then\ t_2\ else\ t_3}} \quad \text{(E-IF)}$$

$$\frac{\mathtt{t} \longrightarrow \mathtt{t'}}{\mathtt{succ\ t} \longrightarrow \mathtt{succ\ t'}} \quad \text{(E-Succ)}$$

$$\vdots$$

Semântica operacional *small-step*: regras II

$$\vdots$$

$$\frac{}{\texttt{pred 0} \longrightarrow \texttt{0}} \quad \text{(E-PredZero)}$$

$$\frac{}{\texttt{pred (succ nv)} \longrightarrow \texttt{nv}} \quad \text{(E-PredSucc)}$$

$$\frac{\texttt{t} \longrightarrow \texttt{t'}}{\texttt{pred t} \longrightarrow \texttt{pred t'}} \quad \text{(E-Pred)}$$

$$\frac{}{\texttt{iszero 0} \longrightarrow \texttt{true}} \quad \text{(E-IsZeroZero)}$$

$$\frac{}{\texttt{iszero (succ nv)} \longrightarrow \texttt{false}} \quad \text{(E-IsZeroSucc)}$$

$$\frac{\texttt{t} \longrightarrow \texttt{t'}}{\texttt{iszero t} \longrightarrow \texttt{iszero t'}} \quad \text{(E-IsZero)}$$

Semântica operacional *small-step*: exemplo

1)

$$\dfrac{\dfrac{\dfrac{}{\texttt{pred succ 0} \longrightarrow \texttt{0}}\,\text{(E-PredSucc)}}{\texttt{iszero pred succ 0} \longrightarrow \texttt{iszero 0}}\,\text{(E-IsZero)}}{\texttt{if iszero pred succ 0 then if false then 0 else succ 0 else pred succ pred 0} \longrightarrow \texttt{if iszero 0 then if false then 0 else succ 0 else pred succ pred 0}}\,\text{(E-If)}$$

2)

$$\dfrac{\dfrac{}{\texttt{iszero 0} \longrightarrow \texttt{true}}\,\text{(E-IsZeroZero)}}{\texttt{if iszero 0 then if false then 0 else succ 0 else pred succ pred 0} \longrightarrow \texttt{if true then if false then 0 else succ 0 else pred succ pred 0}}\,\text{(E-If)}$$