Machete Cálculo lambda

17 de diciembre de 2024

Machete: Tipos y Términos

Las expresiones de tipos (o simplemente tipos) son:

$$\sigma ::= \text{Bool} \mid \text{Nat} \mid \sigma \to \sigma$$

Los términos están dados por:

$$M ::= x \mid \lambda x : \sigma.M \mid MM \mid \text{true} \mid \text{false} \mid \text{if } M \text{ then } M \text{ else } M \mid \text{zero} \mid \text{succ}(M) \mid \text{pred}(M) \mid \text{isZero}(M)$$

Machete: Axiomas y Reglas de Tipado

$$\Gamma \vdash \text{true} : \text{Bool} \qquad (\text{axtrue})$$

$$\Gamma \vdash \text{false} : \text{Bool} \qquad (\text{axfalse})$$

$$\Gamma, x : \sigma \vdash x : \sigma \qquad (\text{axv})$$

$$\frac{\Gamma \vdash M : \text{Bool} \quad \Gamma \vdash P : \sigma \quad \Gamma \vdash Q : \sigma}{\Gamma \vdash \text{if } M \text{ then } P \text{ else } Q : \sigma} \quad (\text{if})$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x : \sigma.M : \sigma \to \tau} \quad (\to\text{i})$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} \quad (\to\text{e})$$

$$\Gamma \vdash \text{zero} : \text{Nat} \qquad (\text{axzero})$$

$$\frac{\Gamma \vdash M : \text{Nat}}{\Gamma \vdash \text{succ}(M) : \text{Nat}} \quad (\text{succ})$$

$$\frac{\Gamma \vdash M : \text{Nat}}{\Gamma \vdash \text{pred}(M) : \text{Nat}} \quad (\text{pred})$$

$$\frac{\Gamma \vdash M : \text{Nat}}{\Gamma \vdash \text{isZero}(M) : \text{Bool}} \quad (\text{isZero})$$

Semántica Operacional

Valores

$$V ::= \text{true} \mid \text{false} \mid \lambda x:\sigma.M \mid \text{zero} \mid \text{succ}(V)$$

(Los valores de tipo Nat pueden escribirse como n, lo cual abrevia $\text{succ}^n(\text{zero})$).

Reglas de Evaluación en un Paso

$$\frac{M_1 \to M_1'}{M_1 M_2 \to M_1' M_2} \quad (app_l \text{ o } \mu)$$

$$\frac{M_2 \to M_2'}{V M_2 \to V M_2'} \quad (app_r \text{ o } \nu)$$

$$(\lambda x:\sigma.M)V \to M\{x := V\} \quad (\beta)$$

$$\text{if true then } M_2 \text{ else } M_3 \to M_2 \quad (if_t)$$

$$\text{if false then } M_2 \text{ else } M_3 \to M_3 \quad (if_f)$$

$$\frac{M_1 \to M_1'}{\text{if } M_1 \text{ then } M_2 \text{ else } M_3 \to \text{if } M_1' \text{ then } M_2 \text{ else } M_3} \quad (if_c)$$

Reglas Adicionales

$$\text{pred}(\text{succ}(n)) \to n \quad (\text{pred})$$

$$\text{pred}(\text{zero}) \to \text{zero} \quad (pred_0, \text{ opcional})$$

$$\text{isZero}(\text{zero}) \to \text{true} \quad (isZero_0)$$

$$\text{isZero}(\text{succ}(n)) \to \text{false} \quad (isZero_n)$$

$$\frac{M \to N}{\text{succ}(M) \to \text{succ}(N)} \quad (succ_c)$$

$$\frac{M \to N}{\text{pred}(M) \to \text{pred}(N)} \quad (pred_c)$$

$$\frac{M \to N}{\text{isZero}(M) \to \text{isZero}(N)} \quad (isZero_c)$$

Deducción natural

Reglas básicas

$$\frac{}{\Gamma, \tau \vdash \tau}\ \textsf{ax}$$

$$\frac{\Gamma \vdash \tau \quad \Gamma \vdash \sigma}{\Gamma \vdash \tau \wedge \sigma}\ \wedge_i \qquad \frac{\Gamma \vdash \tau \wedge \sigma}{\Gamma \vdash \tau}\ \wedge_{e_1} \qquad \frac{\Gamma \vdash \tau \wedge \sigma}{\Gamma \vdash \sigma}\ \wedge_{e_2}$$

$$\frac{\Gamma, \tau \vdash \sigma}{\Gamma \vdash \tau \Rightarrow \sigma}\ \Rightarrow_i \qquad \frac{\Gamma \vdash \tau \Rightarrow \sigma \quad \Gamma \vdash \tau}{\Gamma \vdash \sigma}\ \Rightarrow_e$$

$$\frac{\Gamma \vdash \tau}{\Gamma \vdash \tau \vee \sigma}\ \vee_{i_1} \qquad \frac{\Gamma \vdash \sigma}{\Gamma \vdash \tau \vee \sigma}\ \vee_{i_2} \qquad \frac{\Gamma \vdash \tau \vee \sigma \quad \Gamma, \tau \vdash \rho \quad \Gamma, \sigma \vdash \rho}{\Gamma \vdash \rho}\ \vee_e$$

$$\frac{\Gamma, \tau \vdash \bot}{\Gamma \vdash \neg\tau}\ \neg_i \qquad \frac{\Gamma \vdash \tau \quad \Gamma \vdash \neg\tau}{\Gamma \vdash \bot}\ \neg_e$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash \tau}\ \bot_e$$

Lógica intuicionista

$$\frac{\Gamma \vdash \neg\neg\tau}{\Gamma \vdash \tau}\ \neg\neg_e$$

Lógica clásica

Reglas intuicionistas

$$\frac{\Gamma \vdash \tau}{\Gamma \vdash \neg\neg\tau}\ \neg\neg_i \qquad \frac{\Gamma \vdash \tau \Rightarrow \sigma \quad \Gamma \vdash \neg\sigma}{\Gamma \vdash \neg\tau}\ \text{MT}$$

Reglas clásicas

$$\frac{\Gamma, \neg\tau \vdash \bot}{\Gamma \vdash \tau}\ \text{PBC} \qquad \frac{}{\Gamma \vdash \tau \vee \neg\tau}\ \text{LEM}$$

Cuantificación universal

Regla de eliminación

$$\frac{\Gamma \vdash \forall X.\, \sigma}{\Gamma \vdash \sigma\{X := t\}}\ \forall\mathrm{E}$$

Regla de introducción

$$\frac{\Gamma \vdash \sigma \qquad X \notin \mathsf{fv}(\Gamma)}{\Gamma \vdash \forall X.\, \sigma}\ \forall\mathrm{I}$$

Cuantificación existencial

Regla de introducción

$$\frac{\Gamma \vdash \sigma\{X := t\}}{\Gamma \vdash \exists X.\, \sigma}\ \exists\mathrm{I}$$

Regla de eliminación

$$\frac{\Gamma \vdash \exists X.\, \sigma \qquad \Gamma, \sigma \vdash \tau \qquad X \notin \mathsf{fv}(\Gamma, \tau)}{\Gamma \vdash \tau}\ \exists\mathrm{E}$$

Inferencia de Tipos
Machete

Paradigmas (de Lenguajes) de Programación

1. Algoritmo de inferencia

- $\mathbb{W}(x) \rightsquigarrow \{x : \mathrm{X}_k\} \vdash x : \mathrm{X}_k, \quad \mathrm{X}_k$ incógnita fresca
- $\mathbb{W}(\mathit{0}) \rightsquigarrow \emptyset \vdash \mathit{0} : \mathit{Nat}$
- $\mathbb{W}(\mathit{true}) \rightsquigarrow \emptyset \vdash \mathit{true} : \mathit{Bool}$
- $\mathbb{W}(\mathit{false}) \rightsquigarrow \emptyset \vdash \mathit{false} : \mathit{Bool}$
- $\mathbb{W}(\mathit{succ}(U)) \rightsquigarrow S(\Gamma) \vdash S(\mathit{succ}(M)) : \mathit{Nat}$ donde
  - $\mathbb{W}(U) = \Gamma \vdash M : \tau$
  - $S = \mathit{MGU}\{\tau \stackrel{?}{=} \mathit{Nat}\}$
- $\mathbb{W}(\mathit{pred}(U)) \rightsquigarrow S(\Gamma) \vdash S(\mathit{pred}(M)) : \mathit{Nat}$ donde
  - $\mathbb{W}(U) = \Gamma \vdash M : \tau$
  - $S = \mathit{MGU}\{\tau \stackrel{?}{=} \mathit{Nat}\}$
- $\mathbb{W}(\mathit{iszero}(U)) \rightsquigarrow S(\Gamma) \vdash S(\mathit{iszero}(M)) : \mathit{Bool}$ donde
  - $\mathbb{W}(U) = \Gamma \vdash M : \tau$
  - $S = \mathit{MGU}\{\tau \stackrel{?}{=} \mathit{Nat}\}$
- $\mathbb{W}(\mathit{if}\ U\ \mathit{then}\ V\ \mathit{else}\ W) \rightsquigarrow S(\Gamma_1) \cup S(\Gamma_2) \cup S(\Gamma_3) \vdash S(\mathit{if}\ M\ \mathit{then}\ P\ \mathit{else}\ Q) : S(\sigma)$ donde
  - $\mathbb{W}(U) = \Gamma_1 \vdash M : \rho$
  - $\mathbb{W}(V) = \Gamma_2 \vdash P : \sigma$
  - $\mathbb{W}(W) = \Gamma_3 \vdash Q : \tau$
  - $S = \mathit{MGU}\{\sigma \stackrel{?}{=} \tau, \rho \stackrel{?}{=} \mathit{Bool}\} \cup \{\sigma_1 \stackrel{?}{=} \sigma_2 \mid x : \sigma_1 \in \Gamma_i, x : \sigma_2 \in \Gamma_j, i, j \in \{1,2,3\}\}$
- $\mathbb{W}(\lambda x.U) \rightsquigarrow \Gamma' \vdash \lambda x : \tau'.M : \tau' \to \rho$ donde
  - $\mathbb{W}(U) = \Gamma \vdash M : \rho$
  - $\tau' = \begin{cases} \alpha \text{ si } x : \alpha \in \Gamma \\ \mathrm{X}_k \text{ con } \mathrm{X}_k \text{ variable fresca en otro caso} \end{cases}$
  - $\Gamma' = \Gamma \ominus \{x\}$
- $\mathbb{W}(U\,V) \rightsquigarrow S(\Gamma_1) \cup S(\Gamma_2) \vdash S(M\,N) : S(\mathrm{X}_k)$ donde
  - $\mathbb{W}(U) = \Gamma_1 \vdash M : \tau$
  - $\mathbb{W}(V) = \Gamma_2 \vdash N : \rho$
  - X_k variable fresca
  - $S = \mathit{MGU}\{\tau \stackrel{?}{=} \rho \to \mathrm{X}_k\} \cup \{\sigma_1 \stackrel{?}{=} \sigma_2 \mid x : \sigma_1 \in \Gamma_1, x : \sigma_2 \in \Gamma_2\}$

2. Algoritmo de unificación (Martelli-Montanari)

2.1. Reglas

Se enuncian las reglas para constructores de tipo C en general de cualquier aridad, y en particular para los constructores de tipo de λ^b

$$\sigma, \tau ::= Nat \mid Bool \mid \sigma \to \tau$$

1. **Descomposición**

   $\{\sigma_1 \to \sigma_2 \overset{?}{=} \tau_1 \to \tau_2\} \cup G \mapsto \{\sigma_1 \overset{?}{=} \tau_1, \sigma_2 \overset{?}{=} \tau_2\} \cup G$

   $\{Bool \overset{?}{=} Bool\} \cup G \mapsto G$

   $\{Nat \overset{?}{=} Nat\} \cup G \mapsto G$

   Caso general

   $\{C(\sigma_1, \dots, \sigma_n) \overset{?}{=} C(\tau_1, \dots, \tau_n)\} \cup G \mapsto \{\sigma_1 \overset{?}{=} \tau_1, \dots, \sigma_n \overset{?}{=} \tau_n\} \cup G$

2. **Eliminación de par trivial**

   $\{\mathrm{X}_k \overset{?}{=} \mathrm{X}_k\} \cup G \mapsto G$

3. **Swap**: si σ no es una variable

   $\{\sigma \overset{?}{=} \mathrm{X}_k\} \cup G \mapsto \{\mathrm{X}_k \overset{?}{=} \sigma\} \cup G$

4. **Eliminación de variable**: si $\mathrm{X}_k \notin FV(\sigma)$

   $\{\mathrm{X}_k \overset{?}{=} \sigma\} \cup G \mapsto_{\{\mathrm{X}_k := \sigma\}} G\{\mathrm{X}_k := \sigma\}$

5. **Colisión**

   $\{\sigma \overset{?}{=} \tau\} \cup G \mapsto$ **falla**, con $(\sigma, \tau) \in T \cup T^{-1}$ donde

   $T = \{(Bool, Nat), (Nat, \sigma_1 \to \sigma_2), (Bool, \sigma_1 \to \sigma_2)\}$ y T^{-1} representa invertir cada par

   Caso general: si $C \neq C'$ son constructores de tipo diferentes

   $\{C(\dots) \overset{?}{=} C'(\dots)\} \cup G \mapsto$ **falla**

6. **Occur check**: si $\mathrm{X}_k \neq \sigma$ y $\mathrm{X}_k \in FV(\sigma)$

   $\{\mathrm{X}_k \overset{?}{=} \sigma\} \cup G \mapsto$ **falla**

2.2. Ejemplos

2.2.1. Secuencia exitosa

$$\begin{array}{ll}
 & \{(Nat \to \mathrm{X}_1) \to (\mathrm{X}_1 \to \mathrm{X}_3) \overset{?}{=} \mathrm{X}_2 \to (\mathrm{X}_4 \to \mathrm{X}_4) \to \mathrm{X}_2\} \\
\mapsto^1 & \{Nat \to \mathrm{X}_1 \overset{?}{=} \mathrm{X}_2, \mathrm{X}_1 \to \mathrm{X}_3 \overset{?}{=} (\mathrm{X}_4 \to \mathrm{X}_4) \to \mathrm{X}_2\} \\
\mapsto^3 & \{\mathrm{X}_2 \overset{?}{=} Nat \to \mathrm{X}_1, \mathrm{X}_1 \to \mathrm{X}_3 \overset{?}{=} (\mathrm{X}_4 \to \mathrm{X}_4) \to \mathrm{X}_2\} \\
\mapsto^4_{\{\mathrm{X}_2 := Nat \to \mathrm{X}_1\}} & \{\mathrm{X}_1 \to \mathrm{X}_3 \overset{?}{=} (\mathrm{X}_4 \to \mathrm{X}_4) \to (Nat \to \mathrm{X}_1)\} \\
\mapsto^1 & \{\mathrm{X}_1 \overset{?}{=} \mathrm{X}_4 \to \mathrm{X}_4, \mathrm{X}_3 \overset{?}{=} Nat \to \mathrm{X}_1\} \\
\mapsto^4_{\{\mathrm{X}_1 := \mathrm{X}_4 \to \mathrm{X}_4\}} & \{\mathrm{X}_3 \overset{?}{=} Nat \to (\mathrm{X}_4 \to \mathrm{X}_4)\} \\
\mapsto^4_{\{\mathrm{X}_3 := Nat \to (\mathrm{X}_4 \to \mathrm{X}_4)\}} & \emptyset
\end{array}$$

El MGU es

$$\begin{aligned}
&\{\mathrm{X}_3 := Nat \to (\mathrm{X}_4 \to \mathrm{X}_4)\} \circ \{\mathrm{X}_1 := \mathrm{X}_4 \to \mathrm{X}_4\} \circ \{\mathrm{X}_2 := Nat \to \mathrm{X}_1\} \\
= &\{\mathrm{X}_2 := Nat \to (\mathrm{X}_4 \to \mathrm{X}_4),\ \mathrm{X}_1 := \mathrm{X}_4 \to \mathrm{X}_4,\ \mathrm{X}_3 := Nat \to (\mathrm{X}_4 \to \mathrm{X}_4)\}
\end{aligned}$$

2.2.2. Secuencia fallida

$$
\begin{array}{ll}
 & \{X_1 \to (X_2 \to X_1) \stackrel{?}{=} X_2 \to ((X_1 \to Nat) \to X_1)\} \\
\mapsto^1 & \{X_1 \stackrel{?}{=} X_2, X_2 \to X_1 \stackrel{?}{=} (X_1 \to Nat) \to X_1\} \\
\mapsto^4_{\{X_1:=X_2\}} & \{X_2 \to X_2 \stackrel{?}{=} (X_2 \to Nat) \to X_2\} \\
\mapsto^1 & \{X_2 \stackrel{?}{=} X_2 \to Nat, X_2 \stackrel{?}{=} X_2\} \\
\mapsto^6 & \textbf{falla}
\end{array}
$$

2.2.3. Constructores en general

Se usan los constructores de tipos de listas,

$$\sigma ::= \ldots \mid [\sigma]$$

$$
\begin{array}{ll}
 & \{(X_3 \to X_4 \to X_4) \to X_4 \to [X_3] \to X_4 \stackrel{?}{=} ((X_1 \to X_2) \to [X_1] \to [X_2]) \to X_5\} \\
\mapsto^1 & \{X_3 \to X_4 \to X_4 \stackrel{?}{=} (X_1 \to X_2) \to [X_1] \to [X_2], X_4 \to [X_3] \to X_4 \stackrel{?}{=} X_5\} \\
\mapsto^3 & \{X_3 \to X_4 \to X_4 \stackrel{?}{=} (X_1 \to X_2) \to [X_1] \to [X_2],\ X_5 \stackrel{?}{=} X_4 \to [X_3] \to X_4\} \\
\mapsto^4_{\{X_5:=X_4\to[X_3]\to X_4\}} & \{X_3 \to X_4 \to X_4 \stackrel{?}{=} (X_1 \to X_2) \to [X_1] \to [X_2]\} \\
\mapsto^1 & \{X_3 \stackrel{?}{=} X_1 \to X_2,\ X_4 \to X_4 \stackrel{?}{=} [X_1] \to [X_2]\} \\
\mapsto^4_{\{X_3:=X_1\to X_2\}} & \{X_4 \to X_4 \stackrel{?}{=} [X_1] \to [X_2]\} \\
\mapsto^1 & \{X_4 \stackrel{?}{=} [X_1],\ X_4 \stackrel{?}{=} [X_2]\} \\
\mapsto^4_{\{X_4:=[X_1]\}} & \{[X_1] \stackrel{?}{=} [X_2]\} \\
\mapsto^1 & \{X_1 \stackrel{?}{=} X_2\} \\
\mapsto^4_{\{X_1:=X_2\}} & \emptyset
\end{array}
$$

El MGU es

$$
\begin{array}{l}
\{X_1 := X_2\} \circ \{X_4 := [X_1]\} \circ \{X_3 := X_1 \to X_2\} \circ \{X_5 := X_4 \to [X_3] \to X_4\} \\
= \{X_5 := X_{[X_2]} \to [X_2 \to X_2] \to X_{[X_2]},\ X_3 := X_2 \to X_2,\ X_4 := [X_2],\ X_1 := X_2\}
\end{array}
$$

```
#(1 2 3 4) collect: [ :each | each * 2 ]   ⟶   #( 2 4 6 8 )
#(1 2 3 4)
    inject: 0
    into: [ :each :result | each + result ]   ⟶   10
```

```
" testing "
#( 2 4 ) anySatisfy: [ :each | each odd ]   ⟶   false
#( 2 4 ) allSatisfy: [ :each | each even ]   ⟶   true
```

```
" finding "
'abcdef' includes: $e   ⟶   true
'abcdef' contains: [ :each | each isUppercase ]   ⟶   false
'abcdef'
    detect: [ :each | each isVowel ]
    ifNone: [ $u ]   ⟶   $a
```

```
" String – a collection of characters "
string := 'abc'.
string := string , 'DEF'   ⟶   'abcDEF'
string beginsWith: 'abc'   ⟶   true
string endsWith: 'abc'   ⟶   false
string includesSubString: 'cD'   ⟶   true
string asLowercase   ⟶   'abcdef
string asUppercase   ⟶   'ABCDEF'
```

```
" OrderedCollection – an ordered collection of objects "
ordered := OrderedCollection new.
ordered addLast: 'world'.
ordered addFirst: 'hello'.
ordered size   ⟶   2
ordered at: 2   ⟶   'world'
ordered removeLast   ⟶   'world'
ordered removeFirst   ⟶   'hello'
ordered isEmpty   ⟶   true
```

```
" Set – an unordered collection of objects without duplicates "
set := Set new.
set add 'hello'; add: 'hello'.
set size   ⟶   1
```

```
" Bag – an unordered collection of objects with duplicates "
bag := Bag new.
bag add: 'this'; add: 'that'; add: 'that'.
bag occurrencesOf: 'that'   ⟶   2
bag remove: 'that'.
bag occurrencesOf: 'that'   ⟶   1
```

```
" Dictionary – associates unique keys with objects "
dictionary := Dictionary new.
dictionary at: 'smalltalk' put: 80.
dictionary at: 'smalltalk'   ⟶   80
dictionary at: 'squeak' ifAbsent: [ 82 ]   ⟶   82
dictionary removeKey: 'smalltalk'.
dictionary isEmpty   ⟶   true
```

Streams

```
" ReadStream – to read a sequence of objects from a collection
"
stream := 'Hello World' readStream.
stream next   ⟶   $H
stream upTo: $o   ⟶   'ell'
stream skip: 2.
stream peek   ⟶   $o
stream upToEnd   ⟶   'orld'
```

```
" WriteStream – to write a sequence of objects to a collection "
stream := WriteStream on: Array new.
stream nextPut: 'Hello'.
stream nextPutAll: #( 1 2 3 ).
stream contents   ⟶   #( 'Hello' 1 2 3 )
```

File Streams

```
fileStream := FileDirectory default newFileNamed: 'tmp.txt'.
fileStream nextPutAll: 'my cool stuff'.
fileStream close.

fileStream := FileDirectory default oldFileNamed: 'tmp.txt'.
fileStream contents   ⟶   'my cool stuff'
```

Method Definition

```
messageSelectorAndArgumentNames
    "comment stating purpose of message"
    | temporary variable names |
    statements
```

Class Definition

```
Object subclass: #NameOfSubclass
    instanceVariableNames: 'instVar1 instVar2'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Category–Name'
```

References

1. Andrew Black, Stéphane Ducasse, Oscar Nierstrasz and Damien Pollet, *Squeak by Example*, Square Bracket Associates, 2007, `squeakbyexample.org`.
2. Chris Rathman, *Terse guide to Squeak*, `wiki.squeak.org/squeak/5699`.
3. *Smalltalk*, Wikipedia, the free encyclopedia, `en.wikipedia.org/wiki/Smalltalk`.

Smalltalk Cheat Sheet

Software Composition Group
University of Bern

May 21, 2008

1. The Environment

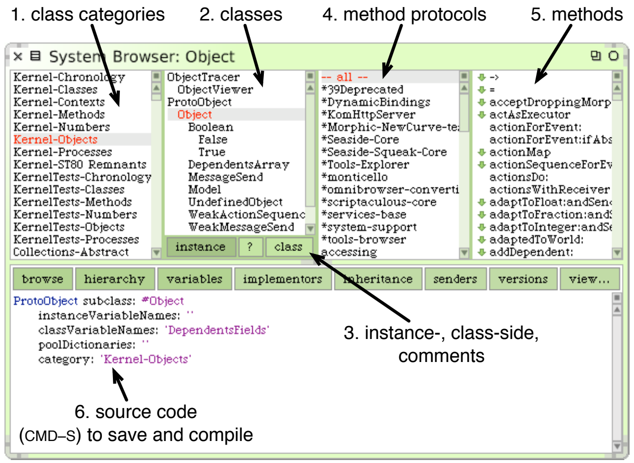

Figure 1: The Smalltalk Code Browser

- Do it (CMD–D): Evaluate selected code.
- Print it (CMD–P): Display the result of evaluating selected code.
- Debug it: Evaluate selected code step-by-step with the integrated debugger.
- Inspect it (CMD–I): Show an *object inspector* on the result of evaluating selected code.
- Explore it (CMD–SHIFT–I): Show an *object explorer* on the result of evaluating selected code.

2. The Language

- Everything is an object.
- Everything happens by sending messages.
- Single inheritance.
- Methods are public.
- Instance variables are private to objects.

Keywords

- self, the receiver.
- super, the receiver, method lookup starts in superclass.
- nil, the unique instance of the class UndefinedObject.
- true, the unique instance of the class True.
- false, the unique instance of the class False.
- thisContext, the current execution context.

Literals

- Integer
  123
  2r1111011 (123 in binary)
  16r7B (123 in hexadecimal)
- Float
  123.4
  1.23e−4
- Character
  $a
- String
  'abc'
- Symbol
  #abc
- Array
  #(123 123.4 $a 'abc' #abc)

Message Sends

1. *Unary messages* take no argument.
   1 factorial sends the message factorial to the object 1.
2. *Binary messages* take exactly one argument.
   3 + 4 sends message + with argument 4 to the object 3.
   #answer −> 42 sends −> with argument 42 to #answer.
   Binary selectors are built from one or more of the characters +, −, *, =, <, >, ...
3. *Keyword messages* take one or more arguments.
   2 raisedTo: 6 modulo: 10 sends the message named raisedTo:modulo: with arguments 6 and 10 to the object 2.

Unary messages are sent first, then binary messages and finally keyword messages:

2 raisedTo: 1 + 3 factorial ⟶ 128

Messages are sent left to right. Use parentheses to change the order:

1 + 2 * 3 ⟶ 9
1 + (2 * 3) ⟶ 7

Syntax

- Comments
  "Comments are enclosed in double quotes"
- Temporary Variables
  | var |
  | var1 var2 |
- Assignment
  var := aStatement
  var1 := var2 := aStatement
- Statements
  aStatement1. aStatement2
  aStatement1. aStatement2. aStatement3
- Messages
  receiver message (unary message)
  receiver + argument (binary message)
  receiver message: argument (keyword message)
  receiver message: argument1 with: argument2
- Cascade
  receiver message1; message2
  receiver message1; message2: arg2; message3: arg3
- Blocks
  [aStatement1. aStatement2]
  [:argument1 | aStatement1. aStatement2]
  [:argument1 :argument2 | | temp1 temp2 | aStatement1]
- Return Statement
  ^ aStatement

3. Standard Classes

Logical expressions

```
true not  ⟶  false
1 = 2 or: [ 2 = 1 ]  ⟶  false
1 < 2 and: [ 2 > 1 ]  ⟶  true
```

Conditionals

```
1 = 2 ifTrue: [ Transcript show: '1 is equal to 2' ].
1 = 2 ifFalse: [ Transcript show: '1 isn''t equal to 2' ].

100 factorial / 99 factorial = 100
   ifTrue: [ Transcript show: 'condition evaluated to true' ]
   ifFalse: [ Beeper beep ].
```

Loops

```
" conditional iteration "
[ Sensor anyButtonPressed ]
   whileFalse: [ "wait" ].

pen := Pen newOnForm: Display.
pen place: Sensor cursorPoint.
[ Sensor anyButtonPressed ]
   whileTrue: [ pen goto: Sensor cursorPoint ].

" fixed iteration "
180 timesRepeat: [
   pen turn: 88.
   pen go: 250 ].

1 to: 100 do: [ :index |
   pen go: index * 4.
   bic turn: 89 ].

" infinite loop (press CMD+. to break) "
[ pen goto: Sensor cursorPoint ] repeat.
```

Blocks (anonymous functions)

```
" evaluation "
[ 1 + 2 ] value  ⟶  3
[ :x | x + 2 ] value: 1  ⟶  3
[ :x :y | x + y ] value: 1 value: 2  ⟶  3

" processes "
[ (Delay forDuration: 5 seconds) wait.
  Transcript show: 'done' ] fork  ⟶  aProcess
```

Collections

```
" iterating "
'abc' do: [ :each | Transcript show: each ].
'abc'
   do: [ :each | Transcript show: each ]
   separatedBy: [ Transcript cr ].

" transforming "
#(1 2 3 4) select: [ :each | each even ]  ⟶  #( 2 4 )
#(1 2 3 4) reject: [ :each | each = 2 ]  ⟶  #( 1 3 4 )
```