Informatics in the light of some Leibniz's works

Laurent Bloch lb@laurentbloch.org

24th of May 2016

1 Informatics, not computer science

The science we are going to talk about, which encompasses algorithms, programming, operating systems, computer networks, databases and some others, is usually named in English "Computer Science", but this wording seems to some people (including myself) quite inappropriate, because it is indeed a science, but not about computers.

The German engineer and professor Karl Steinbuch coined the word „Informatik“ in 1957, the French engineer Philippe Dreyfus created the term « informatique » in 1962, both of them translated in English by "Informatics", which, in my humble opinion, is better than "Computer Science". This word collapses together "information" and "automatics"; information is what gives to something a *form* from the *inside* of this thing; automatics is what makes this process automatic; the result is an automaton. The thing to which a form is given here is an idea, we call it an algorithm. We call the form a program, it is a text.

2 Automates

As far as we can look backwards in time, humans have tried to build automates to avoid working. The trap of prehistoric man, whose door opens under or whose harrow falls on the prey, Vaucanson's duck which was floating and swimming were automates.

These particular automates, and some others of the same kind, do always the same thing.

The kind of automates we are speaking about with Informatics are different, they have a language inside themselves, and they are therefore universal, they can do everything an automate can do, as proved by MM. Alan Turing and Alonzo Church (not everything can be done by an automate). These universal automates consist of a material object, a machine, a computer, and of an invisible, intangible thing, the program.

The program, as we have told, is a text. Some programs define also a language, in which another program may be expressed. It is the way data processing machines are universal, and programmable.

Gottfried Wilhelm Leibniz was a forerunner of these ideas, in several ways. The scientists who created the ideas of Informatics (Gottlob Frege, Alonzo Church, Alan Turing) were unaware of the previous works of their forerunner Leibniz. The scientist who invented the modern computer (another poorly suitable word for this device, and the French « ordinateur » is hardly better), John von Neumann, was aware of the works of Church and Turing in the field of Logic, but the link he saw between these works and the design of what we call today "von Neumann's architecture" for the computer was very weak, if only present.

3 How was Leibniz a forerunner of Informatics

- He built a four-operations calculator (1672-1694), with a bug, like every true computer system.
- He had a foresight of how binary arithmetic could be suitable for automatic calculation (1703).
- His *characteristica universalis*, designed to be the language of a *scientia universalis*, didn't achieve that result, but achieved another, showing that computing could apply not only to numbers, but also to logical propositions, an idea developed later, independently from Leibniz, by Augustus de Morgan, George Boole, Gottlob Frege and their followers.

Moreover, the philosopher Baptiste Mélès established a correspondence between Object oriented programming (OOP) style and the Leibnizian concept of Monad.

These deep insights of Leibniz, one of the most influential (even if often forgotten) thinkers of the millennium, towards the world of Informatics, are not just coincidences, but the outward signs of an often (not so) invisible current of thought, flowing from Aristotle to Kurt Gödel and Alan Turing, with the aim of substituting computing for reasoning everywhere it is possible. Alan Turing's major work was to determine a way of computing where it is possible (and where it isn't).

4 Leibniz's pinwheel calculator

Leibniz got the idea of making a calculating machine in 1672 in Paris, and later he learned about Blaise Pascal's machine[1] when he read Pascal's *Pensées* [13][2].

[1]Pascal began to work on his calculator in 1642, but the first mechanical calculator was designed by Wilhelm Schickard. Schickard's machine was destroyed by fire, but we know it by two letters to Johannes Kepler from 1623 and 1624.

[2]« La machine arithmétique fait des effets qui approchent plus de la pensée que tout ce que font les animaux ; mais elle ne fait rien qui puisse faire dire qu'elle a de la volonté, comme les animaux. » ("The arithmetic machine has effects that are closer to thought

Then he became eager to surpass Pascal's machine, which was only capable of addition and subtraction, with the design of a machine able to multiply and to divide. He presented this design to the *Royal Society* of London on 1 February 1673, received much encouragement and was therefore admitted as a member.

His first preliminary brass machine was built between 1674 and 1685. His so-called older machine was built between 1686 and 1694. The 'younger machine', the surviving machine, was built from 1690 to 1720 (according to Jan-Willem Liebezeit [11]).

It is worthwhile to note that being able to compute the four basic arithmetic operations is equivalent to be able to execute any given numerical computation.

For his design, Leibniz invented a mechanism to keep the operation's multiplicand "in memory": the "stepped reckoner", based on a gearwork now called "Leibniz wheel". The ability to keep data in memory is essential to modern computing, technically as well as theoretically. Leibniz's machine was the first device ever to implement that capacity.

There was a bug in Leibniz's design: a flaw in the carry mechanism prevented the machine from accurate computations with numbers of two or three digits. And, like Pascal before him, and like Babbage after him, Leibniz had difficulty obtaining the precision gearwork needed by his machine to work reliably.

5 Binary arithmetic

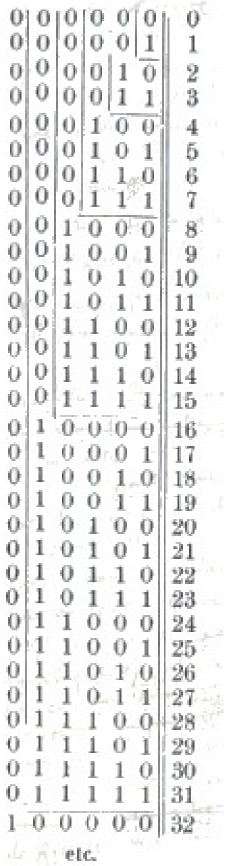
Leibniz got the idea of binary arithmetic by corresponding with, and reading works by, European Christian missionaries posted in China, especially the French Jesuit Joachim Bouvet, who was a civil servant for the Kangxi Emperor, and who sent Leibniz a diagram of the 64 Fuxi's hexagrams. Fuxi (Fohy in Leibniz's text *Explication de l'arithmétique binaire* [8], written in French for the *Académie Royale des Sciences*, the translation in English is here [7]) is a character of Chinese mythology, creator of the Eight Trigrams (Hànyǔ pīnyīn: bāguà) used in daoist cosmology to represent the fundamental principles of reality, seen as a range of eight interrelated concepts. Each consists of three lines, each line either "broken" or "unbroken", respectively representing yin or yang. Due to their tripartite structure, they are often referred to as "trigrams" in English.

If for ancient Chinese people the two kinds of lines were the symbols of yin and yang, they may be also viewed as the two digits of binary arithmetic. Three lines of two kinds may give eight combinations, for the numbers from 0 to 7 ($2^3 - 1$). Fuxi's hexagrams may, with six binary digits, may be viewed as representing the numbers from 0 to 63 ($2^6 - 1$).

than what all animals do, but it does not do anything that could make some say it has a will, as animals have")

Leibniz was neither the first to know Fuxi's hexagrams, nor the first to know binary arithmetic: Thomas Harriot (1560-1621) left a table of binary values, and Juan Caramuel y Lobkowitz (1606-1682) had studied numeral systems with radices (bases) different from **10**. But he was the first one to see a connection between them.

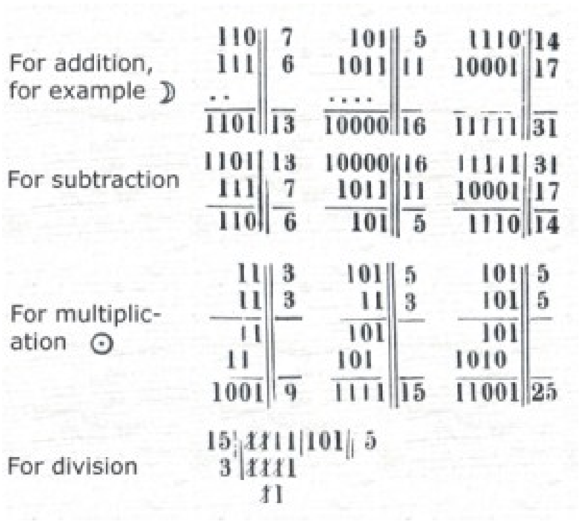
Above all, Leibniz understood that binary arithmetic made calculations much more simple than with other bases: "...instead of the progression of tens, I have for many years used the simplest progression of all, which proceeds by twos, having found that it is useful for the perfection of the science of numbers. Thus I use no other characters in it bar 0 and 1, and when reaching two, I start again...



0	0	0	0	0	0	0
0	0	0	0	0	1	1
0	0	0	0	1	0	2
0	0	0	0	1	1	3
0	0	0	1	0	0	4
0	0	0	1	0	1	5
0	0	0	1	1	0	6
0	0	0	1	1	1	7
0	0	1	0	0	0	8
0	0	1	0	0	1	9
0	0	1	0	1	0	10
0	0	1	0	1	1	11
0	0	1	1	0	0	12
0	0	1	1	0	1	13
0	0	1	1	1	0	14
0	0	1	1	1	1	15
0	1	0	0	0	0	16
0	1	0	0	0	1	17
0	1	0	0	1	0	18
0	1	0	0	1	1	19
0	1	0	1	0	0	20
0	1	0	1	0	1	21
0	1	0	1	1	0	22
0	1	0	1	1	1	23
0	1	1	0	0	0	24
0	1	1	0	0	1	25
0	1	1	0	1	0	26
0	1	1	0	1	1	27
0	1	1	1	0	0	28
0	1	1	1	0	1	29
0	1	1	1	1	0	30
0	1	1	1	1	1	31
1	0	0	0	0	0	32

etc.

Establishing this expression of numbers enables us to very easily make all sorts of operations.



```
                          110 | 7      101 | 5      1110 | 14
For addition,             111 | 6     1011 | 11    10001 | 17
for example ☽             ..          ....
                         1101 | 13   10000 | 16    11111 | 31

                         1101 | 13   10000 | 16    11111 | 31
For subtraction           111 | 7     1011 | 11    10001 | 17
                          110 | 6      101 | 5      1110 | 14

                           11 | 3      101 | 5       101 | 5
                           11 | 3       11 | 3       101 | 5
For multiplic-             11          101           101
ation ⊙                   11          101          1010
                         1001 | 9     1111 | 15    11001 | 25

                         15 | 1111 | 101 | 5
For division              3 | 1111
                              11
```

And all these operations are so easy that there would never be any need to guess or try out anything, as has to be done in ordinary division. There would no longer be any need to learn anything by heart, as has to be done in ordinary reckoning, where one has to know, for example, that 6 and 7 taken together make 13, and that 5 multiplied by 3 gives 15, in accordance with the Table of one times one is one, which is called Pythagorean." (Translation by Lloyd Strickland)[3]

With the binary numeral system, the multiplication becomes extremely simple, because the only multiplication table is the table of one time one. But moreover the digits 0 and 1 may represent either their numerical values, or the values "false" and "true" of the algebraic logic invented by George Boole in the middle of XIXth century [2].

Boole's algebra provides a formalism for propositional logic. For instance, let P and Q be two propositions, "P AND Q" is true if both P and Q are true, and false otherwise. "P OR Q" is true if P, or Q, or both of them are true. Augustus de Morgan showed that "not (P and Q)" is the same as "(not P) or (not Q)" also, "not (P or Q)" is the same as "(not P) and (not Q)". Boolean formalism (not the original Boole's text) uses the following symbols:

- $\neg$ is the negation logic operator (NOT),
- $\wedge$ is the conjunction logic operator (AND),
- $\vee$ is the disjunction logic operator (OR),
- $\iff$ is a metalogical symbol meaning "can be replaced in a logical proof with".

[3] « Et toutes ces opérations sont si aisées, qu'on n'a jamais besoin de rien essayer ni deviner, comme il faut faire dans la division ordinaire. On n'a point besoin non plus de rien apprendre par cœur ici, comme il faut faire dans le calcul ordinaire, où il faut savoir, par exemple, que 6 et 7 pris ensemble font 13, et que 5 multiplié par 3 donne 15, suivant la Table d'une fois un est un, qu'on appelle Pythagorique. »

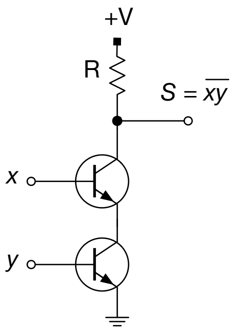
With these symbols, de Morgan's duality may be written as follows:

$$\neg(P \wedge Q) \iff (\neg P) \vee (\neg Q) \tag{1}$$
$$\neg(P \vee Q) \iff (\neg P) \wedge (\neg Q) \tag{2}$$

Therefore it is possible to implement calculations with numbers by logical operations of Boolean algebra. These operations are attractive because they are particularly simple, and easy to implement with electronic circuits. All the modern computers are built with only one operation, the logical "Not And" (NAND), noted $\uparrow$:

$$x \uparrow y \iff \neg(x \wedge y)$$

$x \wedge y$ may be written xy,
$x \vee y$ may be written $x + y$ and
$\neg(x \wedge y)$ simply $\overline{xy}$
(figure by courtesy of Emmanuel Lazard)



6 What are computers?

Modern computers are indeed very simple devices, more simpler than mechanical machines designed by Schickard, Pascal, Leibniz or Babbage. As mentioned above, they are all made of NAND circuits (the usual wording is "NAND gate"). They consist essentially of a memory, which is a storage to store values coded by binary digits, an arithmetic and logical unit (ALU), which joins together logical circuits to implement arithmetic and logical operations, and a control unit with control operations (technically similar to logical and arithmetic operations) to trigger the execution of operations on the right order, at the right time.

It is maybe quite difficult to understand because it is too simple: this simplicity is indeed incredible. Operations are physical devices, made up of electronic circuits, which have effects on data stored in memory. The sequence of operations is described by the text of a program (it is the complicated part of the whole thing); the sentences of the text are called instructions; an instruction corresponds to an operation of the ALU or of the control unit. Thanks to John von Neumann, the text of the program is data like other data, stored in memory too. The circuits of the control unit "read" the text of the program and trigger the execution of the operations, one by one (disregarding some technical details not relevant here).

The effects of control operations are of a few sorts:

- to modify the sequence of operations by giving the place of the next instruction to be read;

- to choose the next operation, depending of the result of the previous one;
- to repeat a sequence of operations;
- to *load* data from memory into special little storage inside the ALU, in order to have faster execution of some operations; these storage places in the ALU are called registers; registers are analogous to the "stepped reckoner" invented by Leibniz and mentioned above, they are useful to keep an operand for the time of the operation it is submitted to;
- to *store* data from registers into memory.

The effects of arithmetic and logic operations are of a few sorts:

- to move data around in memory;
- to copy data from one place to another in memory;
- to modify data according to boolean operations;
- to combine data according to boolean operations; the result of such a combination may be an arithmetical operation.

As all data are numerically encoded, all operations on data, numerical, textual, graphic or whatever, are arithmetic (i.e. logical) operations.

7 Putting all the pieces together

The computing system we have described so far is set up from many pieces, all of them are quite simple, but to work efficiently with it we need to put them together in a coherent way. The way to achieve coherence is *language*, and here too Leibniz had the right ideas. Unfortunately, almost everybody was unaware of them, and they had to be reinvented by people like Gottlob Frege, Alonzo Church, Alan Turing and some others. And also, the authors of the first computer languages like John Backus were originally unaware of the works of Frege, Church and Turing, and they had to be rediscovered later.

In fact, the need for true languages was not evident in the first place. When John Backus invented Fortran, he was thinking of mathematical formulas. Later, with the Algol project, the idea of language came up.

Behind the computer languages is the idea of *formal system* (a formal system consists of a finite number of rules which operate on countable sets). Before the modern formal systems, Leibniz had the idea of a *characteristica universalis*, a language for a *scientia universalis*. Leibniz hasn't written a formal treatise upon this project, his ideas about it are spread out in many unpublished papers and letters, and Louis Couturat [5] and Bertrand Russell [14] deserve the credit for putting them in light. I'm borrowing from Renée Bouveresse [4] her description of Leibniz's program for an *universal science*; it consists of two parts: an *universal characteristic*, or notation system, to write down any information item without ambiguity and to help for communication between scientists, and a *calculus ratiocinator*, or formal method for reasoning, because calculus is nothing else as an operation by means of characters, which is relevant not only for quantities, but also for any sort of reasoning.

These ideas of logical calculus are exactly the ideas inherent to computer programming.

8 Communication between objects

"Object-oriented programming (OOP) is a programming paradigm based on the concept of 'objects', which may contain data, in the form of fields, often known as *attributes*; and code, in the form of procedures, often known as *methods*." (Wikipedia).

This style of programming, as opposed to procedural and functional styles, avoids sharing data between procedures or passing arguments between them. Procedures and data are encapsulated into so-named *objects* which communicate between each other by message exchange. The messages contain data and indication of the methods that the emitting object asks the receiving object to apply to the data.

According to this model of program, an object may *receive* messages, which modify its status, and *send* messages, which tend to modify other objects, i.e. the world around[4].

Baptiste Mélès noticed [12] that an analogy could be built between this behaviour of objects and some properties of Leibniz's *monads*: "The qualities of a monad must be its perceptions; a perception is a representation in something simple of something else that is composite. And a monad's actions must be its appetitions, which are its tendencies to go from being in one state to being in another, i.e. to move from one perception to another; these tendencies are the sources of all the changes it undergoes." (Translation by Jonathan Benett)[5].

In his paper Baptiste Mélès gives to us some examples of objects in Java. He notices that the programmer, in this world of classes (types) of objects, occupies the place of God, because he knows (he's supposed to know…) everything about all classes.

OOP gives us objects able to communicate between each other: this property could help to elaborate models around the notion of *biological function*,

[4] In the beginning, OOP gave rise to great enthusiasm, it was supposed to give computer programming the power to *represent* the real world. It was of course an illusion, computer programs artifacts can only represent, manage and process information, and OOP is a particular way to do that. But OOP may be a programming style well suitable to build abstract models of some real world's objects.

[5] « Une monade en elle-même, et dans le moment, ne saurait être discernée d'une autre que par les qualités et actions internes, lesquelles ne peuvent être autre chose que ses perceptions (c'est-à-dire les représentations du composé, ou de ce qui est dehors dans le simple), et ses appétitions (c'est-à-dire ses tendances d'une perception à l'autre) qui sont les principes du changement. »

which is the reason some object or process occurred in a system that evolved through a process of selection or natural selection.

Conclusion

Leibniz's work is immense, he approached numerous domains with thoughts well in advance on his time, and some of them are still to be discovered and understood. It is a pity that Voltaire, a by far lesser philosopher, could have mocked him with cruelty, and moreover after his death, in his novel *Candide*, where he is painted as the philosopher Pangloss, whose mantra is "all is for the best" in the "best of all possible worlds". If, as stated by Roland Barthes, Voltaire belongs to the Academy of the overrated, it is not the case for Leibniz.

It is still to be well established that the science of Informatics is the daughter of Logic, probably more than of Mathematic. According to the German logician Heinrich Scholz, in the science of Logic there are two periods: from Aristotle to Leibniz, and from Leibniz until now, because Leibniz introduced new ways of logical calculus. Almost nobody was aware of Leibniz's works on Logic, and it is only in the second half of the XIXth century, when similar ideas arose (de Morgan, Boole, Frege...), that they were discovered.

In any case, Leibniz's writings on Logic and Calculus would be of great help for informaticians to better understand their science.

References

[1] Laurent Bloch. *Les systèmes d'exploitation des ordinateurs – Histoire, fonctionnement, enjeux.* Paris: Vuibert, 2003. URL: http://www.laurentbloch.org/MySpip3/Systeme-et-reseau-histoire-et.

[2] George Boole. *An Investigation of the Laws of Thought.* Dover, 1854. URL: https://archive.org/details/aninvestigationo15114gut.

[3] Jacques Bouveresse. *Essais V - Descartes, Leibniz, Kant.* Marseille: Agone, 2006. URL: http://agone.org/bancdessais/essaisv/.

[4] Renée Bouveresse. *Leibniz.* Paris: PUF, 1994.

[5] Louis Couturat. *La Logique de Leibniz.* Paris: Alcan (Olms), 1901. URL: http://gallica.bnf.fr/ark:/12148/bpt6k110843d.

[6] Paul Graham. *Hackers and Painters.* May 2003. URL: http://www.paulgraham.com/hp.html.

[7] Gottfried Wilhelm von Leibniz. *Explanation of binary Arithmetic.* Translation by Lloyd Strickland. 1703. URL: http://www.leibniz-translations.com/binary.htm.

[8] Gottfried Wilhelm von Leibniz. *Explication de l'arithmétique binaire.* 1703. URL: http://www.laurentbloch.org/MySpip3/L-arithmetique-binaire-par-Leibniz-98.

[9] Gottfried Wilhelm von Leibniz. *Principes de la Nature et de la Grâce fondés en raison - Monadologie*. Garnier-Flammarion, 1714. URL: `https://fr.wikisource.org/wiki/Principes_de_la_nature_et_de_la_gr%C3%A2ce`.

[10] Gottfried Wilhelm von Leibniz. *Principles of Nature and Grace Based on Reason*. Translation by Jonathan Bennett. Early Modern Texts, 1714. URL: `http://www.earlymoderntexts.com/assets/pdfs/leibniz1714a.pdf`.

[11] Jan-Willem Liebezeit. *Leibniz Rechenmaschinen*. Friedrich Schiller Univ. of Jena. July 2004. URL: `http://www.fmi.uni-jena.de/Fakult%C3%A4t/Institute+und+Abteilungen/Abteilung+f%C3%BCr+Didaktik/Didaktik+der+Informatik/Leibniz_Rechenmaschinen-p-121165.html`.

[12] Baptiste Mélès. *Dieu a-t-il programmé le monde en Java ? - Monadologie leibnizienne et programmation orientée objet*. Aug. 2015. URL: `http://baptiste.meles.free.fr/site/B.Meles-Leibniz_prog_objet.pdf`.

[13] Blaise Pascal. *Pensées*. Fragment 741 (Lafuma), 617 (Sellier), 340 (Brunschvicg). 1670. URL: `http://www.penseesdepascal.fr/`.

[14] Bertrand Russell. *A critical exposition of the philosophy of Leibniz, with an appendix of leading passages*. Cambridge, [Eng.]: The University Press, 1900. URL: `https://archive.org/details/cu31924052172271`.