Language and Software Studies, by Florian Cramer

Language Florian Cramer Software and language are intrinsically related , since software may process language , and is constructed in language . Yet language means different things in the context of computing : formal languages in which algorithms are expressed and software is implemented , and in so-called “ natural ” spoken languages . There are at least two layers of formal language in software : programming language in which the software is written , and the language implemented within the software as its symbolic controls . In the case of compilers , shells , and macro languages , for example , these layers can overlap . “ Natural ” language is what can be processed as data by software ; since this processing is formal , however , it is restricted to syntactical operations . While differentiation of computer programming languages as “ artificial languages ” from languages like English as “ natural languages ” is conceptually important and undisputed , it remains problematic in its pure terminology : There is nothing “ natural ” about spoken language ; it is a cultural construct and thus just as “ artificial ” as any formal machine control language . To call programming languages “ machine languages ” doesn ’ t solve the problem either , as it obscures that “ machine languages ” are human creations . High-level machine-independent programming languages such as Fortran , C , Java , and Basic are not even direct mappings of machine logic . 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 aspects of the structure of Samuel Beckett ’ s writing.1 In this sense , computer control languages could be more broadly defined as syntactical languages as opposed to semantic languages . But this terminology is not without its problems either . Common languages like English are both formal and semantic ; although their scope extends beyond the formal , anything that can be expressed in a computer control language can also be expressed in common language . 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 “ operational semantics ” in programming languages , for example in the construction of a programming language interpreter or compiler . Just as this interpreter doesn ’ t perform “ interpretations ” in a hermeneutic 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 might more suitably 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 , ” and “ print , ” in conjunction with arithmetical and punctuation symbols ; in alphabetic software controls , words like “ list , ” “ move , ” “ copy , ” and “ paste ” ; in graphical software controls , such as symbols like the trash can . Ferdinand de Saussure states that the signs of common human language are arbitrary2 because it ’ s purely a cultural-social convention that assigns phonemes to concepts . Likewise , it ’ s purely a cultural convention to assign symbols to 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 of good use to humans.3 This compromise between operability and usability is obvious in , for example , 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 memory use , teletype paper consumption , and human typing effort at the same time . Any computer control language is thus a cultural compromise between the constraints of machine design—which is far from objective , but based on human choices , culture , and thinking style itself 4—and the equally subjective user preferences , involving fuzzy factors like readability , elegance , and usage efficiency . The symbols of computer control languages inevitably do have semantic connotations simply because there exist no symbols with which humans would not associate some meaning . But symbols can ’ t denote any semantic statements , that is , they do not express meaning in their own terms ; humans metaphorically read meaning into them through associations they make . Languages without semantic denotation are not historically new phenomena ; mathematical formulas are their oldest example . In comparison to common human languages , the multitude of programming languages is of lesser significance . The criterion of Turing completeness of a programming language , that is , that any computation can be expressed in it , means that every programming language is , formally speaking , just a riff on every other programming language . Nothing can be expressed in a Turingcomplete language such as C that couldn ’ t also be expressed in another Turingcomplete language such as Lisp (or Fortran , Smalltalk , Java ...) and vice versa . This ultimately proves the importance of human and cultural factors in programming languages : while they are interchangeable in regard to their control of machine functions , their different structures—semantic descriptors , grammar and style in which algorithms can be expressed—lend themselves not only to different problem sets , but also to different styles of thinking . Just as programming languages are a subset of common languages , Turingincomplete computer control languages are a constrained subset of Turingcomplete languages . This prominently includes markup languages (such as HTML) , file formats , network protocols , and most user controls (see the entry “ 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 for 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 indicating different degrees of syntactic restraint built into the very design of a computer control language . In its most powerful Turing-complete superset , computer control language is language that executes . As with magical and speculative concepts of language , the word automatically performs the operation . Yet this is not to be confused with what linguistics calls a “ performative ” or “ illocutionary ” speech act , for example , the words of a judge who pronounces a verdict , a leader giving a command , or a legislator passing a law . The execution of computer control languages is purely formal ; it is the manipulation of a machine , 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 , especially 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,5 as is the 1987 New York Stock Exchange crash that involved a chain reaction of “ sell ” recommendations by day trading software.6 Writing in a computer programming language is phrasing instructions for an utter idiot . The project of Artificial Intelligence is to 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 after initial enthusiasm has made many experimental poets since the 1950s abandon their experiments with computer-generated texts.7 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 . Among the unrealized hopes are artificial intelligence , graphical user interface design with its promise of an " intuitive " or , to use Jef Raskin ' s term , " humane interface , " 8 and major currents of digital art . Digital installation art typically misperceives its programmed behaviorist black boxes as " interactive , " and some digital artists are caught in the misconception that they can overcome what they see as the Western male binarism of computer languages by reshaping them after romanticized images of indigenous human languages . 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 , that is , as chunks of coded symbols . Nelson Goodman ' s criteria of writing as " disjunct " and " discrete , " or consisting of separate single entities that differ from other separate single entities , also applies to digital files.9 The very meaning of " digitization " is to structure analog data as numbers and store them as numerical texts composed of discrete parts . All computer software controls are linguistic regardless of their perceivable shape , alphanumerical writing , graphics , sound signals , or whatever else . The Unix command " rm file " is operationally identical to dragging the file into the trashcan on 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 may enable or restrain certain uses of the language . In this respect , the differences between ideographic-pictorial and abstract-symbolic common languages also apply to computer control languages . Pictorial symbols simplify control languages through predefined objects and operations , but make it more difficult to link them through a grammar and thus express custom operations . 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 " If I am home tomorrow at six , I will clean up every second room in the house " through a series of pictograms . Abstract , grammatical alphanumeric languages are more suitable for complex computational instructions.10 The utopia of a universal pictorial computer control language (with icons , windows , and pointer operations) is a reenactment of the rise and eventual fall of universal pictorial language utopias in the Renaissance , from Tommaso Campanella ' s " Città del sole " to Comenius ' " Orbis pictus " — although the modern project of expressing only machine operations in pictograms was less ambitious . The converse 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. " 11 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 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 to 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 oneto-one reversible . This is why proprietary software companies can keep their source “ code ” secret . It is likely that the computer cultural understanding of “ code ” is historically derived from the name of the first high-level computer programming language , “ Short Code ” from 1950.12 The only programming language that 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 language . Software as a whole is not only “ code ” but a symbolic form involving cultural practices of its employment and appropriation . But since writing in a computer control language is 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 is incumbent upon critics to reflect the sometimes more and sometimes less amusing constraints and game rules computer control languages write into culture . Notes 1 . Hugh Kenner , “ Beckett Thinking , ” in Hugh Kenner , The Mechanic Muse , 83–107 . 2 . Ferdinand de Saussure , Course in General Linguistics , ” Chapter I : Nature of the Linguistic Sign. ” 3 . See the section , “ Saussurean Signs and Material Matters , ” in N. Katherine Hayles , My Mother Was a Computer , 42–45 . 4 . For example , Steve Wozniak ’ s design of the Apple I mainboard was consijdered “ a beautiful work of art ” in its time according to Steven Levy , Insanely Great : The Life and Times of Macintosh , 81 . 5 . Joseph Weizenbaum , “ ELIZA—A Computer Program for the Study of Natural Language Communication between Man and Machine. ” 6 . Marsha Pascual , “ Black Monday , Causes and Effects. ” 7 . Among them concrete poetry writers , French Oulipo poets , the German poet Hans Magnus Enzensberger , and the Austrian poets Ferdinand Schmatz and Franz Josef Czernin . 8 . Jef Raskin , The Humane Interface : New Directions for Designing Interactive Systems . 9 . According to Nelson Goodman ’ s definition of writing in The Languages of Art , 143 . 10 . Alan Kay , an 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 : Addison Wesley , 1989 , 203 . 11 . Swift , Jonathan , Gulliver ’ s Travels , Project Gutenberg Ebook , available at http : // www.gutenberg.org / dirs / extext197 / gltrv10.txt / . 12 . See Wolfgang Hagen , “ The Style of Source Codes . ”