On Commands and Executions: Tyrants, Spectres and Vagabonds

Gauthier, D.

Publication date
2018
Document Version
Final published version
Published in
Executing Practices

DATA browser 06
EXECUTING PRACTICES

Geoff Cox
Olle Essvik
Jennifer Gabrys
Francisco Gallardo
David Gauthier
Linda Hilfling Ritasdatter
Brian House
Yuk Hui
Marie Louise Juul Søndergaard
Peggy Pierrot
Andy Prior
Helen Pritchard
Roel Roscam Abbing
Audrey Samson
Kasper Hedegård Schiølin
Susan Schuppli
Femke Snelting
Eric Snodgrass
Winnie Soon
Magdalena Tyżlik-Carver

DATA browser 06
EXECUTING PRACTICES

Edited by Helen Pritchard, Eric Snodgrass and Magda Tyżlik-Carver

Published by
Open Humanities Press 2018

Freely available at
data-browser.net/db06.html

ISBN (print): 978-1-78542-056-6
ISBN (PDF): 978-1-78542-057-3
ISBN (ePUB): 978-1-78542-058-0

DATA browser series template designed by Stuart Bertolotti-Bailey.

Book layout and typesetting by Mark Simmonds & Esther Yarnold

The cover image is derived from Multi by David Reinfurt, a software app that updates the idea of the multiple from industrial production to the dynamics of the information age. Each cover presents an iteration of a possible 1,728 arrangements, each a face built from minimal typographic furniture, and from the same source code.
www.o-r-g.com/apps/multi

On Commands and Executions: Tyrants, Spectres and Vagabonds

David Gauthier

It is difficult to address the notion of command and execution without addressing that of tyranny. The concept of execution is an eerie construct that at once implies a prescription and a proscription in its suggestion that a rule or command is imposed and enforced on an indeterminate substrate (subjects, objects, matter or otherwise). Thus, it also suggests a certain type of violence that is at once effected and effaced, or, differently put, execution insinuates a despotic foreclosure. In that sense, the problematics of execution are central to the notion of control, which speaks both to the order of reason that it imposes and by which it is assessed. It also points to moments and milieux of erasure where a given order vanishes in indeterminacy—intervals and gaps that the order itself creates and forbids, its necessary residual exterior.

While the software/hardware divide has been a recurrent topic of conversation within the field of Software Studies, I argue that the subject needs to be pushed forward to consider the under-theorised notions of command/execution. Moving from a conception of software as ideology to a conception of software as tyranny, this article shows how the symbolic order of the law, which underpins notions of command and instruction, leads to an impasse when confronted with the question of execution. In turn, rather than seeking an understanding of execution from the despotic perspective of commands and instructions, the current inquiry identifies the various loci where such a perspective collapses and it petitions for a practice of execution that conceives of it as an event in its own right rather than a mere afterthought.

Software as Ideology

In order to illustrate the problematic the notion of execution entails, I will first focus on a particular debate about source code and ideology that took place between Wendy Hui Kyong Chun (2005, 2008) and Alexander R. Galloway (2006). This debate was partly prompted by the nascent field of

Software Studies which elected "software" as the prime object of study of New Media discourse (Fuller 2006). In her articles, Chun warns that in divorcing software from hardware and in focusing on its discursive and semantic aspects, one effects an epistemological and political move since "software perpetuates certain notions of seeing as knowing … creating an invisible system of visibility. The knowledge software offers is as obfuscatory as it is revealing" (2005, 27). To further grasp the arguments of the debate, it is worth highlighting how the advent of Computer Science, with its emphasis on symbolic programming languages, drastically changed the ways in which computing was conceived from the 1950s onwards. Programming and coding practices, prior to the advent of computing languages, were affairs of crafty local conventions and customs that were highly tailored for individual machines across various sites (Nofre et al. 2014, 49). With the growing commercialisation of computing machinery, the concept of programming languages came about as a means to standardise these local conventions and customs, encapsulating them into syntactic and semantic forms that would present traits of both mathematical notations and natural language:

> *The notion of a programming language, which is connected to the idea of universality, became central to this exercise of boundary work that sought to disengage the activity of programming from local conventions, and to transform it into a transcendent and universal body of knowledge. From this endeavour, programming languages and algorithms emerged as epistemic objects stripped of any marks that would associate them with specific hardware.* (Nofre et al. 2014, 66)

The consequence of the advent of "universal" languages was not only that programming acquired a type of "machine independence" (source code able to be built and executed on a variety of machines), but more importantly, it brought about an amassing of linguistic objects written in various "universal" programming languages, and which, in turn, developed an epistemic and discursive life of their own.Programming languages could thus carve out their own computing invariant — a transcendent "island of semantic stability" (66) — by rendering invisible the machine that was once literally in plain sight. It is clear, then, that the universalisation of programming *as* language produced a kind of stratification

and disjunction of computing that cut off the tacit and innate relationship programming had, and indeed still has, with the material, processual and "crafty" aspects of hardware which, consequently, became an invisible and illegible "black box" (Brown and Carr qtd. in Nofre et al. 2014, 54).

Speaking of this disjunction between the legible symbolic programming language and the illegible "black box", Chun posits that, as a result, "software is a functional analog to ideology" (Chun 2005, 43). This analogy between software as an object in itself and as an ideology stems from the fact that software instantiates a strict division and upholds an illusory dialectical logic of cause and effects (input and output) between infrastructure — the obscure and illegible "black box" — and superstructure — manifest and legible programming languages. This rupture speaks to the foreclosure of language over the matter of computing, an operation that totalises the linguistic regime of programming by concealing the totality of its material substrate. Inevitably, then, questions of operations and meaning are (re)claimed by this linguistic regime alone in that it is the only regime capable of lending itself to "objective" interpretations and, in so doing, legitimatises itself. By locating the birth of symbolic programming languages at the grave of material hardware, Computer Science put forth a type of "source" (code) reading of computer programs solely based on human-readability, as opposed to machine-readability, for instance. Addressing this divide, Chun concludes by noting that "because of the histories and gazes [it] erase[s]; and because of the future [it] points toward[s] … [s]oftware has become a commonsense shorthand for culture and hardware a shorthand for nature" (46).

To grasp the potency of Chun's warning, it is important to turn to Galloway's intervention and show how his framings, according to Chun, further highlight the illusory conflation of code (software) and execution (hardware). In his article "Language Wants To Be Overlooked", Galloway (2006) acknowledges that code necessitates a hardware infrastructure in order to function; he writes, "code exists first and foremost as commands issued to a machine. Code essentially has no other reason for being than instructing some machine how to act" (326). We can clearly see how Galloway's concept of code sustains this split between infrastructure (the machine) and superstructure (code as written commands issued to control

the machine) when he famously declares that "code is the only language that is executable" (325). The paramount problem with this conception of command and control, instruction and execution, code and machine is that, as Chun rightly puts it, "[in making] the argument that code is automatically executable, the process of execution itself must not only be erased, but source code also must be conflated with its executable version" (2008, 305). This erasure of execution, by conflating linguistic commands and machine operations, has the corollary of reducing notions of contingent computing events and processes solely to written instructions which command them. In other words, in conflating code and execution one conflates logos with action, explicitly erasing all the problematics, discrepancies and variations action entails (303). Going further with her analysis, as I will discuss in the next section, Chun posits that symbolic code thus becomes law wherein executive, legislative and juridical power coincide to establish a pure state of exception—"code as law as police", where the gap between word and force, and logic and praxis is effectively effaced (2011, 101).

Leaving aside Chun's discussion of the law for now, I would like to emphasise that Galloway's concept of software as language or machine (2006, 327) is solely concerned with the manipulation of symbols. The symbolic order of the command, to put it this way, is put in a prescriptive relationship with its physical "support". The processual and temporal gap existing between the issuing of a command and the return of results is denied any agency whatsoever as the logic of symbols and codes supersedes the one of their entropic medium, a non-processual or eventless notion of execution that seems to be symptomatic of some software oriented media theories. In this regard, both Galloway's and Lev Manovich's (2001) notions of transcoding are worth examining. For Manovich, "to 'transcode' something is to translate it into another format" (47). Similarly, for Galloway, software is a prime exemplar of "technical transcoding without figuration" (2006, 319), where the various "lower level" layers composing the subsystems of the machine (logic gates, registers, etc.) are put into a relation of pure equivalence. As Galloway notes, "one of the outcomes of this perspective is that each layer is technologically related, if not entirely equivalent, to all the other layers" (327).[1]
We thus can clearly see that for both theorists the temporal

and material process by which the machine codes and decodes is completely bracketed since their concept of transcoding solely privileges the outcome of this process, that is, the resulting written format or data structure (323). For Galloway, "there is a privileged moment in which the written becomes purely machinic and back again" (319), for which, then, everything that is machinic ought to be equivalent. While Galloway does not develop his notion of "machinic" further than simply alluding to a complex aggregate of "'lower' symbolic interactions of voltages through logic gates" (319), he does differentiate between conceiving of software as language and conceiving of software as machine (327) in positing that "code is machinic first and linguistic second" (326). While it can be argued that software commands differ from "illocutionary" commands and that software is dissimilar to "speech acts", the point of the current inquiry is to examine the notion of command as such. It aims at problematising how this notion relies on a given symbolic order (arithmetical, logical, algorithmic, legal, machinic, etc.) that substitutes itself for the event that is execution, which, I argue, has nothing to do with symbols alone but rather points elsewhere.

Software as Tyranny

While arguments depicting software as being the "machinic turn" of ideology, in the case of Chun's earlier essays (2005, 2008), or allegory, in the case of Galloway (2006), seem convincing, I intend to look elsewhere to account for the tension between command and execution, word and action. I find it peculiar, to say the least, that the Church-Turing thesis in its physical form, which I believe lurks underneath these discussions about symbolic algorithms and their physical instantiation, is framed in terms of ideology or allegory. Therefore, in what could be considered a bold move, I follow the conviction that "ideology has no importance: what matters is not ideology … but the organisation of power" (Guattari and Lotringer 2009, 37). Thus, rather than seeking inspiration from a critique of ideology, as do Chun and Galloway, I turn to critiques of violence and theories of law and authority that address how concepts of law are enforced through rules, instructions and commands. While Chun's later essay (2011) does turn to a critique of violence, in which she develops the notion of software as law, or code as law, she does not address

and focus on the intricacy of the tandem command-execution in the manner I am suggesting here.[2] To be clear, my aim is not to reify a false idea that symbols are immaterial constructs and thus unreal, or to reduce software to hard-ware, or to argue that infrastructure supersedes superstructure, but rather to theoretically look at how symbolic commands are made to operate in the first place.

According to the mathematical form of the Church-Turing thesis, which is mainly concerned with effective procedures, executability and reliability can be defined as such:

> *Executability: the procedure consists of a finite number of deterministic instructions (i.e. instructions determining a unique next step in the procedure), which have finite and unambiguous specifications commanding the execution of a finite number of primitive operations.*
> *Reliability: when the procedure terminates, the procedure generates the correct value of the function for each argument after a finite number of primitive operations are performed.* (Piccinini 2011, 737)

From these informal descriptions, it is worth examining how a command (instruction) is necessarily active in the sense that it is prescriptive: it requests and constrains action to fulfil the promise of its execution which, in turn, should shed expected effects. Yet the command itself does not act per se, but rather prescribes an action that it, in turn, assesses or judges ("correct value"). A distinction must thus be made between what Jacques Derrida calls "performative" and "constative" (1990, 969), where the former denotes the act of execution and the latter the part of judgement that assesses the effects of the former in light of its initial commanding. In short, the constative, which both definitions of executability and reliability speak to, forms a hermeneutic loop (interpretation, action/execution, interpretation), where the central moment of action — the primitive operation — is at once effected and effaced by interpretation itself.[3] Hence, the constative always presumes the performative, "that is to say [its] essential precipitation, which never proceeds without a certain dissymmetry and some quality of violence" (969).

According to the aforementioned definitions, to do justice to an instruction, a primitive operation has to generate a correct output. However, as Derrida points out, there is no justice of

the performative as such, but only just-ness, that is, performing according to prior conventions, methods, or protocols; the performative, he writes, "cannot be just, in the sense of justice … it always maintains within itself some irruptive violence, it no longer responds to the demands of theoretical rationality" (969). The implicitness and precipitateness of the performative buried within the constative hermeneutic loop speaks, in more general terms, of the conflation of command and execution as discussed in the previous section. What this conflation does, I argue, is to veil the "irrational" violence of the performative that still, necessarily, constitutes the core of the constative. While there may be rules, methods and protocols prescribed by a given command or instruction, the urgency and precipitateness of the performative make it act, nonetheless, "in the night of non-knowledge and non-rule" (967). What the notion of execution harbours then is an act that is at once a "non-knowledge", a "non-rule", a "non-protocol", a "non-method". In other words, the concept of execution points to the reverse side of the law, that is, its necessary primitive exterior.

The rapport between the interior and exterior of the law begs further nuancing. For Derrida, "violence is not exterior to the order of *droit* [law]. It threatens it from within" (989). Yet, as I argued above, the violence of execution stands as a primitive outside to the symbolic order of law; it operates in an inordinately different register as "non-knowledge" and ultimately as "non-law" or "out-law". The order of law, the hermeneutic loop of the constative, as I discussed above, may well comprise a certain placeholder for the moment of action/execution, but it nonetheless is articulated by a totally different language (if actual language there is), which at once prompts execution as such only to efface it after the fact by substituting it with an interpretation of its deciphered effects: a correct instruction for a correct value. Yet the moment of action/execution still remains illegible from the perspective of the constative. The problematic of the symbolic order is its despotic attempt to codify, and therefore foreclose everything by means of substitution, giving it the grounds and monopoly to justify itself as a righteous transcendental order capable of "decreeing to be violent, this time in the sense of an outlaw, anyone who does not recognize it" (987).

There are thus two types of outlaws I want to unearth here: (1) the heretic outlaw that has been judged as such for not recognising the law's order (not following conventions, method, protocol, etc.) and consequently ruled "outside" by decree — an error or "miscomputation" (Piccinini 2007, 505) — and (2) the "autochthon" outlaw that executes and hence founds the constative loop outright, and who therefore stands "outside" the law by necessity — primitive operations. Both vouch for, from the perspective of the law, a sense of legible illegibility, or "foreignness", since they both imply a passage to action as a moment of non-law, a transgression of order.

For Derrida, the moments of action/execution are, by themselves, moments of "mystique". He writes, "[these] moments supposing we can isolate them, are terrifying moments … [They] are themselves, and in their very violence, uninterpretable or indecipherable. That is what I am calling 'mystique'" (1990, 991). What Derrida points to with "uninterpretable" and "indecipherable" is the limit of interpretation as such. Derrida's "mystique" speaks to the event that is execution and how symbolic instructions feign "that of which is in progress" during the event; he writes "[i]t is precisely in this ignorance that the eventness of the event consists, what we naively call its presence" (991). This ignorance [*non-savoir*] as a moment of deferring or drifting of interpretation, as a suspension of the law, is paradoxically equated to its own presence and fosters its own becoming. Law is a spectre during the moment of execution, it is a presence in absence. As a result, execution always exceeds its interpretation or interpretation *tout court*: "[it] is the moment in which the foundation of law remains suspended in the void or over the abyss, suspended by a pure performative act that would not have to answer to or before anyone" (991–3). Thus, the first aforementioned outlaw may well be condemned as heretic — the position of the error or miscomputation — but it nonetheless harbours an eccentricity that exceeds the law and its instruction, an eccentricity that has to answer to or before no one.

Unpacking the term heresy sheds light on what the becoming of the law entails at the moment of action/execution. Etymologically, heresy is derived from the greek αἱρετικός [hairetikos], which, accor-ding to Thayer's Greek-English lexicon, denotes at once "fitted or able to take or choose"

and "schismatic, factious, a follower of the false doctrine". The former sense of the term designates an action (taking or choosing) that, as mentioned above, exceeds interpretation, while the latter denotes an interpretation or judgement as such, which takes place after the fact/action. Both senses thus speak to the becoming of heresy from action to its judgment. As a result, at the moment of action/execution, the becoming of the law coincides with the becoming of heresy. In fact, Derrida tells us, these two becomings are exactly the same. The moment of conservation of the law, by which the hermeneutic loop is instantiated and heretic positions are decreed as such, is the same as the moment of the founding the law. Any position before the law, such as the heretic position, calls for a potential repetition of itself: "[a] position is already iterability, a call for self-conserving repetition" (997). In other words, a position before the law permits and promises, it defies and puts forward a vow to repeat and iterate.

Thus what I have termed the heretic outlaw above is in fact the same conceptual personage as the autochthon outlaw. The figure of the outlaw, then, "would no longer be before the law, rather [it] would be before a law not yet determined, before the law as before a law not existing yet, a law yet to come" (993). Put differently, law's transgression is before the law in the sense that it is an infringement of an existing law yet, at the same time, it points to the potential commencement of another: a proscription becoming prescription. There is no pure founding position of the law as such, only iterations of it, as "conservation in its turn refounds, so that it can conserve what it claims to found" (997). Hence, the heretic position is at once a position of commencement and commandment, a promise of a new order; and "even if the promise is not kept in fact, iterability inscribes the promise as guard in the most irruptive instant of foundation" (997). In this way, the law threatens outlaws, always necessarily, as much as outlaws threaten the law from within, always necessarily. Besides, isn't the heretic position a key position in that it allows for a critique of violence and the law in the first place?

What this amounts to, following Derrida's notion that there is no strict opposition between the conservation and foundation of the law, no position before the law that does not necessarily imply its own iteration, and vice versa, is that the position of the heretic is as forcible as the one of the police, which,

by decree, is supposed to enforce the law. In fact, the terms heretic and police are metonyms that refer to mere positions during the moment of action/execution. As stated above, during this event, the whole order of the law is suspended, interpretation deferred, and "that of which is in progress" during this interval equates to a symbolic void, a moment of "non-law". There can only be symbolic substitutes for what amounts to mere positional acts during execution. At this level of reality, betrayal and enforcement are both in states of becoming, that is, not yet individuated or, rather, judged as such. This is precisely the paradox of law: the insurmountable distance it creates between its prescriptive instructions and its actual "presence-in-action", or, rather, "absence-in-action".

In light of this, Chun's insight of conceiving code as law can be thought of anew. In equating code to law and law to police, thus producing a triad of code as law as police, she writes, "[code] as law as police, like the state of exception, makes executive, legislative and juridical powers coincide. Code as law as police erases the gap between force and writing … in a complementary fashion to the state of exception" (2011, 101). I beg to differ from this perspective and keep the moment of execution as a moment of suspension of the law, a moment of "non-law", a moment of "non-writing", yet a moment of force and intensity, as I argue in the next section. What Derrida shows us, by equating law's conservation and foundation, is that the legislative and executive powers already coincide, albeit in a strange way, and thus, that the state of exception is no exception after all. Yet, the strangeness and clandestinity of the coinciding of the legal and executive comes not from their coinciding as such but more from the fact that law is always necessarily non-present at the moment of action/execution. Derrida talks about the spectre of the law to account for this non-presence, or absence. Thus, Chun's motto of code as law as police can be refactored as code as law as spectre. A position of law is a promise at the moment of execution, a becoming yet to shed the iteration that will "conserve what it claims to found" (Derrida 1990, 997).

Outlaws, Itinerants, and Vagabonds

So far, I have shown that the notion of execution from the perspective of the law merely points to its primitive exterior. What if this perspec-tive were to be reversed? What would

a practice of execution then entail, rather than producing a sequence of instructions? It is not because the law loses its ground and becomes phantom-like that "that of which is in progress" during the moment of execution amounts to nothing, a pure void. There is nothing particularly profound in effecting this reversal of perspective, taking the viewpoint of the heretic outlaw, so to speak. In a sense, that is precisely what Gilbert Simondon's critique of hylomorphism is all about.

To be rather brief at this point, the hylomorphic scheme conceives of both organic or inorganic individuals as engendered by the conjugate of form and matter. One of the classic examples used to illustrate the form-matter dynamic is that of a brick. Simply put, according to the hylomorphic scheme, the production of a brick would be as follows: give a passive lump of clay (potential) a parallelepiped form (actualisation). In other words, a pure form—the parallelepiped—is applied to an indeterminate raw lump of material—the clay—so the lump itself undergoes a transformation and takes the shape of a parallelepiped and, in turn, sheds an individual brick. In this scheme, the form itself is of prime importance since it directs matter in its process of transformation from an undetermined shape to a determined one; put differently, form actualises matter's latent potential. Form is thus the sole source of actualisation that governs the transformation of the lump of raw clay—it determines the indeterminate.

Simondon acknowledges that there is a notion of a genesis, or more precisely of an ontogenesis, involved in hylomorphism, yet it is an "ontogenesis in reverse" (2013, 23).[4] What Simondon does is to reverse this reverse, so to speak, by devising concepts that allow for "knowing the individual through individuation rather than [knowing] individuation from the individual" (24). Instead of conceiving of ontogenesis as a restricted and narrow concept denoting the genesis of a given individual (as hylomorphism does), Simondon conceives of it as a "partial and relative resolution manifesting itself in a system containing potentials and involving a certain incompatibility in relation to itself, incompatibility composed of forces and tension" (25). In a sense, Simondon's notion of individuation stands against the telos of hylomorphism, that is, against erecting the Individual as a privileged origin (form) and finality (brick). The individual he puts forth is thus grasped as a relative

reality, never fully realised, and the process of individuation perpetual rather than transitive.

The tension and contrasts between the form-matter couple of hylomorphism are even more clearly and vividly exposed by the discourse on the instruction-execution divide I have critiqued. As argued earlier, positions before the law are always mere potentials at the moment of action/execution, and thus the law itself is always in a process of becoming rather than final, as it can never truly be founded once and for all. Because of this problem of origin and finality of the law—its incompatibility in relation to itself—a rapport can be drawn here with Simondon's critique of hylomorphism. For Simondon, the technical operation that "imposes a form to a passive and indeterminate material" is not only a phantom-like operation, but more importantly is tyrannical. He writes:

> *[It] is not only an abstract operation considered by the spectator that sees what comes in and out of the workshop without knowing what the actual elaboration is. It is essentially an operation commanded by a free man [of the Republic] and executed by the slave … The true passivity of matter is its abstract availability under the given order that others will execute.* (51)

Simondon's image of the spectator (or should I say spectre) who remains outside of the workshop is most evocative here: the workshop is hylomorphism's own "outside"—"[t]he hylomorphic scheme corresponds to the knowledge of a man who remains outside of the workshop and only considers what comes in and what comes out of it" (46). The same outside perspective could be said of a programmer who considers digital execution solely from his computer's command line. His remark of the situation of the slave can be linked to the one of the outlaws and the heretics depicted in the previous section. The hylomorphic scheme, like that of the law, is necessarily founded on primitive external entities that it appropriates by despotic means. Yet, in his treatise, Simondon argues that to truly grasp the process of form-taking, such as the moulding of a brick, "it is not enough to enter the workshop and work with the artisan: one should enter the mould itself to follow the operation of form taking at different levels of magnitude of physical reality" (2013, 46).

Moving from question of law to questions of science, Gilles Deleuze and Félix Guattari engage with notions of interiority and exteriority of the law, and frame the aforementioned

perspectival reverses in these terms:

> *A distinction must be made between two types of science, or scientific procedures: one consists in "reproducing," the other in "following." The first involves reproduction, iteration and reiteration; the other, involving itineration, is the sum of the itinerant, ambulant sciences … following is not at all the same thing as reproducing, and one never follows in order to reproduce … Reproducing implies the permanence of a fixed point of view that is external to what is reproduced: watching the flow from the bank. But following is something different from the ideal of reproduction. Not better, just different. One is obliged to follow when one is in search of the "singularities" of a matter, or rather of a material, and not out to discover a form.* (Deleuze and Guattari 1987, 372)

What thus becomes clear is how software as law institutes this transcendental fixed point of view — the aforementioned constative loop — by isolating, stratifying, discretising, categorising and foreclosing the spatiotemporal continuum the process of execution articulates. Computer Science, as the science that legislates, is thus responsible for abstracting moments and locales from this continuum and structuring logical concepts and categories out of these abstractions. Yet the theorematic coordinates such a science puts forth are based on various spatiotemporal cuts and erasures; in other words, from a spatiotemporal continuum a logical series is extracted that, as a result, features as many forbidden zones or vanishing points as there are terms in the series. The theorematic power of Computer Science comes from its given authority in decreeing laws and concepts that produce the sacrosanct apodictic apparatus of empty repetition — that is, the repetition of the same and the similar. Without this apodictic apparatus, Computer Science would be destined to follow the progression of a given spatiotemporal phenomenon at ground zero and thus lose its transcendental, and fixed, point of view.

Execution asks to be followed, not iterated. Practices of execution entice an itineration within the residual outside of software, that is, an itineration at ground level where the theorematic coordinates of software are projected on the ground. In order to account for the spatiotemporal individuation of the event of execution proper, one has to step out of

Computer Science's apodictic apparatus of categorisation and traverse the zones of indeterminacy this apparatus constructs. To follow is to cross the interstice's in-between states, in-between commands and in-between rules and laws. It is to traverse these moments of non-law, non-knowledge, non-rule, non-protocol, non-method; in short, to follow is to transgress the imposed dominant order and, in so doing, to problematise the rationale behind its disposition of minoring an outside. The reason I have, in the previous section, focused on the notion of outlaw and positions of heresy before the law is to call attention to power relations inherent in this process of minoring. The problem of execution concerns the domain of epistemology as well as that of work and labour, be it human or non-human. Not only does the creation of a residual outside raise questions of legibility and illegibility in terms of knowledge, but further, it promulgates certain types of social practices and work hierarchies that perpetuate types of despotism and tyranny based on certain valuations of work and systems of visibility and invisibility based on this very outside.[5]

While one may be lured into looking for notions of execution in Computer Science books or to practice execution from his/her computer's command line, I suggest one has to look elsewhere and engage differently with code and circuitry to truly grasp and follow the event that is execution. As short concluding remark, I would like to suggest that luckily, another type of heretic "science" of execution, or rather a practice, already exists that is not usually featured in Computer Science literature per se but is, nonetheless, always and necessarily performed when producing a piece of hardware or a piece of software — that is the practice of *debugging*. True "occult science", debugging requires one to follow the thread of execution of a given program, that is, to follow the itineration and vagabonding of signs and signals within the architecture of a given machine at a given time. A bug, error, failure, or miscomputation necessarily begs to be followed. It is an event itself, or, rather, speaks to the individuation of execution in and for itself. It requires that the illusory disjunction or stratification of instruction and execution, signs and matter, and the discretised dynamics this disjunction puts forth be suspended and problematised. What the practice of debugging highlights is the fragile conjunction of signs and signals in focusing on the technical operations that mediates them in time and space.

To debug is to open bare the foreclosure of the aforementioned symbolic order of the law and enter Simondon's mould, so to speak: to observe and intervene during the event that links the two technological half-chains of the sign and the signals, the opcode and the dipole.

Debugging, as liminal and vagabond science, as well as an effective practice of execution, is potent in problematising and debunking the tyrannic minoring of an outside some Computer Science concepts necessarily produce, and, in turn, that some Software Studies discourses reproduce. After all, debugging is about problems and problematisation, may it be of a piece of machinery or a piece of theory. In fact, problematics is its only mode of operation. There are no software stacks nor interfaces along the path of the vagabond outlaw, only curious spectres.

Notes

1. The same emphasis on the symbolic outcome of an execution can be said of Galloway's equating two quadratic equations written in a "high-level" and "low-level" programming languages (2006, 319). Surely both equations, expressed differently, shed the same numerical solution, yet their respective technical unfolding during execution are nothing but equal, as Chun points out (2008, 306–7).

2. See the present collection's contribution "RuntimeException() — Critique of Software Violence" by Geoff Cox, who also discusses software in terms of violence, in a different, albeit complementary, way to this chapter.

3. The notion of interpretation here does not necessarily denotes a semantic interpretation as a comprehension of the meaning of a command or result in a mathematical or linguistic sense. The loop structure I am describing here holds for purely mechanistic conceptions of computing such as the one put forth by Piccinini (2008, 2007). Interpretation, in this case, thus relates to notions of internal semantics rather than external ones (Piccinini 2008, 214–5).

4. All citations from Simondon are my translations.

5. See Linda Hilfling Ritasdatter's contribution "BUGS IN THE WAR ROOM — Economies and /of Execution" in the present collection, where she addresses on question software maintenance and labour in terms of neo-colonial hegemony.

References

Chun, Wendy Hui Kyong. 2005. "On Software, or the Persistence of Visual Knowledge." *Grey Room* 18: 26–51.

——. 2008. "On 'Sourcery,' or Code as Fetish." *Configurations* 16 (3): 299–324.

——. 2011. "Crisis, Crisis, Crisis, or Sovereignty and Networks." *Theory, Culture & Society* 28 (6): 91–112.

Deleuze, Gilles, and Félix Guattari. 1987. *A Thousand Plateaus: Capitalism and Schizophrenia.* Minneapolis: University of Minnesota Press.

Derrida, Jacques. 1989. "Force De Loi: Le Fondement Mystique De L'Autorité / Deconstruction and the Possibility of Justice." *Cardozo Law Review* 11: 920–1046.

Fuller, Matthew. 2006. "Software Studies Workshop." Piet Zwart Institute — Software Studies Workshop. http://web.archive.org/web/20100327185154/http://pzwart.wdka.hro.nl/mdr/Seminars2/softstudworkshop.

Galloway, Alexander R. 2006. "Language Wants To Be Overlooked: On Software and Ideology." *Journal of Visual Culture* 5 (3): 315–31.

Guattari, Félix, and Sylvère Lotringer. 2009. *Chaosophy: Texts and Interviews 1972-1977*. Semiotext(e) Foreign Agents Series. Los Angeles, CA: Semiotext(e).

Manovich, Lev. 2002. *The Language of New Media*. MIT Press ed. Leonardo. Cambridge, MA: MIT Press.

Nofre, David, Mark Priestley, and Gerard Alberts. 2014. "When Technology Became Language: The Origins of the Linguistic Conception of Computer Programming, 1950–1960." *Technology and Culture* 55 (1): 40–75.

Piccinini, Gualtiero. 2007. "Computing Mechanisms." *Philosophy of Science* 74 (4): 501–26.

——. 2008. "Computation without Representation." *Philosophical Studies* 137 (2): 205–41.

——. 2011. "The Physical Church — Turing Thesis: Modest or Bold?" *The British Journal for the Philosophy of Science* 62 (4): 733–69.

Simondon, Gilbert. 2005. *L'individuation à la lumière des notions de forme et d'information*. Krisis. Grenoble: Millon.