Seminari Simmetria, Informatica
Milano – Bicocca

Simmetria e ortogonalità nei linguaggi di programmazione

Claudio Ferretti

Università degli Studi di Milano – Bicocca
Dip. di Informatica, Sistemistica e Comunicazione (DISCo)
Viale Sarca 336/14 – Milano - Italy

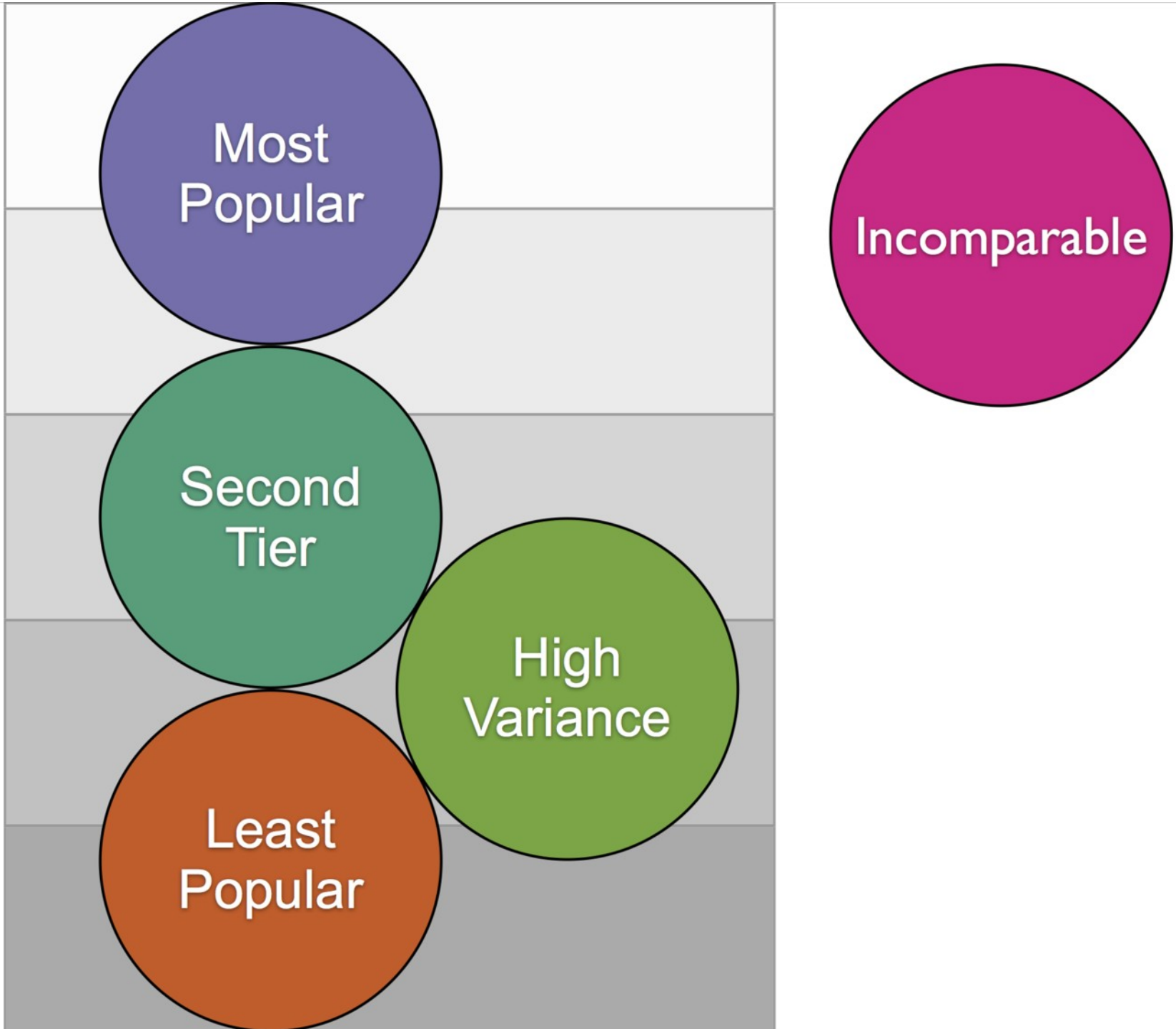
Perché il titolo

- Elementi su cui rifletteremo:
  - Esistono molti (e diversi!) linguaggi
  - Li possiamo valutare sotto diversi aspetti
  - Due aspetti sono: simmetria e ortogonalità
  - Le caratteristiche portano a vantaggi e svantaggi per ogni linguaggio
  - ...vediamo esempi di linguaggi (anche a livello macchina)

Criteri di valutazione dei linguaggi di programmazione

- Elementi di valutazione dei linguaggi:
  - Espressività
  - Sinteticità
  - Produttività
  - Comprensibilità in lettura
  - altro... (esigenze di contesto: adozione, dominio applicativo, ...)


actionscript
asp
assembly
c
c#
c++
coffeescript
haskell
java
javascript
objective-c
perl
php
python
r
ruby
scala
shell
arduino
clojure
coldfusion
d
delphi
emacs lisp
erlang
f#
fortran
go
groovy
lua
matlab
ocaml
powershell
prolog
puppet
racket
scheme
tcl
apex
augeas
bro
ceylon
dcpu-16 asm
dylan
ec
ecl
fancy
fantom
ioke
julia
kotlin
logtalk
mirah
nimrod
nu
ooc
parrot
standard ml
turing
ada
applescript
arc
autohotkey
boo
coq
dart
eiffel
elixir
fact
or
haxe
io
max
nemerle
objective-j
opa
rebol
self
smalltalk
vala
verilog
vhdl
xquery
common lisp
gosu
openedge abl
pure data
rust
scilab
supercollider
viml
visual basic


Most Popular
Second Tier
High Variance
Least Popular
Incomparable

Hello, world [edit]

The traditional "Hello, world!" program demonstrates how different INTERCAL is from standard programming languages. In C, it could read as follows:

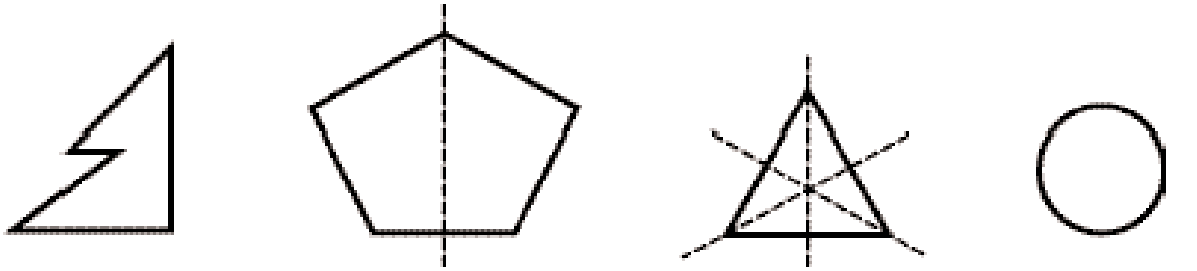
```c
#include <stdio.h>

int main() {
    printf("Hello, world!\n");
}
```

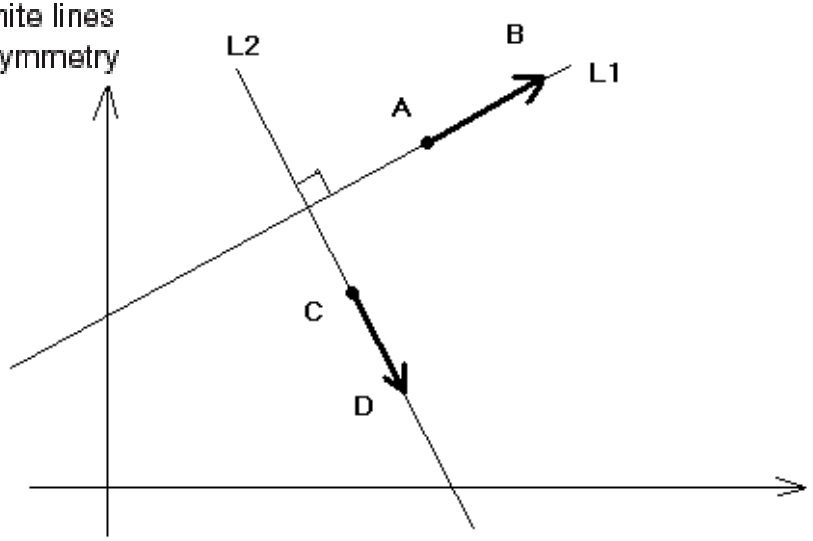
The equivalent program in C-INTERCAL is longer and harder to read:

```
DO ,1 <- #13
PLEASE DO ,1 SUB #1 <- #238
DO ,1 SUB #2 <- #108
DO ,1 SUB #3 <- #112
DO ,1 SUB #4 <- #0
DO ,1 SUB #5 <- #64
DO ,1 SUB #6 <- #194
DO ,1 SUB #7 <- #48
PLEASE DO ,1 SUB #8 <- #22
DO ,1 SUB #9 <- #248
DO ,1 SUB #10 <- #168
DO ,1 SUB #11 <- #24
DO ,1 SUB #12 <- #16
DO ,1 SUB #13 <- #162
PLEASE READ OUT ,1
PLEASE GIVE UP
```

Perché il titolo

- Due parole, Simmetria / Ortogonalità:
  - Usuale accezione (immagini e strutture):

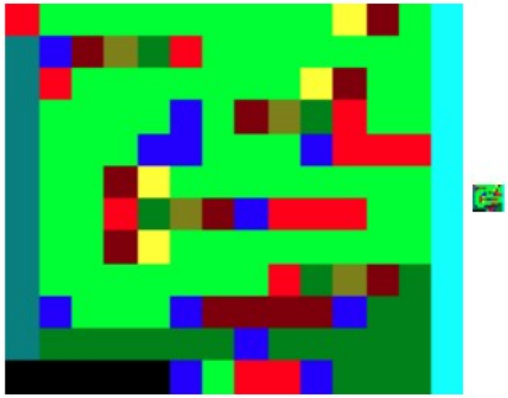




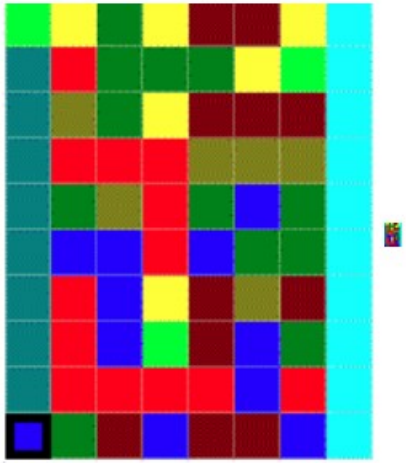
www.analyzemath.com

Geometrie nei linguaggi?

The following is an enlarged Hello, world! program in Brainloller, followed by the actual program:



There is also a shorter one:



Linguaggio Brainloller

Color	RGB	Function
red	(255,0,0)	>
darkred	(128,0,0)	<
green	(0,255,0)	+
darkgreen	(0,128,0)	-
blue	(0,0,255)	.
darkblue	(0,0,128)	,
yellow	(255,255,0)	[
darkyellow	(128,128,0)	]
cyan	(0,255,255)	rotates the IP 90° clockwise
darkcyan	(0,128,128)	rotates the IP 90° counter-clockwise
others	n/a	nop

At execution start, everything is 0, the IP (instruction pointer) is at the top left, and moves to the east.

Linguaggi più…familiari

‘Hello world’

- (print "Hello world!") / print "Hello world!"
- int main()

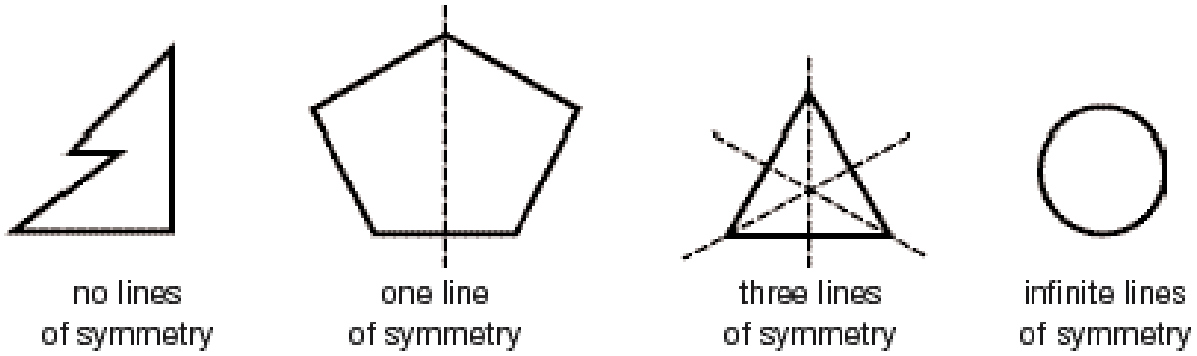
```
{
    return printf("\nHello World!");
}
```

- public class HelloWorld

```
{
 public static void main(String[] args)
 {
  System.out.println("Hello world!");
 }
}
```

Simmetria

- Simmetria nelle figure:
  - Le figure non cambiano sotto alcune trasformazioni:



Nozione di simmetria nei linguaggi

- Da Internet:
  - “symmetry gives a set of possible transformations that do not change a given property”
  - e.g. Geometry: translation; rotation or reflection (some figures)
  - Programming: Moving a call of a pure function to a different time, or task, does not change its result

    Reflexive/symmetric/associative operators and functions

Un caso semplice: 'specchio' in Ruby

- Scelta dei progettisti di Ruby:
  - Inserire l'operatore '+ unario' come specchio del più usuale '-'

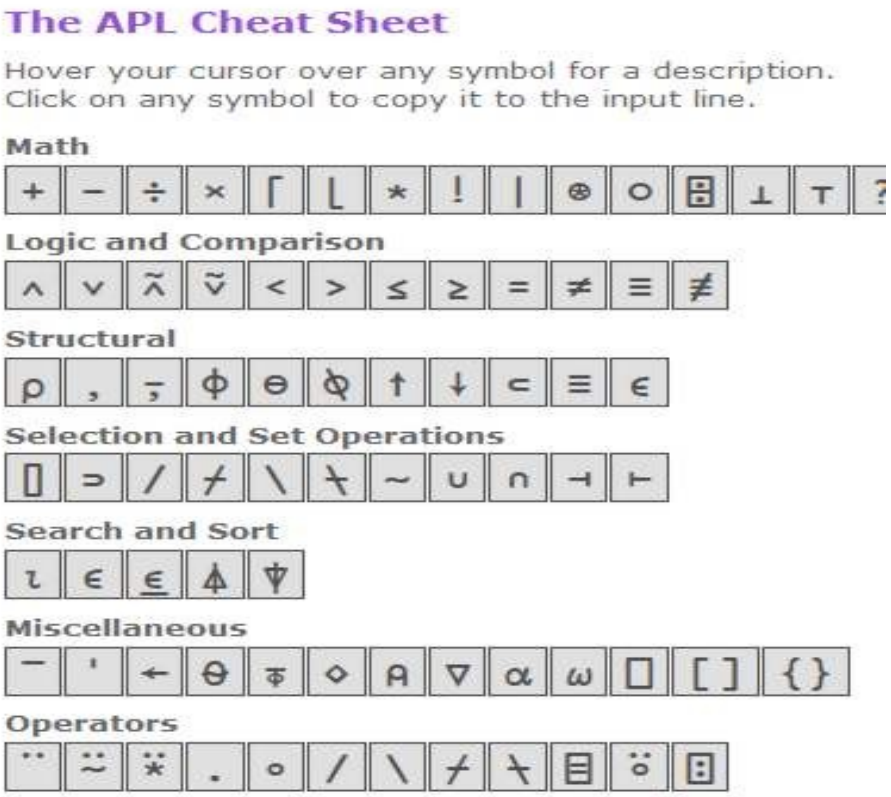
4.6.1 Unary + and –

The unary minus operator changes the sign of its numeric argument. The unary plus is allowed, but it has no effect on numeric operands—it simply returns the value of its operand. It is provided for symmetry with unary minus, and can, of course, be redefined. Note that unary minus has slightly lower precedence than unary plus; this is described in the next section on the ** operator.

Un caso antico: APL

- Non facilmente leggibile...!

```
      ∇DET[⎕]∇
     ∇ Z←DET A;B;P;I
[1]    I←⎕IO
[2]    Z←1
[3]   L:P←(|A[;I])⍳⌈/|A[;I]
[4]    →(P=I)/LL
[5]    A[I,P;]←A[P,I;]
[6]    Z←-Z
[7]   LL:Z←Z×B←A[I;I]
[8]    →(0 1 ∨.=Z,1↑⍴A)/0
[9]    A←1 1 ↓A-(A[;I]÷B)∘.×A[I;]
[10]   →L
[11]  ⍝EVALUATES A DETERMINANT
      ∇
```



Il linguaggio LISP

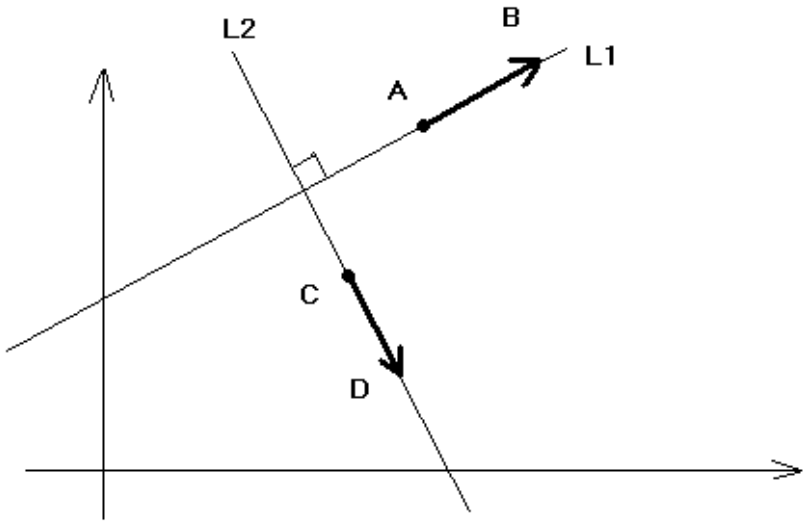
- Un'altra definizione:
  - "symmetry: the features of a programming language operate equally on all of the things in the programming language":

    In LISP: "a pervasive symmetry between programs and data"
  - `(setq x 25) (print x)`
  - `(defmacro setTo10(num)`

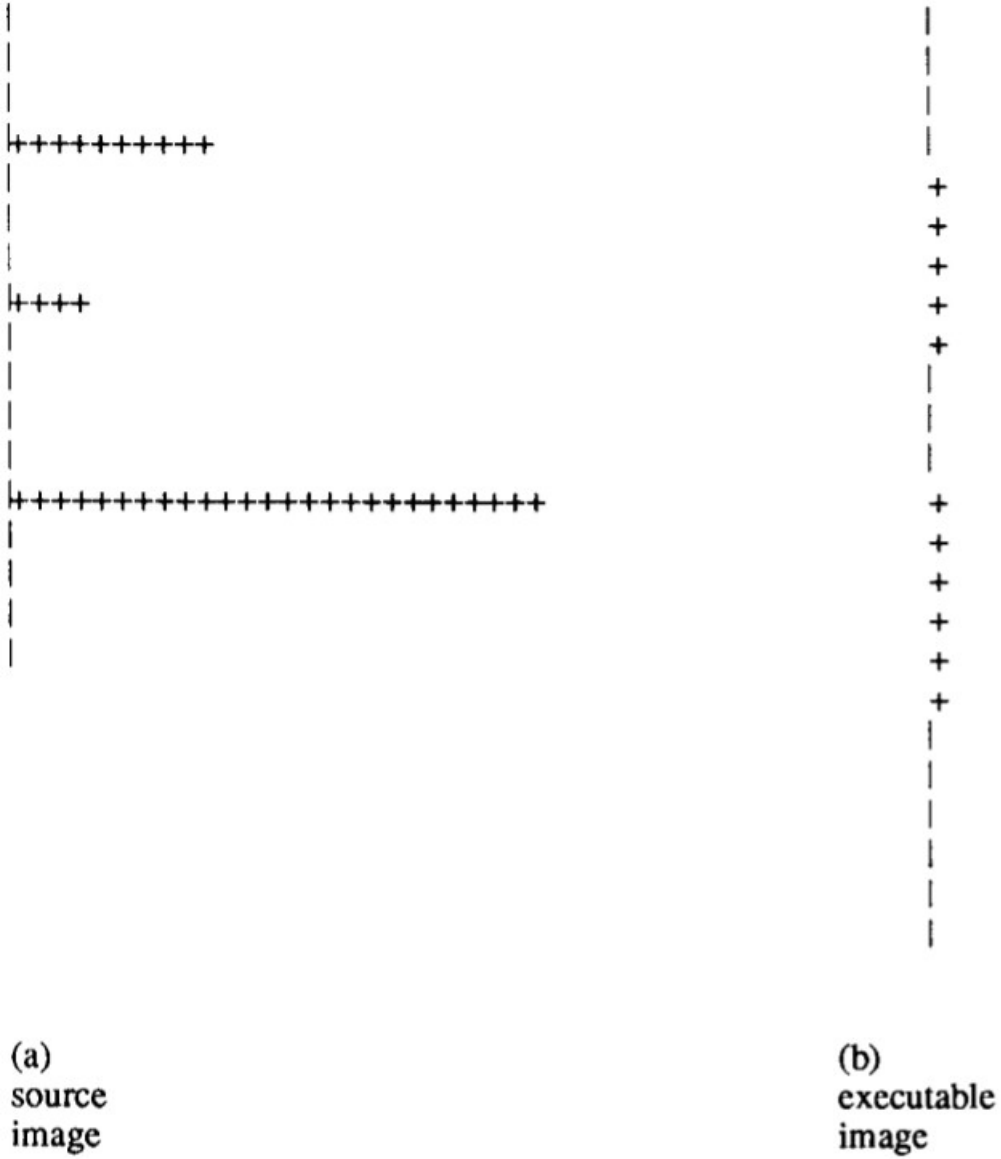
    `(setq num 10)(print num) )`
  - `(setTo10 x)`

Ortogonalità

- Ortogonalità nelle figure:
  - Indivudo 2 ‘dimensioni’, che posso percorrere separatamente:



Research...



(a)
source
image

(b)
executable
image

Figure 2-6

..nel 1990

Integrating Tools and Support-Code with the Developed Program

Shmuel Rotenstreich

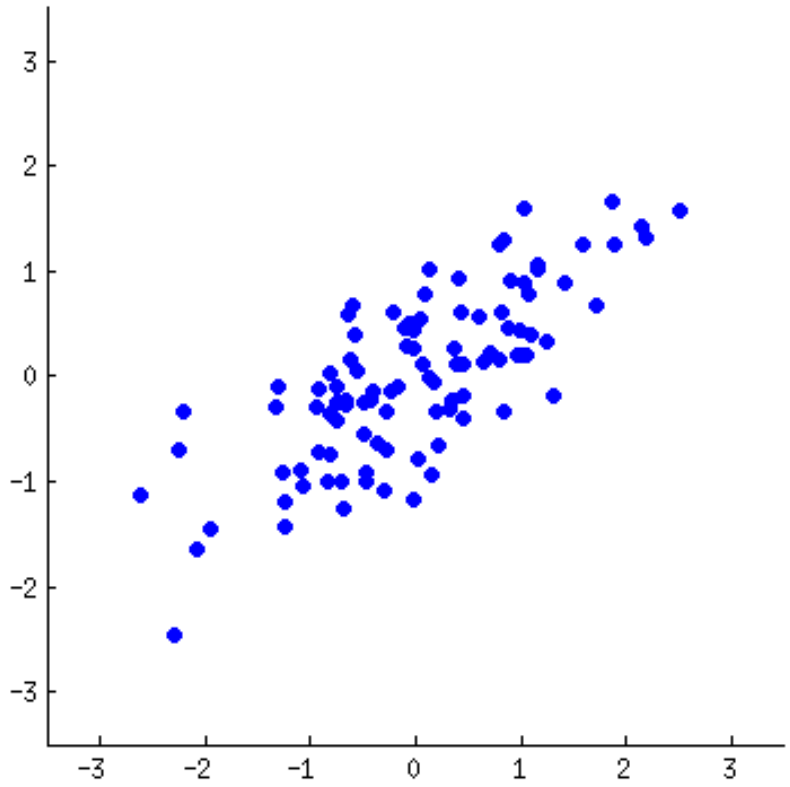
Dept. of EE&CS
The George Washington University

We suggest a framework that combines the developed program with its support-code. The physical connection is of the hypertext type[12]. In addition to the physical connection, the developed program and its support-code are combined syntactically and semantically by a programming language called an *Orthogonal Language*. that is an extension of the language in which the developed program is written. This paper discusses Orthogonal-C which is an orthogonal extension of C. Although,

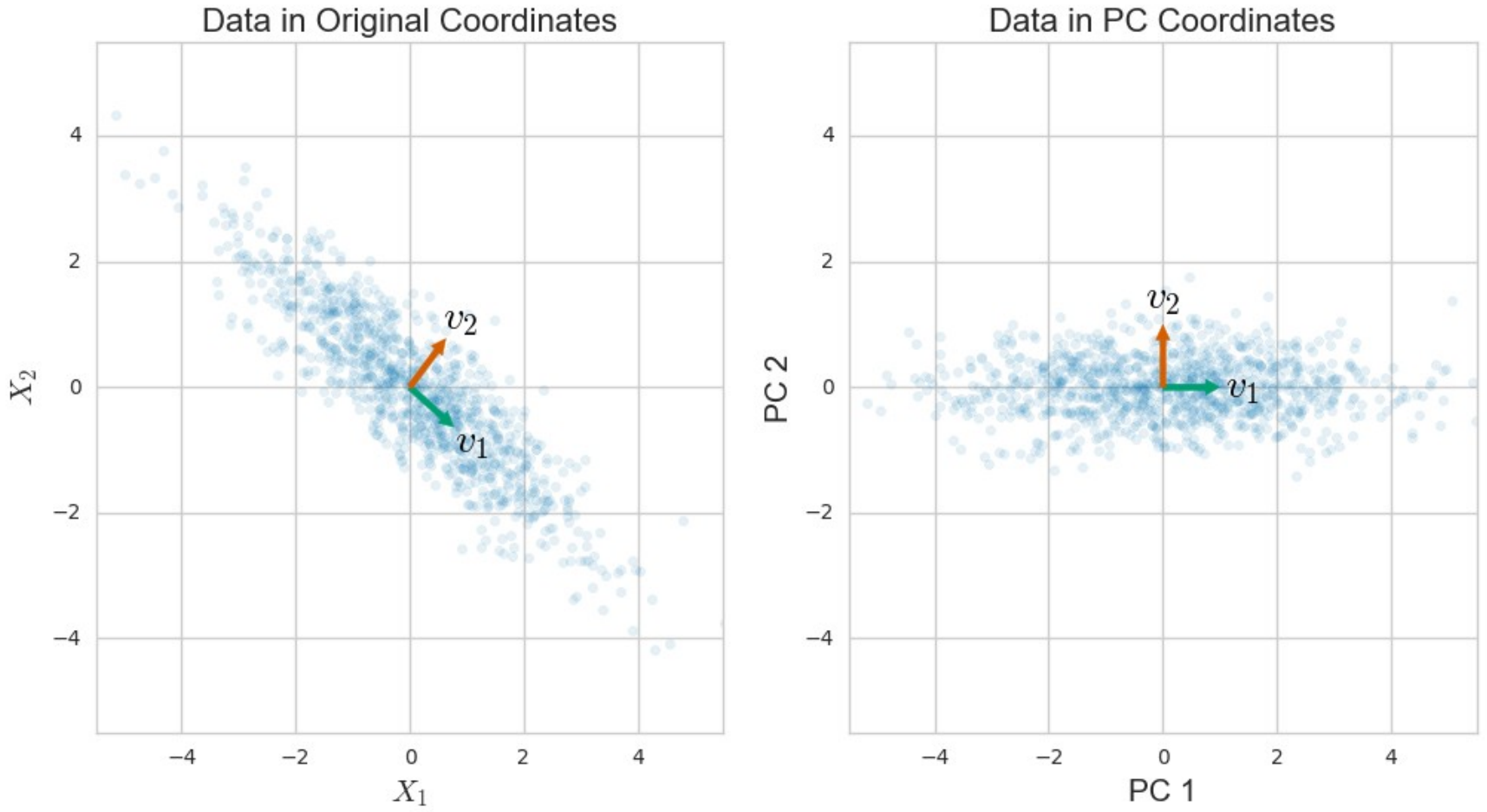
```
1    anexample()
2    {
3      int x,y;
4      x=2;
5      ....
           printf("trace at line 6\n");
           printf(" x=%f,y=%f ",x,y);
           if(x+y>100) printf("should not get here\n");
6      y=x+100;
7      ....
8    }
```

Figure 2-3

Quando l'ortogonalità può aiutare



Dati trasformati in ‘ortogonali’



Perché il titolo

- Da Internet:
  - Orthogonality is the property that means "Changing A does not change B". An example of an orthogonal system would be a radio, where changing the station does not change the volume and vice-versa.
  - A non-orthogonal system would be like a helicopter where changing the speed can change the direction.
  - From Eric S. Raymond's "Art of UNIX programming":

    Orthogonality is one of the most important properties that can help make even complex designs compact. In a purely orthogonal design, operations do not have side effects; each action (whether it's an API call, a macro invocation, or a language operation) changes just one thing without affecting others. There is one and only one way to change each property of whatever system you are controlling.

A basso livello: ortogonalità in linguaggio macchina/assembly

- Somma in CPU IBM, due forme:

- A Reg1, memory_cell
- AR Reg1, Reg2

- Somma in CPU VAX, forma unica:

- ADDL operand1, operand2

- "A relatively small set of primitive constructs can be combined in a relatively small number of ways to build the control and data structures of the language.

  It is associated with simplicity; the more orthogonal the design, the fewer exceptions."

Una CPU più 'ortogonale': MC68000

General format for MC68000 instructions.

```
Instruction format is:
        mnemonic.size    source,destination
Examples:
        ADD.L D1,D2
        MOVE.B #15,-1(A0)
        ADD D1,D2            (size assumed to be ".W")
        BGE LOOP1            (only one argument)
        RTS                  (no arguments)
Explanations:
        mnemonic = instruction abbreviation (ADD, CMP, MULS, etc.)
        size (optional) = operand size:
                .B means byte data (8 bits)
                .W means word data (16 bits; default size)
                .L means long word data (32 bits)
        source (optional) = source operand addressing mode
        destination (optional) = destination operand addressing mode
```

Per confronto: CPU 8086, e succ.

- Operazione somma:
  - Elenco le opssibili combinazioni per la scelta degli operandi:

ADD	REG, memory
	memory, REG
	REG, REG
	memory, immediate
	REG, immediate
		Add.
		
		Algorithm:
		
		operand1 = operand1 + operand2
		
		Example:
		
		MOV AL, 5 ; AL = 5
		ADD AL, -3 ; AL = 2
		RET

Un confronto più esteso:

Addressing modes available to the Motorola MC68000 (“M”) and the Intel 8086 (“I”).

The information at the intersection of a row and a column indicates the availability of that source/destination addressing-mode combination

	destination	Dn	An	(An)	(An)+	-(An)	d16(An)	d8(An,Xn)	Abs.W	Abs.L
	#options	8	8	8	8	8	8	128	N/A	N/A
source	#options									
Dn	8	MI	MI	MI	M	M	MI	MI	MI	M
An	8	MI	MI	MI	M	M	MI	MI	MI	M
(An)	8	MI	MI	MI	M	M	M	M	M	M
(An)+	8	M	M	M	M	M	M	M	M	M
-(An)	8	M	M	M	M	M	M	M	M	M
d16(An)	8	MI	MI	M	M	M	M	M	M	M
d8(An,Xn	128	MI	MI	M	M	M	M	M	M	M
Abs.W	N/A	MI	MI	M	M	M	M	M	M	M
Abs.L	N/A	M	M	M	M	M	M	M	M	M
d16(PC)	1	M	M	M	M	M	M	M	M	M
d8(PC,Xn)	16	M	M	M	M	M	M	M	M	M
Immediate	1	M	M	M	M	M	M	M	M	M

Torniamo a un linguaggio ‘alto’

- Linguaggio SQL, per lavorare su database:
- Ci sono due elementi sintattici diversi per la stessa operazione:
  filtro con WHERE, o filtro con HAVING ?
  - `select invoice,count(*) as cnt from invoice where invoice='x'`
    `group by invoice having cnt>1`

Quindi?

- Ogni linguaggio ci ‘costringe’ a lavorare mentalmente in modi specifici
- Ciascuno di noi ‘funziona meglio’ in uno o nell’altro linguaggio
- Ci sono esigenze a volte contrastanti:
  produrre (scrivere) / mantenere (leggere) / avere prestazioni (esguire) ...
- Simmetria: rende flessibile esprimere (scrivere) molte strutture
- Ortogonalità: rende veloce l’apprendimento (e in parte la lettura)

Ultima+1 slide...

Grazie per
l'attenzione !