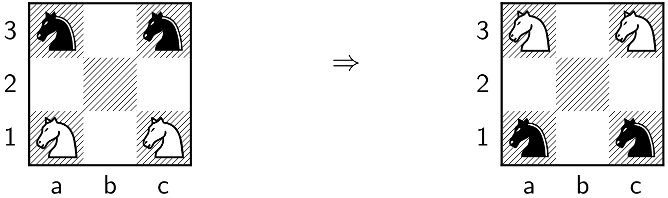
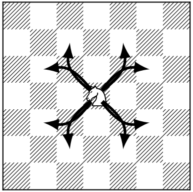
Fundamentals of Computing

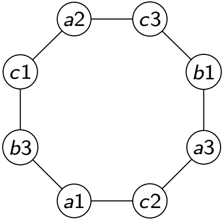
Gianluca Rossi

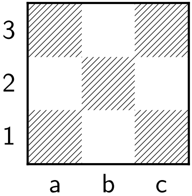
gianluca.rossi@uniroma2.eu

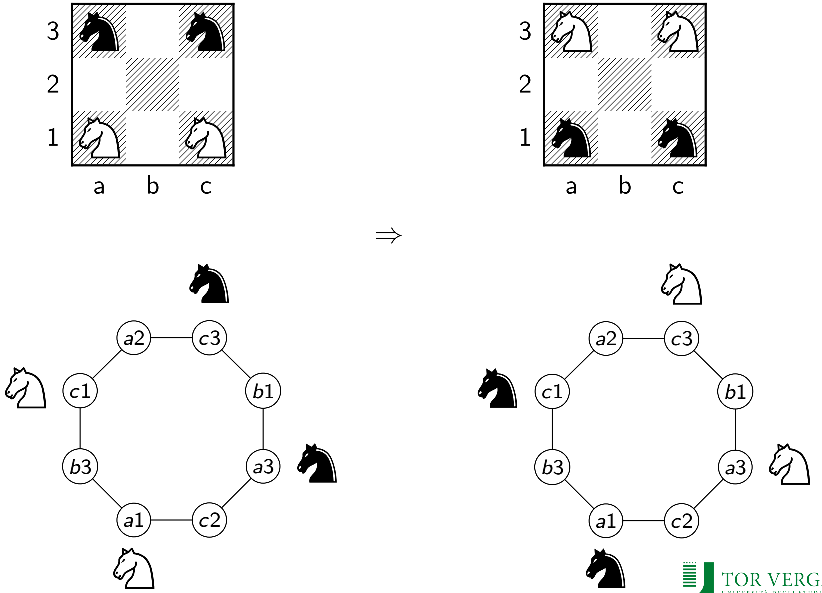
TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

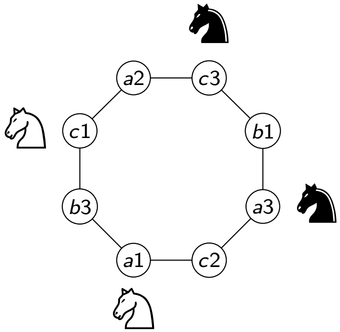

3
2
1
a
b
c
⇒
3
2
1
a
b
c

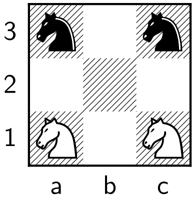


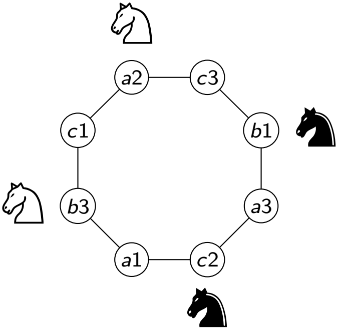

a2
c3
c1
b1
b3
a3
a1
c2


3
2
1
a
b
c

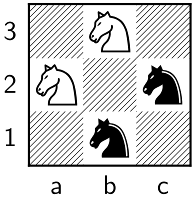

3
2
1
a
b
c
⇒
a2
c3
c1
b1
b3
a3
a1
c2


a2
c3
c1
b1
b3
a3
a1
c2

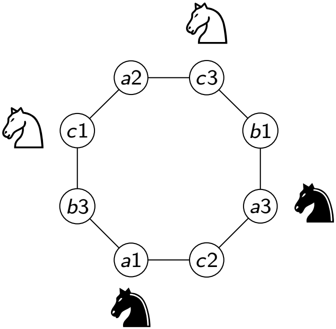

3
2
1
a
b
c

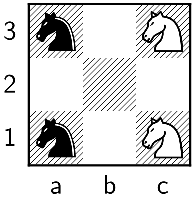


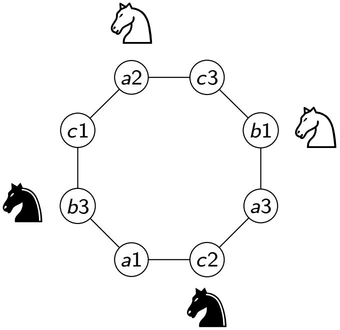
a2
c3
c1
b1
b3
a3
a1
c2



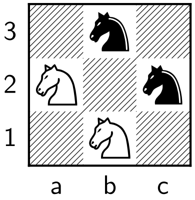
3
2
1
a
b
c


a2
c3
c1
b1
b3
a3
a1
c2

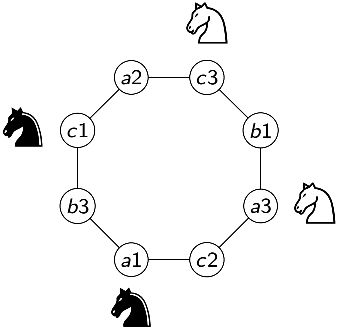

3
2
1
a
b
c

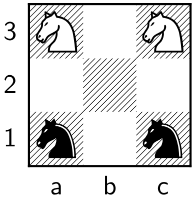


a2
c3
c1
b1
b3
a3
a1
c2



3
2
1
a
b
c


a2
c3
c1
b1
b3
a3
a1
c2


3
2
1
a
b
c

Algorithmic method

1. Problem definition: a stakeholder poses a problem (mathematics, finance, administration, meteorology, betting,...);
2. Find a good mathematical model for the problem;
3. Find a recipe (the algorithm);
4. Code the recipe in a computer programming language:
5. Test the program against syntactical errors (eventually go back 4);
6. Test solutions against semantic errors (eventually go back 2);

A numerical example

Problem: Find $\sqrt{x}$ that is y such that $y^2 = x$

Model: Let $y_0 = g$,

$$\sqrt{x} \approx y_i = \frac{1}{2}\left(y_{i-1} + \frac{x}{y_{i-1}}\right)$$

and $|\sqrt{x} - y_{i+1}| \leq |\sqrt{x} - y_i|$

Algorithm:

1. Guess a value g;
2. If g^2 is "close" to x stop;
3. Update g with

$$\frac{1}{2}\left(g + \frac{x}{g}\right)$$

4. Repeat from 2

A numerical example

Let $x = 20$

g	g^2	$0.5 \cdot (g + x/g)$
5.0	25.0	4.5
4.5	20.25	4.472
4.472	20.0007	4.4721
4.4721	20.000000007	

What is an algorithm?

- A sequence of simple actions (elementary instructions);
- A flow control mechanism to determine the next instruction;
- Stop conditions.

From algorithm to program

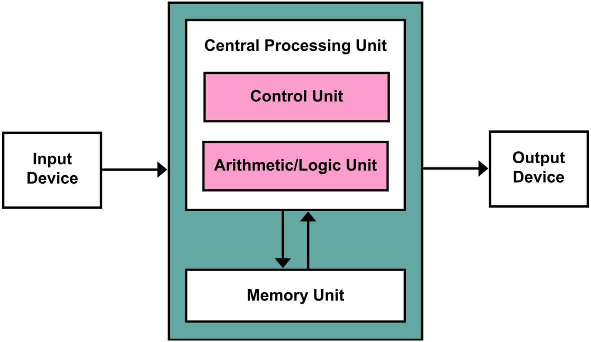
```
x = 20
g = 5.0

while abs(g*g - x) > 0.00001 :
    g = 0.5*(g+x/g)

print(g)
```

- arithmetic and logic instructions;
- test (conditionals) instructions;
- storing instruction

...executed by a special program (interpreter)




```
x = 20
g = 5.0

while abs(g*g - x) > 0.00001 :
    g = 0.5*(g+x/g)

print(g)
```

Binary encoding

Data and instructions programs are stored in memory using an alphabet of two symbols, 0 and 1 that represent two different electrical states.

Integers

Positional notation

$$\begin{aligned}(11110110100)_2 &= 1 \cdot 2^{10} + 1 \cdot 2^9 + 1 \cdot 2^8 + 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + \\ & \quad 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 \\ &= 1024 + 512 + 256 + 128 + 32 + 16 + 4 = (1972)_{10}.\end{aligned}$$

Binary encoding

Conversion $(N)_{10} \to N_2$

- If $N \in \{0,1\}$, B is N
- Assume $b'_k b'_{k-1} \dots b'_0$ encodes $\lfloor N/2 \rfloor$

$$
\begin{aligned}
N &= 2\lfloor N/2 \rfloor + b_0 \\
&= 2\left(b'_k \times 2^k + \ldots + b'_0 \times 2^0\right) + b_0 \\
&= b'_k \times 2^{k+1} + \ldots + b'_0 \times 2^1 + b_0 \times 2^0.
\end{aligned}
$$

then $b'_k b'_{k-1} \dots b'_0 b_0$ encodes N.

Binary encoding

```python
n = 16

finish = False

while not finish:
    if n == 0 or n == 1:
        print(n)
        finish = True
    else:
        b = n%2
        print(b)
        n = n//2
```

Binary encoding

- **Negative integers**: one of the bits will be used for the sign
- **Rational and floating point notation**:

$$x = 0.m \cdot 2^e$$

m and e (*mantissa* and *exponent*) are signed integers

Encoding of x

- 24 bits for m
- 8 bits for e

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Binary encoding

Text and ASCII code

Cod.	Car	Cod.	Car	Cod.	Car	Cod.	Car	Cod.	Car
33	!	52	4	71	G	90	Z	109	m
34	”	53	5	72	H	91	[	110	n
35	#	54	6	73	I	92	\	111	o
36	$	55	7	74	J	93	]	112	p
37	%	56	8	75	K	94	ˆ	113	q
38	&	57	9	76	L	95	_	114	r
39	’	58	:	77	M	96	‘	115	s
40	(	59	;	78	N	97	a	116	t
41	)	60	¡	79	O	98	b	117	u
42	*	61	=	80	P	99	c	118	v
43	+	62	¿	81	Q	100	d	119	w
44	,	63	?	82	R	101	e	120	x
45	-	64	@	83	S	102	f	121	y
46	.	65	A	84	T	103	g	122	z
47	/	66	B	85	U	104	h	123	{
48	0	67	C	86	V	105	i	124	—
49	1	68	D	87	W	106	j	125	}
50	2	69	E	88	X	107	k	126	∼
51	3	70	F	89	Y	108	l		

Binary encoding

Program instructions and machine language

- A fixed number of bits for the *operation code* or **opcode**
- The remaining bits will encode the operands.

100110	*operand1*	*operand2*

High level programming languages

- No dependency from the architectures
- Formalism that is human-readable
- Introduce high level of abstraction
- Need a tool thats cover the gap with the architecture

High level programming languages

Compiled languages

The program is translated, by a *compiler*, in a machine language program equivalent to the original.

Interpreted languages

The program is executed line by line by an *interpreter* at runtime.

The compiler and the interpreter are two program that can be executed on the current architecture.

Programming languages

Symbols: $+$, $=$, $*$,..., `while`, `print`,...

Syntax: Rules that describe how combine symbols to get instruction and programs

Semantic: Meaning of symbols, instructions, and programs

The limits of computational method

Efficiency

- The expected running time of a program must be *adequate* to the application.
- Sometimes, the expected running time exponentially grows with the size of the input.
- The exponential growth is not compatible with the performance improvements between generations of computer architectures.

Incompleteness

There are problems that cannot be solved by a computer program.

The limits of computational method

Programs can be encoded with sequences of binary symbols so, like other data, can be input of other programs.

The halting problem

Given a description of a computer program and an input, determining whether the program will finish running, or continue to run forever.

Theorem

No general algorithm exists that solves the halting problem for all possible program–input pairs.

Prof

Assume, by contradiction, that there exists the program `halt(P,x)` that answers `True` if and only if `P(x)` terminates. Then there also exists

```
Paradox(P):
    if halt(P, P) is True:
        run forever
    else:
        print('done!')
```

`Paradox(Paradox)` ends iff `halt(Paradox, Paradox)` is `False` iff `Paradox(Paradox)` never ends. Contradiction, so `halt` cannot exist.