Ipsa forma est substantia
Language(s) as a foundation for computer science

Simone Martini

Dipartimento di Informatica – Scienza e Ingegneria
Alma mater studiorum • Università di Bologna
and
EPI Focus • INRIA Sophia / Bologna

HaPoC Symposium at IACAP-14 July 3, 2014

The rules of engagement:

What are programs, algorithms, machines,
and how do we understand their languages?

More generally

How the languages of computer science relate to that discipline?

Some concepts

- information
- effective (computation, procedure, process, . . .)
- feasible
- interaction
- abstraction hierarchy
- . . .

- they are intrinsically tied to the linguistic way we use to express them

More generally

How the languages of computer science relate to that discipline?

Some concepts

- information
- effective (computation, procedure, process, . . .)
- feasible
- interaction
- abstraction hierarchy
- . . .

- they are intrinsically tied to the linguistic way we use to express them

Computation is symbol pushing

Turing's analysis reveals the simple combinatorial structure of computation

A dismissive comment

Turing's "machines": These machines are humans who calculate.

[L. Wittgenstein,
Remarks on the Philosophy of Psychology, Vol. 1,
Blackwell, Oxford, 1980.]

On one hand, it is a great praise
cfr. Church's "evident immediately"

But on the substance, W. misses the point...

The computing machine

- A deep introspective analysis of a human process
- Generates an abstract, combinatorial mathematical concept

- It is a finite, alphabetic description

A Turing's parapraxis (?):

(Mechanism and writing are from our point of view
almost synonymous.)

Turing, A.M. *Computing Machinery and Intelligence.* Mind LIX, p. 456 (1950)

Discuss: J. Lassègue, G. Longo, What is Turing's Comparison between Mechanism and Writing Worth? CiE 2012.

The computing machine

- A deep introspective analysis of a human process
- Generates an abstract, combinatorial mathematical concept

- It is a finite, alphabetic description

A Turing's parapraxis (?):

(Mechanism and writing are from our point of view almost synonymous.)

Turing, A.M. *Computing Machinery and Intelligence*. Mind LIX, p. 456 (1950)

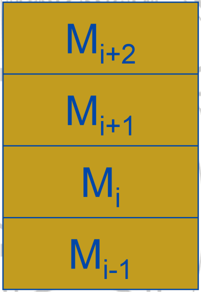
Discuss: J. Lassègue, G. Longo, What is Turing's Comparison between Mechanism and Writing Worth? CiE 2012.

Computation is performed by a machine

- "*The computable*" is invariant
- But "*a computation*" is not:
  a specific combinatorial process,
  happening on a (abstract) machine
- A (universal) abstract machine M exists to interpret (execute) its own language L_M
- A machine *is a black box* for its own language
  [after Von Neumann's *Report on EDVAC*, 1948...]

Hierarchies

Universality allows hierarchies of machines



M_{i+2}

M_{i+1}

M_i

M_{i-1}

Machine M_i:

- uses language $L_{M_{i-1}}$
  "it is written in $L_{M_{i-1}}$"
- to implement its own language L_i
- hides (to some point) machine M_{i-1}

At any level i we do not know (and it is not required to know) which could be level 0

From patterns to abstractions

Cfr. E. Visser. Understanding Software through Linguistic Abstraction, 2013

- A programming pattern:
  A recipe to solve a re-occurring problem; applied manually.
  E.g. Calling/returning sequences in assembly, using the return stack
- A linguistic abstraction:
  A construct providing a “black-box” for that pattern
  E.g. Functions and their parameter passing mechanisms.
- The abstraction gets autonomous life,
  and autonomous semantics!
- It frees the user from the details of level $i-1$: portability

From patterns to abstractions

Cfr. E. Visser. Understanding Software through Linguistic Abstraction, 2013

- A programming pattern:
  A recipe to solve a re-occurring problem; applied manually.
  E.g. Calling/returning sequences in assembly, using the return stack
- A linguistic abstraction:
  A construct providing a “black-box” for that pattern
  E.g. Functions and their parameter passing mechanisms.
- The abstraction gets autonomous life,
  and autonomous semantics!
- It frees the user from the details of level $i - 1$: portability

From patterns to abstractions, 2

Many examples:

- Abstraction on control:
  functions, structured programming, exceptions, semaphores, threads, . . .
- Abstraction on data:
  structured data types, dynamically allocated data, abstract data types, messages . . .
- Abstraction on control and data:
  objects, inheritance, modules, . . .

Programming languages evolve converting new patterns into abstractions, and giving them autonomous life.

From patterns to abstractions, 3

PL need to conquer new fields:

- Concurrency:
  name passing models (π-calculus)
- Real-Time:
  Esterel
- Web services:
  BPEL (Business Process Execution Language), Jolie
- Big data:
  ??
- Cloud:
  ??
- Mobile computing:
  ??

Translations

- Of course we compile a level onto a lower level
- But (some) abstractions at level *i* are *conceptually irreducible* to lower levels: emergent phenomena
- There are no fully faithful translation, even inside the same language

> *A language fills a niche in the honeycomb of potential perceptions and interpretations. It articulates a construct of values, meanings, suppositions which no other language exactly matches or supersedes. [...] We speak worlds.*
>
> *[G. Steiner,* Errata, *ch. 7, p. 99; 1997]*

Translations

- Of course we compile a level onto a lower level
- But (some) abstractions at level *i* are *conceptually irreducible* to lower levels: emergent phenomena
- There are no fully faithful translation, even inside the same language

> *A language fills a niche in the honeycomb of potential perceptions and interpretations. It articulates a construct of values, meanings, suppositions which no other language exactly matches or supersedes. [. . .] We speak worlds.*
>
> *[G. Steiner,* Errata*, ch. 7, p. 99; 1997]*

“Programming” languages

- What we insist in calling programming languages
- Are powerful tools to organize, make coherent, and model reality

  data models
  procedural models
  interaction models
  synchronization models
  organization models
  . . .

New models

Our models are intrinsically different way from the model of, e.g., continuous mathematics (i.e., physics)

- Discrete
- Effective
- Scalable at different abstraction levels

Moreover, and crucially

Our programming languages are also
(a huge part of) the metalanguage
in which we express the discipline.

Forme is substance

- The way we express a concept

  an algorithm, a protocol, a software architecture, . . .

  is co-essential to that very concept.

- The essence of our discipline lays in the immaterial linguistic expression of computation and interaction

- And, of course, there is never a fully faithful translation between one such expression and another. . .

  [cfr. e.g. George Steiner, *After Babel*,1998^2]

“Programming” languages

No scientific discipline exists without first inventing a visual and written language which allows it to break with its confusing past.

[B. Latour, Visualisation and Cognition: Thinking with Eyes and Hands; 1986]

Referring to Dagognet, F.: Tableaux et Langages de la Chimie. Paris : Le Seuil 1969;
and to: Ecriture et Iconographie. Paris : Vrin 1973.

What we call programming languages are both such a founding language and the very object of the discipline.

“Programming” languages

No scientific discipline exists without first inventing a visual and written language which allows it to break with its confusing past.

[B. Latour, Visualisation and Cognition: Thinking with Eyes and Hands; 1986]

Referring to Dagognet, F.: Tableaux et Langages de la Chimie. Paris : Le Seuil 1969;
and to: Ecriture et Iconographie. Paris : Vrin 1973.

What we call programming languages are both such a founding language and the very object of the discipline.

Let us follow Latour. . .

"Programs" are:

- mobile
- immutable when they move
- flat
- "their scale may be changed at will":
  phenomena can be dominated with the eyes and held by hands
- reproduced and communicated at little cost
- may be reshuffled and recombined
- may be made part of a written text
- they merge with geometry (they are a faithful model of reality)

They are inscriptions, like geographical maps, or diagrams.

More: programming languages are a formal, general language of (and for) inscriptions.

Let us follow Latour. . .

“Programs” are:

- mobile
- immutable when they move
- flat
- “their scale may be changed at will”:
  phenomena can be dominated with the eyes and held by hands
- reproduced and communicated at little cost
- may be reshuffled and recombined
- may be made part of a written text
- they merge with geometry (they are a faithful model of reality)

They are inscriptions, like geographical maps, or diagrams.

More: programming languages are a formal, general language of (and for) inscriptions.

Let us follow Latour. . .

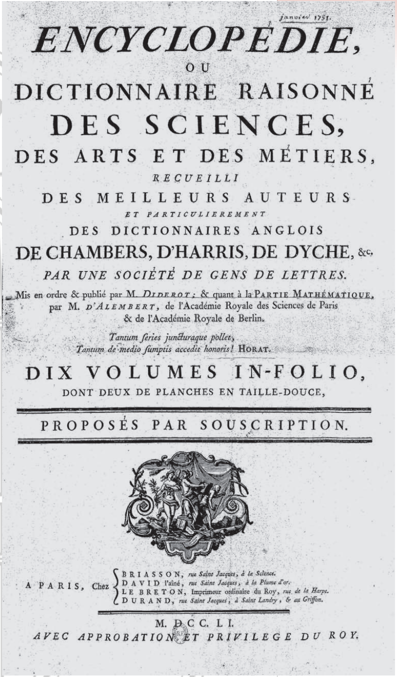
"Programs" are:

- mobile
- immutable when they move
- flat
- "their scale may be changed at will":
  phenomena can be dominated with the eyes and held by hands
- reproduced and communicated at little cost
- may be reshuffled and recombined
- may be made part of a written text
- they merge with geometry (they are a faithful model of reality)

They are inscriptions, like geographical maps, or diagrams.

More: programming languages are a formal, general language of (and for) inscriptions.

Programming languages, once integrated in human languages, become an important piece of that "languaging" (Maturana) which forms the interaction between us, and among us and the world.



janvier 1751.

ENCYCLOPÉDIE,

OU

DICTIONNAIRE RAISONNÉ DES SCIENCES, DES ARTS ET DES MÉTIERS,

RECUEILLI

DES MEILLEURS AUTEURS

ET PARTICULIEREMENT

DES DICTIONNAIRES ANGLOIS DE CHAMBERS, D'HARRIS, DE DYCHE, &c.

PAR UNE SOCIÉTÉ DE GENS DE LETTRES.

Mis en ordre & publié par M. *DIDEROT*; & quant à la PARTIE MATHÉMATIQUE,
par M. *D'ALEMBERT*, de l'Académie Royale des Sciences de Paris
& de l'Académie Royale de Berlin.

Tantum ſeries juncturaque pollet,
Tantum de medio ſumptis accedit honoris! HORAT.

DIX VOLUMES IN-FOLIO,

DONT DEUX DE PLANCHES EN TAILLE-DOUCE,

PROPOSÉS PAR SOUSCRIPTION.

A PARIS, Chez
BRIASSON, *rue Saint Jacques, à la Science.*
DAVID l'aîné, *rue Saint Jacques, à la Plume d'or.*
LE BRETON, Imprimeur ordinaire du Roy, *rue de la Harpe.*
DURAND, *rue Saint Jacques, à Saint Landry, & au Griffon.*

M. DCC. LI.

AVEC APPROBATION ET PRIVILEGE DU ROY.

On s'est adressé aux plus habiles de Paris et du royaume. On s'est donné la peine d'aller dans leurs ateliers [...]
À peine, entre mille, en trouve-t-on une douzaine en état de s'exprimer avec quelque clarté sur les instruments qu'ils emploient et sur les ouvrages qu'ils fabriquent.

[D. Diderot, Prospectus à l'Encyclopédie, 141; 1751.]

We asked the most skilled in Paris and in the kingdom. We even went into their workshops [...]
Among a thousand one will be lucky to find a dozen who are capable of explaining the tools or machinery they use, and the things they produce with any clarity.

[D. Diderot, Prospectus à l'Encyclopédie, 141; 1751.]

PL are a radically new way of saying things (and saying them clearly).
They provide that missing language for saying things in several areas of the human experience.

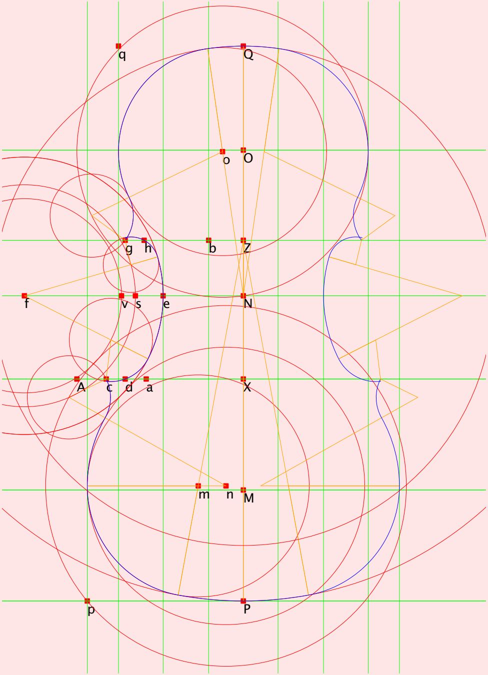
> *Inarticulate does not mean stupid; indeed, what we can say in words may be more limited than what we can do with things. [...] Here is a, perhaps the, fundamental human limit: language is not an adequate "mirror-tool" for the physical movements of the human body.*
>
> *[R. Sennett, The Craftsman. 2009]*

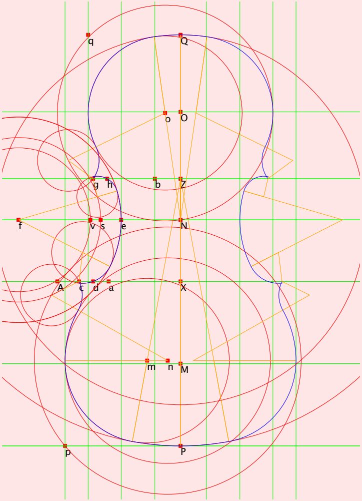
The example of Stradivari's skills and technique.

> *Inarticulate does not mean stupid; indeed, what we can say in words may be more limited than what we can do with things. [...] Here is a, perhaps the, fundamental human limit: language is not an adequate "mirror-tool" for the physical movements of the human body.*
>
> *[R. Sennett, The Craftsman. 2009]*

The example of Stradivari's skills and technique.

> *Programming languages provide a way for us to describe to each other what we know how to do.* *[. . .]*
>
> [They are] *intellectual organizing principle*[s] *for understanding and describing the past, and making sense of the kinds of expertise that flourished and came to maturity.*
>
> *[H. Mairson, Functional Geometry and the Traité de Lutherie. ICFP 2013]*


q
Q
o
O
g
h
b
Z
f
v
s
e
N
A
c
d
a
X
m
n
M
p
P



Appendix: Violin by Andrea Amati

```
(define Amati
  (let ((xq 400)) ;; should be 208mm in the Amati---this is just a screen fit...

; LAYOUT OF THE AREA on which the curves are drawn...
(let ((X (label "X" (point 0 000)))) ; this could be anywhere---just to center it on the output screen
  (let ((A (label "A" (xshift X (- (/ xq 2)))))
        (Q (label "Q" (yshift X xq))))
    (let ((N (label "N" (pointfrom X Q (/ 1 4)))))
      (let ((q (label "q" (xshift (intersect (horizontal Q) (vertical A))
                                  (/ (distance X N) 2))))
            (vv (xshift A (/ (distance X N) 8)))
            (O (label "O" (yshift Q (- (* (distance X N) (/ 5 4))))))
            (Z (label "Z" (yshift N (* (distance X N) (/ 2 3)))))
            (P (label "P" (yshift X (- (* (distance X N) (/ 8 3)))))))
        (let ((p (label "p" (intersect (horizontal P) (vertical vv))))
              (M (label "M" (pointfrom X P (/ 1 2))))
              (a (label "a" (xshift A (/ (distance X Z) 2)))))
          (let ((b (label "b" (xshift Z (- (/ (distance A a) 2))))))
           (let ((ee (label "e" (xshift (intersect (vertical b) (horizontal N)) (- (* (xdistance b p) (/ 3 8)))))))
             (let ((c (label "c" (xshift (intersect (vertical p) (horizontal X)) (/ (xdistance ee p) 4))))
                   (d (label "d" (xshift (intersect (vertical p) (horizontal X)) (/ (xdistance ee p) 2))))
                   (h (label "h" (xshift (intersect (vertical ee) (horizontal Z)) (- (/ (xdistance ee p) 4)))))
                   (g (label "g" (xshift (intersect (vertical ee) (horizontal Z)) (- (/ (xdistance ee p) 2))))))
                      (list X A Q N q O Z P p M a b ee c d h g
                            (horizontal N) (horizontal O) (horizontal Z) (horizontal P) (horizontal X) (horizontal M)
                            (vertical p) (vertical q) (vertical b) (vertical ee)

; THE LOWER BOUTS...
(let ((ZMcircle (circle Z (distance Z M)))
      (ZPcircle (circle Z (distance Z P))))
  (let ((m (label "m" (bottom (intersect ZMcircle
                                         (make-line 1 p) ; line w/slope 1 through p
                                         )))))
    (let ((mcircle (circle m (distance M P))))
      (let ((n (label "n" (xshift m (- (distance X Z) (distance M P))))))
        (let ((ncircle (circle n (+ (distance M P) (distance X Z) (- (distance M P)))))
              (reverse-lower-left
                (lower-circle (reverse-curve (circle n (distance X Z)) (+ (distance X Z) (/ (distance X N) 2)) c))))
                   (list m n (circle n (distance n (center reverse-lower-left)))
                         ZPcircle mcircle ncircle reverse-lower-left
                         (make-curve P c (list ZPcircle mcircle ncircle reverse-lower-left)) ))))))

; THE UPPER BOUTS...
(let ((Ncircle (circle N (distance N Q)))
      (o (label "o" (top (intersect
                          (circle N (distance N O))
                          (make-line -1 q) ; line w/slope -1 through q
                          )))))
   (let ((ocircle (circle o (distance O Q))))
     (let ((reverse-upper-left
           (upper-circle (reverse-curve ocircle
                                        (distance N O)
                                        g))))
              (list o (circle o (distance o (center reverse-upper-left)))
                   Ncircle ocircle reverse-upper-left
                   (make-curve Q g (list Ncircle ocircle reverse-upper-left)) ))))

; THE MIDDLE BOUTS...
(let ((f (label "f" (xshift ee (- (distance X Z)))))
      (v (label "v" (xshift ee (- (/ (distance X N) 2)))))
      (s (label "s" (xshift ee (- (/ (distance N Z) 2))))))
  (let ((ecircle (circle f (distance f ee)))
        (vcircle (circle f (distance f v)))
        (scircle (circle f (distance f s))))
    (let ((reverse-lower-middle
            (upper-circle (reverse-curve ecircle (distance f v) d)))
          (reverse-upper-middle
            (lower-circle (reverse-curve ecircle (distance f s) h))))
              (list f v s (circle f (distance f ee)) reverse-upper-middle reverse-lower-middle
                (make-curve g c (list reverse-upper-middle (circle f (distance f ee)) reverse-lower-middle)) ))))))))))))))
```

Hybris?

Galileo, on the 450-th anniversary of his birth:

- “The book [of the universe] is written in mathematical language,
- and the symbols are triangles, circles and other geometrical figures”

- But also numbers, effective procedures and abstractions.

- The descriptions co-exist and complement each other
- in the fruitful plurality of languages and descriptions

Hybris?

Galileo, on the 450-th anniversary of his birth:

- “The book [of the universe] is written in mathematical language,
- and the symbols are triangles, circles and other geometrical figures”

- But also numbers, effective procedures and abstractions.

- The descriptions co-exist and complement each other
- in the fruitful plurality of languages and descriptions