# EXPLAINING COMPUTING SYSTEMS BEHAVIOURS
## Functions, Abstractions, Idealizations

NICOLA ANGIUS [1]

(Work in conjunction with Guglielmo Tamburrini)

[1]Department of History, Human Sciences, and Education.
University of Sassari, Italy
nangius@uniss.it

HaPoC 2015, Pisa, October, 11,

## Motivations

### Philosophy of Computer Science

- Explaining computing systems behaviors (CSB) is a pervasive activity in computer science.
- The methodological problem of analyzing what it is to explain CSB has received relatively scarce attention in the philosophy of computer science (missing in SEP).

### Philosophy of Science

Import on models of explanation (functional, causal, mechanist, . . . )

### Philosophy of the cognitive (neuro-)sciences

Import on computational modeling and explanation

## Main aim

Computing systems form a vast class of physical systems.

Restriction to subclasses of the broad class of computing systems, i.e. digital computers running offline a variety of programs written in some high-level programming language.

- A pluralistic account of explanations of CSB is provided, which emphasizes the complementary character of **mechanist** and **functional** explanations
- Resisting the identification of explanations in computer science with causal-mechanistic explanations (Piccinini 2007; Piccinini and Craver 2011).

## Summary

## Program Specifications

Why was an incorrect output observed in this particular run of program P on personal computer C?

The explanatory request presupposes the existence of norms of executions: *specifications*.

- In explaining why some executions of a program P conforms with or deviates from user requirements one is *ipso facto* explaining why certain human goals and intentions are complied with or not in those runs of P.

## What-How-What

User specifications usually express something about **what** is to be accomplished (or to be avoided) whithout saying much about the ways in which this is to be done.

Programmers must choose one among the alternative courses of action that are avilable to fulfil user specifications, in other words, **how** they intend to fulfil the *'what'*.

Layers of functional organization descitpions connected by *how-what* relationships:

- Logic gates and circuits
- microarchitecture
- Instruction Set Architecture (ISA)
- Assembly language instructions
- High level programming language instructions

## Functional and Structural Properties

Computational systems are physical machines that can be defined by two kind of properties (Turner 2014):
*functional* properties
*structural* properties

Functional and structural properties descriptions are settled in the hierarchy of what-how descriptions in terms of specification|implementation *must-prescriptions*.

Explanations of CSB that are incorrect in the light of those must-prescriptions often **abtstract from** any reference to physical components and processes.

## A Machine Computing a Factorial

Requirement:
For any
$n \in \mathbb{N}, \quad 0! = 1; (n+1)! = (n+1) * n!$

Pseudo-Pascal high level code:

```
1  BEGIN
2    read(n);
3    i := 0;
4    f := 1;
5    WHILE i < n DO;
6      BEGIN
7        i = i + 1;
8        f = f * i;
9      END
10   write f;
11 END
```

MIPS assembly language instructions for high level code line 5:

```
5.1    slt $t0, $s1, $s0;
5.2    bne $t0, $zero, L1;
```

Iimplemented by the following six-field code-machine instruction:

| 000000 | 10001 | 10000 | 01000 | 00000 | 101010 |
|---|---|---|---|---|---|
| *op* | *rs* | *rt* | *rd* | *shamt* | *funct* |

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

Logic gates → device level

## Functional Analysis of Miscomputations

Let us suppose that C ouputted a different value instead of $m!$ while $m$ was given as input.

The what-how hierarchy outlined so far supplies one with a set of a *purely functional descriptions* against which one may *look for an explanation* of the incorrect CSB.

Downward movements along this hierarchy correspond to the steps of a *functional decomposition analysis* (Bechtel and Richardson 1993; Cummins 1975): explaining a capacity in terms of sub-capacities and their organization.

## Candidate Explanations

Selecting in the what-how hierarchy the higher-level functional descriptions that are not satisfied by the observed CSB:

- Erroneous conceived specification:
  $0! = 2$
  Provided that the incorrect recursive definition is inherited throughout the lower functional levels, *no extra explanatory work is accrued by mentioning lower level functional properties or the physical components of C running P*.

- Pascal encoding error:
  $WHILE\ i \leq n\ DO$
  Explanations appealing to code errors do not need to mention lower level functional descriptions or taking note of what are the physical components of C.

- . . . . . . . . . . . . . . . . . . . . . . . .

## Mechanist Explanation

Mechanisms: "entities and activities organized such that they are productive of regular changes from start or set-up to finish or termination condition" (Machamer *et al.* 2000).

Mechanisms - Mechanism Schemata - Mechanism Sketches

Craver (2007): "Progress in building mechanist explanations involes movement along the . . . sketch - schema - mechanism axis."

Piccinini and Craver (2011): "decompositional, constitutive explanations gain their explanatory force by describing mechanisms . . . and, coversely, that they lack explanatory force to the extent that they fail to describe mechanisms"

## Abstraction in CSB Explanations

- No explanatory benefit is accrued by supplementing the description of a functional mismatch between some functional requirement R and instruction I with a description of physical processes by means of which I is actually carried out.
- Additional descriptions introduce a surplus of irrelevant functional or structural details to a more parsimonious *explanans* and fail to confer additional explanatory force, or may even jeopardize its intelligibility.

**Abstraction** from negligible functional or structural details is an explanatory virtue in computer science, preventing those movements along the sketch-schema-mechanism axis.

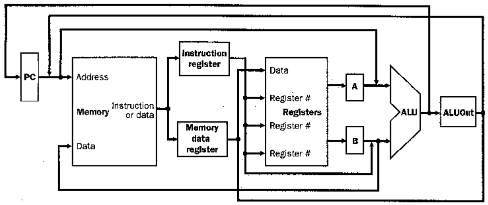
## Explanation in Science and Technology

Explanations of behaviors manifested by technological systems often arise against the background of requirements reflecting human goals.

No reference to human goals and intentions is needed to explain why the volcano Vesuvius erupted and destroyed the town of Pompeii in 79 AD.

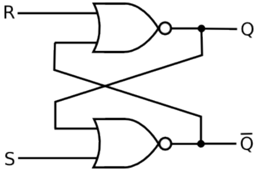
The sense in which user requirements are claimed to be prescriptive differs from the sense in which law-like statements appearing in scientific explanations are often claimed to be prescriptive too.

## Hardware Exceptions



- First clock cycle: Instruction is picked up from memory using the memory address contained in PC; the instruction is saved in IR; PC is incremented.
- Second clock cycle: Fields *rs* and *rt* are read; their values are saved in registers A and B to be used in the next clock cycle.
- Third clock cycle: Fields *op* and *funct* are read; the ALU computes the corresponding mathematical operation; the result is saved in ALUOut.
- Fourth clock cycle: Destination register corresponding to *rd* is used to save the value in ALUOut.

## Flip-Flops and role fillers



Provides a description of functional roles that any physical device (transistors, vaccum tubes, etc . . . ) must satisfy in order to count as a physical role filler for an S-R latch, without including more detailed descriptions of their potential physical role fillers.

Isolating a faulty transistor: this is a causal explanation for violations of the functional S-R latch description insofar as it makes reference to one of its physical role fillers.

One can hardly achieve greater explanatory force over this parsimonious explanation by including causal details about the physical processes carried out by transistors.

## Flip-Flops and role fillers

Physical characteristics of potential role fillers are ignored in functional descriptions of S-R latches, except for requiring that adequate role fillers must possess distinguished states to be conventionally associated to one value from the set $\{0, 1\}$, and functional conditions for state transitions in the same set.

However, the orbits and state space trajectories of transistors and vacuum tubes are usually and more richly described by means of continuous, rather than discrete, macroscopic variables.

The explanations of incorrect CSB neglect every intermediate state that an actual computing system goes through.

## From Turing 1950

The digital computers considered in the last section may be classified amongst the "discrete-state machines". These are the machines which move by sudden jumps or clicks from one quite definite state to another. These states are sufficiently different for the possibility of confusion between them to be ignored. Strictly speaking there are no such machines. Everything really moves continuously. But there are many kinds of machines which can profitably be thought of as being discrete-state machines. For instance in considering the switches for a lighting system it is a convenient fiction that each switch must be definitely on or definitely off. There must be intermediate positions, but for most purposes we can forget about them. As an example of a discrete-state machine we might consider a wheel which clicks round through 120 once a second, but may be stopped by a lever which can be operated from outside; in addition a lamp is to light in one of the positions of the wheel . . . (Turing 1950, 439-440).

## Modeling Reactive Systems

Why this class of machines display the specifed regular behaviour?

*Reactive systems*: systems that interact with their environment and carry out non-terminating computations.

*Formal Methods*: models involve an extensive use of both abstraction and distortive **idealization** (Novack 1979; Cartwright 1989; Weisberg 2007).

- *Abstraction* hides mechanist details of lower descriptions;
- *Distortive idealization* makes the inclusion of those details impossibile.

## Model Checking

Model checking: enables one to verify whether runs of reactive system R satisfy, according to a suitable model M of R, properties that are decidable in M.

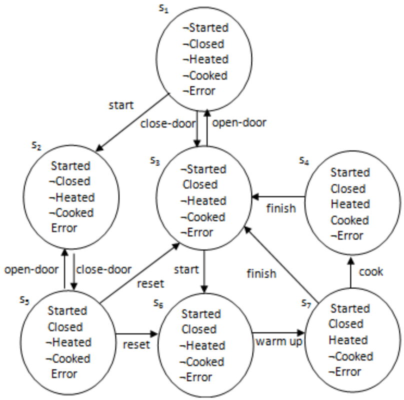
Models: Kripe Structures, Büchi automata, etc . . .

Properties: safety, invariant, liveness, etc . . . .
Expressed by means of *temporal logic formulas*.

- *Data abstraction*: a function mapping variables apppearing in more fine-grained representations of the program into some macro-variables.

  Model abstracts from many features of the actual computing system by including macro-state obtained by a collapse of many actual states.

## Macro-Wave Oven



$M = (S, S_0, R, L)$

$AP = \{Started, \neg Started, Closed, \neg Closed$
$Heated, \neg Heated, Cooked, \neg Cooked,$
$Error, \neg Error\}$

Liveness: whenever the oven is on, it will eventually start heating.
CTL formula: $\mathbf{AG}(Started \rightarrow \mathbf{AF}Heated)$

$M \models \mathbf{AG}(Started \rightarrow \mathbf{AF}Heated)$ ?

## Distortive Idealizations in Model Checking

Excluding system trajectories that are physically possible but either reflect unreasonable interactions with the environment or else arise on account of hardware failures.

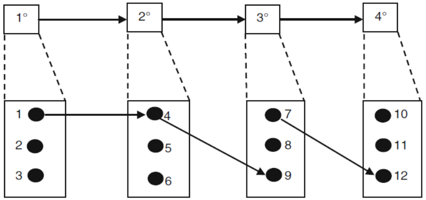
Examples:
$\pi = s_1, s_3, s_1, s_3, \ldots, s_1, s_3$
$\pi' = s_1, s_2, s_5, s_3, s_1$

*Fairness constraints*: defined in terms of the set of states that are required to appear infinitely often in any allowed path and interpreted as CTL formulas.

Example:
$\mathit{Started} \wedge \mathit{Closed} \wedge \neg \mathit{Error}$

## Distortive Idealizations in Model Checking

*False Negatives*: due to the increment of KS's granularity induced by data abstraction.



If lower implementing descriptions were to be considered in the how/that hierarchy, such paths would appear to be a difficulty since lower levels became non-correct implementations.

*Abstraction Refinement*: In case $M \models_F \mathbf{AG}(\mathit{Started} \rightarrow \mathbf{AF}\mathit{Heated})$ produced false negatives, the granularity of the KS is decreased up to individuating the faulty modelled transitions.

## Conclusions

Wesely Salmon (1998, p.75):

“Although some philosophers have maintained that the mechanistic explanation, when it can be given, supersedes the functional explanation, Wright holds that they are perfectly compatible, and that the functional explanation need not give way to the mechanistic explanation. I think he is correct in this view.”

## References

Bechtel, W., & Richardson, R. C. (1993). Discovering complexity. Princeton, NJ: Princeton UP.

Cartwright, N. (1989) [1994]. Natures capacities and their measurement. Oxford, New York: Oxford University Press.

Craver, C. F. (2007). Explaining the brain. Oxford: Oxford University Press.

Cummins, R. (1975). Functional analysis. Journal of Philosophy, 72(20), 741765.

Machamer, P., Darden, L., & Craver, C. F. (2000). Thinking about mechanisms. Philosophy of science, 1-25.

Nowak, L. (1979). The structure of idealization. Towards a systematic interpretation of marxian idea of science. Dordrecht: Kluwer.

Piccinini, G., & Craver, C. (2011). Integrating psychology and neuroscience: Functional analyses as mechanism sketches. Synthese, 183(3), 283-311.

Salmon, W. C. (1998). Causality and explanation. Oxford: Oxford University Press.

Turing, A. M. (1950). Computing machinery and intelligence. Mind, 433-460

Turner, R. (2014). Programming languages as technical artifacts. Philosophy & Technology, 27(3), 377-397.

Weisberg, M. (2007). Three kinds of idealization. The Journal of Philosophy, 104(12), 639659.