‘Informational equivalence’ but ‘computational differences’? Herbert Simon on representations

David Waszek

CNRS (France), Archives Henri Poincaré

Oct. 27, 2021
HaPoC 2021, Zürich, Oct 27–29, 2021

Introduction

This talk is about an idea developed by Herbert Simon:

- differences between forms of representation (e.g., between a diagram and a list of sentences on paper; or between forms of mental representation)

can be understood on the model of

- differences between ways to store the same data in a computer.

> *Two representations are informationally equivalent if the transformation from one to the other entails no loss of information, i.e., if each can be constructed from the other. [...] Two representations are computationally equivalent if the same information can be extracted from each (the same inferences drawn) with about the same amount of computation. (Simon 1978, pp. 4–5)*

I explore where this idea came from, and how Simon tried to make it precise by borrowing the concepts of 'data type' and/or 'data structure' from computer science. Ultimately, I argue that these concepts cannot provide him with the right level of abstraction to ground the ambitious, 'architecture-independent' notion of representation he needed.

The paper most frequently quoted is:

Larkin, J. and Simon, H. A. (1987). 'Why a Diagram Is (Sometimes) Worth Ten Thousand Words', *Cognitive Science* 11.1, pp. 65–100.

which is about diagrams in scientific practice.

But there is much more behind it. In particular:

Simon, H. A. (1978). 'On the Forms of Mental Representation'. In: Savage, C. W., ed. *Perception and Cognition*, Minneapolis: U. of Minnesota Press, pp. 3–18.

Simon, H. A. (1972). 'What Is Visual Imagery? An Information Processing Interpretation'. In: Gregg, L. W., ed. *Cognition in Learning and Memory*, New York: Wiley, pp. 183–204.

Simon's methodology

Simon's overall methodology is to approach both deliberate problem-solving (as in theorem-proving in logic) and largely unconscious cognitive processes by developing computer programs that perform these tasks.

For this purpose, he and Allen Newell developed list-processing programs; in fact, their work played a crucial role in the development of Lisp-like programming languages. See:
Priestley, M. (2017). 'AI and the Origins of the Functional Programming Language Style'. *Minds and Machines* 27.3, pp. 449–472.

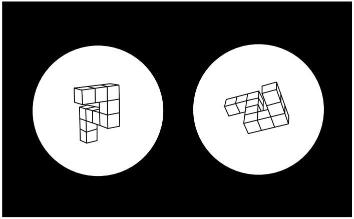
The data used by their programs is organized in 'relational' (or 'associative') structures (basically, nodes containing data + links between nodes).

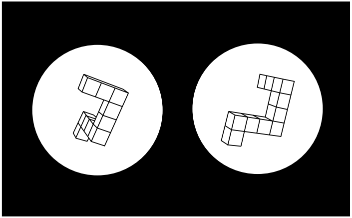
The mental imagery debate

Simon's work on representations is initially rooted in the so-called 'imagery debate' about the forms of mental representations: are mental representations all language-like or are some of them image-like, and what does the distinction even mean?

> *The human brain encodes, modifies, and stores information that is received through its various sense organs, transforms that information by the processes that are called "thinking," and produces motor and verbal outputs of various kinds based on the stored information. So much is noncontroversial [...]. What is highly controversial is how information is stored in the brain—in the usual terminology, how it is "represented"—or even how we can describe representations, and what we mean when we say that information is represented in one way rather than another. (Simon 1978, p. 3)*

This debate was ignited by a number of results coming from experimental psychology, the most famous of which is due to Roger Shepard and Jacqueline Metzler (1971). They displayed pairs of drawings like





and asked experimental subjects whether the two objects presented could be obtained from one another by rotation. Their result: the response time is proportional to the angle between the two objects—as if the subjects were performing a mental rotation at a fixed speed on a mental image.

Roughly, the main strand of the imagery debate is between

- people (most famously Steven Kosslyn) who defended the existence of pixel-array-like mental representations, close to raw perception;
- people (most famously Zenon Pylyshyn) who thought the experimental data could and should be explained by what Pylyshyn called "abstract" mental representations, at a remove from raw perception (and closer to descriptions, though they need not be expressible in sentences of natural language).

On this, Simon is open-minded, though sympathetic to Pylyshyn.

But for Simon, the main question about forms of representation is elsewhere.

He thinks in terms of the ‘relational’ or ‘associative’ structures mentioned above, which he and Newell used for their AI programs. Such a structure, he thought, can encode ‘abstract’ information in Pylyshyn’s sense (e.g., encode ‘a red ball to the left of a blue ball’ rather than clouds of red and blue pixels) and yet, depending on how its links are organized, behave computationally like an image rathen than like sentences. An example will make this clear.

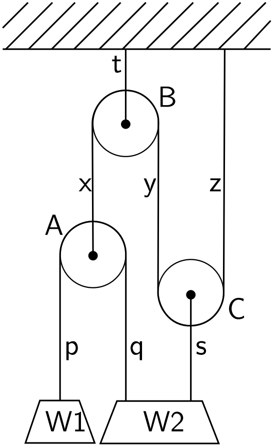
Simon's proposal

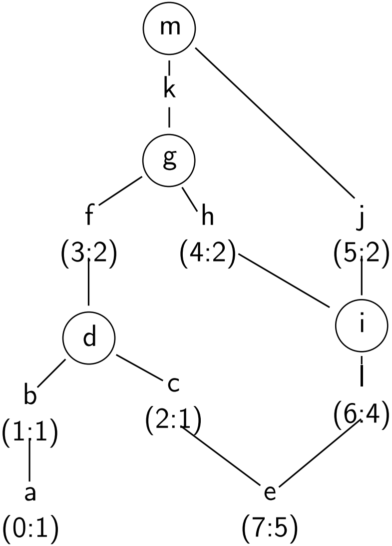
The clearest articulation of Simon's idea is in his 1987 paper, which does not tackle mental representations directly, but 'external' ones (diagrams on paper used in scientific problem-solving). It was co-written with Jill Larkin, a psychologist and educationalist who studied the use of external representations for the solution of physical problems.

Note that for Simon, the external case and the mental case are identical at the right (information-processing) level of abstraction, so the shift is not consequential in his view. As he wrote in his autobiography a couple of years later:

> *In order to deal with the difficulties one by one, [Jill Larkin and I] fudged a bit, alleging that we were talking about diagrams on paper rather than mental pictures; but most of our argument carries over in a straightforward way. (Simon 1989, p. 383).*

The paper studies (among other things) a statics problem with pulleys and ropes, comparing a sentence-based and a diagram-based solution method. Here is the diagram, and the relational structure meant to encode it:





The advantage of the diagram here is supposed to be that, when working on an element (say, a weight), the diagram gives easy access to the elements physically connected to it.

This advantage, absent from Larkin and Simon's sentential representation, is meant to be captured by the relational structure shown above.

In this sense, the relational structure above and the sentential representation (a simpler, linear relational structural) are computationally different.

An abstract concept of representation?

Simon's idea is that, at the right level of analysis, a mental representation, a physical diagram on paper, and a relational structure in a computer program can be the same.

The computational point of view, he believes, provides us with an abstract (i.e., physical realization–independent, or, one might say, architecture-independent) concept of representation that can justify equating these very different entities.

To clarify this abstract concept of representation, he refers to the computer science concepts of 'data types' (in 1978) and of 'data structures' (in 1987). Let us examine what he has in mind.

Data types

The notion of 'abstract data type' in computer science was developed in the early 1970s, mostly for reasons of modularity (i.e., in order to decompose complex programs into simpler components whose behavior could be specified independently of each other), among others by Hoare, Parnas (a colleague of Simon at CMU), and Liskov and Zilles. The latter write:

> *An abstract data type defines a class of abstract objects which is completely characterized by the operations available on those objects. This means that an abstract data type can be defined by defining the characterizing operations for that type. (Liskov and Zilles 1974, p. 51)*

At first sight, Simon uses this idea, but complement it: he needs, not only an interface of operations, but a specification of which operations are fast.

> *Defining a representation means (1) specifying one or more data types, and (2) specifying the primitive (i.e., "fast") operations that can be performed on information in those data types. (Simon 1978, pp. 7–8).*

Indeed, as far as I know, no one in computer science back then was much concerned about adding complexity specification to the interface of a data type; as late as 1994, Meng Lee and Alexander Stepanov wrote:

> *It has been commonly assumed that the (time and space) complexity of an operation is part of its implementation and should not be specified at the interface level. This assumption is incorrect since it invalidates the main reason for the separation of interfaces and implementations, namely, ability to substitute one module for another with the conforming interface. (Lee and Stepanov 1994, p. 26).*

Data structures

So, should we conclude that Simon's representations = abstract data types + complexity specifications for the operations in the interface?

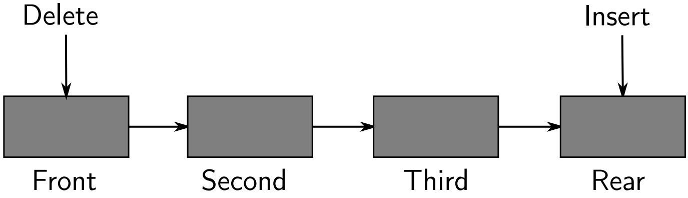
No. Other remarks by Simon complicate the picture:

> *A data type is some particular way of organizing information in memory. For example, among the data types that are commonly used in computing are lists and arrays. [...] The declaration that information will be represented in lists or in arrays does not say anything about the physical location of the information in memory. (Simon 1978, p. 8)*

There, he seems to be referring to what is now more usually called 'data structures', a terminology he indeed shifted to in 1987.

But what exactly is a data structure and can the concept do what Simon needs?

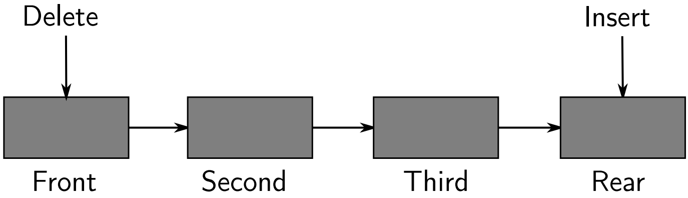
Typically, a data structure, say a queue, is introduced *via* pictures like this (from Knuth),



and *via* discussions of the different ways one can implement it.

So far, this fits with Simon's 'some particular way of organizing information in memory' that 'does not say anything about the physical location of the information in memory'.

An important example for him, coming from his list-processing days, is that a chained list can be implemented on the basis of a number-indexed array (as in standard addressable memory), which he takes to show that a data structure is more abstract than a particular implementation in a particular architecture.



But what precisely does a picture like the above mean? What do the arrows represent?

- One might interpret the arrows as fast operations; then one falls back on the 'interface of fast operations' account suggested above;
- one might want to interpret them as 'explicitly represented' relationships; but a closer look shows that this basically means the same thing (see Kirsh 1990, 'When Is Information Explicitly Represented?');
- or, one might interpret them more concretely, say as pointers in memory; but then, the notion will become architecture-dependent.

In general, it seems that any way of specifying the organization in memory more concretely than through an interface of fast operations will presuppose a particular architecture:

Switching to computers with a different architecture, for instance with memory slots that are accessed by content ("content-addressable memory") rather than—as is usual—by address, will change the relationship between data organization and fast operations.

Conclusion: representations as defined by operations

In sum, it appears that the only way to define a physical realization–independent notion of representation, like Simon needs, is through the first route: by an interface of fast operations, irrespective of possible implementations in particular architectures.

Is that a problem?

Not necessarily, but it shifts the question somewhat from the one Simon initially seemed to be trying to solve.

Think of the pulleys-and-weights problem discussed by Larkin and Simon. In other contexts, the same diagram may be used for other purposes than going from one element (say, a weight) to physically related ones. For example, one might look for symmetries in the problem. (This is even clearer on their second, geometrical example.) The 'fast' operations on the diagram would then be different.

But this means that the same diagram, just used differently, will correspond to a different representation in Simon's sense. In the end, Simon is not comparing the diagram and the sentences in and of themselves, but only relative to a particular way of using each. His approach can do no more.

By speaking of "data structures" that correspond to particular ways of organizing information, which are not tied to a particular implementation or architecture, yet are intrinsically linked to certain fast operations, Simon obfuscates the issue and gives the impression that his approach can achieve more than this; but it cannot.