LANGUAGE

FLORIAN CRAMER

Software may process language, and it is constructed with language. The language of which software is made up is the vocabulary and formal-logical grammar in which its algorithms are expressed and in which are used by its controls. In other words, it is the programming language in which the software is being written, it is implemented within the software as its symbolic controls, or – in the case of compilers, shells or macro language for example – both at once. Computer programming languages are conventionally called "artificial languages", and languages like English "natural languages"; a problematic terminology because nothing is "natural" about spoken language either, which is a cultural construct, and therefore "artificial", just as much as any machine control language. The term "machine language" doesn't cut it either because it obscures that "machine languages", too, are human creations, written by humans, in the case of high-level machine-independent programming languages such as Fortran C, Java, Basic, they are not even direct mappings of machine registers. If programming languages are human languages for machine control, they could be called cybernetic languages. But these languages can also be used outside machines – in programming handbooks, for example, in programmer's dinner table jokes, or as abstract formal languages for expressing logical constructs, such as in Hugh Kenner's use of the Pascal programming language to explain structures of Samuel Beckett's writing.

In this sense, computer control languages could be more broadly defined as formal – or syntactical – languages as opposed to semantic languages. But neither is this differentiation without its problems. Common languages like English are both formal and semantic; anything that can be expressed in a computer control language can as well be expressed in them. It follows that computer control languages are a formal (and as such rather primitive) subset of common human languages.

To complicate things even further, computer science has its own understanding of an "operational semantics" in programming languages, for example in the construction of a programming language interpreter or compiler. However, just as this interpreter doesn't perform "interpretations" in a hermeutic sense of semantic text explication, the computer science notion of "semantics" defies linguistic and common sense understanding of the word, since compiler construction is purely syntactical, and programming languages denote nothing but syntactical manipulations of symbols.

What could rather be called the semantics of computer control languages resides in the symbols with which those operations are denoted: in most programming languages, English words like "if", "then", "else", "for", "while", "goto", "print" in conjunction with arithemical and punctuation symbols; in alphabetic software controls, words like "list", "move", "copy" and "paste", in graphical software controls, symbols like the trash can.

Ferdinand de Saussure states that the signs of common human language are arbitrary because it's purely a cultural-social convention which assigns certain phonemes to certain concepts. Likewise, it's a purely a cultural convention to assign certain symbols to certain machine operations. But just as the cultural choice of phonemes in spoken language is restrained by what the human voice can pronounce, the assignment of symbols to machine operations is limited to what can be efficiently processed by the machine and well used by humans.[1] This compromise between operability and usability is obvious, for example, in Unix commands. Originally used on teletype terminals, the operation "copy" was abbreviated to the command "cp", "move" to "mv", "list" to "ls" etc., in order to cut down machine storage capacity, teletype paper consumption and human typing effort at the same time. Any computer control language thus is a cultural compromise between the constraints of machine design – which is far from "objective", but based on human choices, culture, and thinking style itself[2] – and the equally subjective use preferences, involving fuzzy factors like readability, elegance and usage efficiency.

But while the symbols of computer control languages inevitably do have semantic connotations, simply because there exist no symbols

[1] See the section "Saussurean Signs and Material Matters" in N. Katherine Hayles, My Mother Was a Computer, Chicago: The University of Chicago Press, 2005, 42-45

[2] for example, Steve Wozniak's design of the Apple I mainboard was considered "a beautiful work of art" in its time according to Steven Levy, Levy, Insanely Great: The life and times of Macintosh, New York: Viking, 1994, 81

without such connotations in the human world, they can't denote any meaning themselves. This is no historically new phenomenon, but has always applied, for example, to mathematical formulas.

In comparison common human languages, the Babylonian multitude of programming language is of lesser significance. The criterium of Turing completeness of a programming language, i.e. that any computation can be expressed in it, means that every programming language is, formally seen, just a riff on every other programming language. Nothing can be expressed in a Turing-complete language such as C that couldn't also be expressed in a Turing-complete language such as Lisp (or Fortran, Smalltalk, Java. . .) and vice versa. The fact that these languages are functionally interchangeable shows the importance of cultural factors in programming – the semantics of its mnemonic handles and the style in which algorithms can be expressed in it.

Just as programming languages are a subset of common languages, Turing incomplete computer control languages are a constrained subset of Turing complete languages. This prominently includes markup languages (HTML) respectively file formats, network protocols and most user controls (see *interface*) of computer programs. In most cases, languages of this type are restrained from denoting algorithmic operations for computer security reasons – to prevent virus infection and remote takeover. This shows how the very design of a formal language is a design of machine control. Access to hardware functions is limited not only through the software application, but through the syntax the software application may use for storing and transmitting the information it processes. To name one computer control language a "programming language", another a "protocol" and yet another a "file format" is merely a convention, a nomenclature of indicating different degrees of syntactic restraint built into the design of a computer control language.

In its most powerful, Turing complete superset, computer control language is language that executes. Like in magic and speculative concepts of language, the word automatically performs the operation. Yet this is not to be mixed up with what linguistics calls a "performative" or "illocutionary" speech act, for example in the words of a judge who speaks a verdict, or a leader giving a command or a government making a law. The execution of computer control languages is purely formal, the manipulation of a machine, and not a social performance based on human conventions such as accepting a verdict. Computer

languages become performative only through the social impact of the processes they trigger when their outputs aren't critically checked. Joseph Weizenbaum's software psychotherapist Eliza, a simple program that syntactically transforms input phrases, is a classical example, or the 1987 New York stock exchange crash that was partly caused by a chain reaction of "sell" recommendations by day trading software.

Writing in a computer programming language means to phrase instructions for an utter idiot. The project of Artificial Intelligence is to practically prove that intelligence is just a matter of a sufficiently massive layering of foolproof recipes; in linguistic terms, that semantics is nothing else but (more elaborate) syntax. As long as A.I. fails to deliver this proof, the difference between common languages and computer control languages continues to exist, and language processing through computers remains restrained to formal string manipulations, a fact that has made many experimental poets since the 1950s abandon their experiments with computer-generated texts after initial enthusiasm.[3]

The history of computing is rich with confusions of formal with common human languages, and false hopes and promises that formal languages would become more like common human languages eventually: among them, artificial intelligence research, graphical user interface design with its promise of an "intuitive" or "humane interface" (Jef Raskin), digital art that misperceives its programmed behaviorist black boxes as "interactive", and artists caught in the misconception that they can overcome the Western male binarism of computer languages by reshaping them after their romanticized image of indigenous human languages.

Still, the digital computer is a symbolic machine that computes syntactical language and processes alphanumerical symbols, it treats all data – including images and sounds – as textual – which is the very meaning of "digitization". Correspondingly, all computer software controls are linguistic regardless their perceivable shape, alphanumerical writing, graphics, sound signals, or whatever else. The command "rm file" is operationally identical to dragging the file into the trashcan of a desktop. Both are just different encodings for the same operation, just as alphabetic language and morse beeps are different encodings for the same characters. As a symbolic handle, this encoding

[3]Among them, concrete poetry writers, French Oulipo poets, the German poet Hans Magnus Enzensberger and the Austrian poets Ferdinand Schmatz and Franz Josef Czernin.

may enable or restrain certain uses of the language though. In this respect, the differences between ideographic-pictorial and abstract-symbolic common languages also apply to computer control languages. Pictorial symbols simplify control languages simplify denotation of predefined objects and operations, but make it more difficult to link them through a grammar and thus express different operations. In other words, just as a pictogram of a house is easier to understand than the letters h-o-u-s-e, the same is true for the trashcan icon in comparison to the "rm" command. But it is difficult to precisely express the operation "Tomorrow at six, I will clean up every second room of the house" through a series of pictograms, the same is true for phrasing more complex computational instructions, for which more abstract alphanumeric language is a better tool.[4] Although the project of expressing just syntactical operations, not semantics in a pictorial computer control language was much less ambitious, it reenacts the rise and eventual failure of universal pictorial language utopias in the Renaissance, from Tommaso Campanella's "Città del sole" to Comenius' "Orbis pictus".

The oppposite approach to utopian language designs occurs when computer control languages get appropriated and used informally in everyday culture. Jonathan Swift tells how scientists on the flying island of Lagado "would, for example, praise the beauty of a woman, or any other animal, [...] by rhombs, circles, parallelograms, ellipses, and other geometrical terms". Likewise, there is programming language poetry which, unlike most algorithmic poetry, writes its program source as the poetical work, or crossbreeds cybernetic with common human languages. These "code poems" or "codeworks" often play with the interference between human agency and programmed processes in computer networks.

In computer programming and computer science, "code" is often understood either as a synonym of computer programming language or as a text written in such a language. This modern usage of the

[4] Alan Kay, the inventor of the graphical user interface, conceded in 1990 that "it would not be surprising if the visual system were less able in this area than the mechanism that solve noun phrases for natural language. Although it is not fair to say that 'iconic languages can't work' just because no one has been able to design a good one, it is likely that the above explanation is close to truth". This status quo hasn't changed since. Alan Kay, User Interface: A Personal View, in: Brenda Laurel (ed.), The Art of Human-Computer Interface Design, Reading, Mass.: Addison-Wesley, 1990, p. 203

term “code” differs from the traditional mathematical and cryptographic notion of code as a set of formal transformation rules that transcribe one group of symbols into another group of symbols, for example, written letters into morse beeps. The translation that occurs when a text in a programming language gets compiled into machine instructions is not an encoding in this sense because the process is not one-to-one reversible. This is why prorietary software companies can keep their source “code” secret. Most probably, the computer cultural understanding of “code” is historically derived from the name of first high-level computer programming language, “Short Code” from 1950.[5] The only programing language which is a code in the original sense is assembly language, the human-readable mnemonic one-to-one representation of processor instructions. Conversely, those instructions can be coded back, or “disassembled”, into assembly.

Software as a whole is more than “code” because it also involves cultural practices of its employment and appropriation. But since writing (or “code”) in a computer control language is thus what materially makes up software, critical thinking about computers is not possible without an informed understanding of the structural formalism of its control languages. Artists and activists since the French Oulipo poets and the MIT hackers in the 1960s have shown how their limitations can be embraced as creative challenges. Likewise, it’s upon critics to reflect the sometimes more and sometimes less amusing constraints and game rules computer control languages write into culture.

[5]See Wolfgang Hagen, “The Style of Source Codes,” in: Wendy Hui Kyong Chun and Thomas Keenan (ed.), New Media, Old Media, New York: Routledge, 2005