# Hardware and Software Bugs in the Legal System

Alec Thompson, Ryan Gibb

November 2025

## 1 Introduction

Legal systems are the largest social institutions run exclusively on natural language. Two devices make their scale possible:

1. Text can serve as a legal source regardless of human cognition. There are swathes of legal rules that the entire legal profession is unaware of, but are nevertheless law according to the system's meta (source) rules.
2. Legal systems contain rules which force judges to apply the law regardless of the socio-political consequence. Whilst these rules are fuzzy, they are not infinitely flexible: judges, especially in lower-tier courts, are part of a much larger machine.

These devices separate judicial desire from legal outcome and make scale possible. Equally, they raise the possibility of exploits within the legal system. The quintessential legal hacker finds a rule in some authoritative law book - potentially one no-one had considered before - and uses it against the system. In the words of Bruce Schneier :

> Legal code isn't the same as computer code, but it's a series of rules with inputs and outputs. And just like computer code, legal code has bugs. And some of those bugs are also vulnerabilities. And some of those vulnerabilities can be exploited[1]

The purpose of this short article is to consider how closely this analogy holds. It begins with a brief comparison of legal and cyber ontologies. It then discusses various legal hardware and software bugs and the possibility of 'legal

[1] https://www.schneier.com/blog/archives/2023/01/kevin-mitnick-hacked-california-law-in-1983.html

viruses' before exploring several 'legal hacks.' Whilst the modern legal system is vulnerable to some exploitation, the pervasive human presence in its operation restricts its vulnerability. This position will change if the legal system becomes more automated, taking human beings out of the loop and rendering the input-output process unsupervised.

## 2 Cyber-legal Ontology

Legal systems are closest to computer systems when used to process information. To understand how, imagine ordering a kit to build a legal system.

You open the box and find many items: pens, paper, books, servers, computers, trains, strange costumes, various buildings, and many humans who claim to be 'legal experts.' How does it all fit together? The first step is to determine which function of the legal system you want to prioritize. Let's start small and design a module for resolving commercial disputes between private citizens quickly, predictably, consistently, and at scale.

In terms of Von Neumann Architecture, the **inputs** are the facts of trial, the **output** legal concepts; and the **CPU** and **Control Unit** are contained within the lawyers and judges making decisions about which processes to run. The conventions of legal reasoning - permitted logical arguments and analogies - supply the **ARU**, whilst the substantive rules themselves are the **programs**. **Memory** in the legal system is stored in inorganic storage (paper and silicon) which behaves like **ROM**, and organic storage (the memory of human lawyers) which behaves more like **RAM**. In theory, one could apply the Ancient Roman Legal Rules using their formal legal texts (ROM), but one's outcomes would differ due to the lack of contemporary tacit knowledge (RAM).

The most common legal program (known as a 'conditional program') is to use rules which label facts using the following operator: IF [fact-pattern] THEN [legal concept]. These concepts are then used to direct the court towards an outcome to the dispute, often with specific reference to the remedy, such as 'divorce', 'compensation', or 'imprisonment.'

We can further divide legal programs by their function. 'Substantive law' involves fact-patterns generated outside the legal system, and serves as the main interface with other societal sub-systems. Conversely, 'procedural law' governs facts emerging from within the legal system and acts as a **runtime environment**. Both sets of rules are Turing Complete and, in terms of paradigm, resemble functional programming languages.[2]

[2]Attempts to convert legal rules into code have already proceeded on this basis, specifically CATALA.

Like digital code, legal code varies in quality. Clean legal code is clear and logically consistent, matching legal concepts to facts in a consistent manner. The most advanced code is organized in hierarchical ('nested') concepts: **specific performance** is a remedy for a **breach of contract** which requires a **valid contract**, determined by **contract law** which is a subset of **the law of obligations**, which in turn is a subset of **private law**. These nested rules are extremely detailed and complex, with each category implying various features about its contents.

Today, most conditional programs are stored off-head in legal texts. [3] Scalability and volume require as much cognitive off-loading as possible. Most law is stored as inorganic text, and the legal status of that text is guaranteed by other inorganic texts.[4] Head-storage is typically used to cache highly fluid, constantly shifting rules. For example, the tacit sense of what a judge or jury will regard as 'foreseeable' or 'grievous bodily harm.' Although temporary, it is essential: no matter how much time and intelligence you have, without RAM you cannot execute programs correctly.

The hardware of the legal system are 'legal institutions', containing both organic and inorganic components. In daily operation, the former is more important. Legal experts both store legal rules in local memory and carry out legal operations to resolve cases. These experts communicate and coordinate their functions using emails, judgments, conversations, which act as the legal version of **Databus**. Ensuring these 'processors' work is difficult: human operators can be lazy, tired, and disobedient. In general, their breakdown rate is moderated by the surrounding 'rule-following tradition'; which in turn is sustained by social-selection, education, strange customs, costumes, and buildings, designed to produce a sense of awe and allegiance.

Human beings are also required to resolve ambiguous cases and update the system. When there is a legal gap - or where a rule needs to be changed - the human operators must create new legal texts. The split between agile and waterfall coding approaches is reproduced in law-making styles. Editing the law-base can be fast-fail and iterative (common law precedent; law is made, then 'worked pure' in practical application), or deliberative and prospective (civil law legislation; law is planned, written, and rolled out; edits require new cycles of planning). Legal doctrine also demonstrates the characteristics of legacy code and tech debt. Leaving old law on the books requires workarounds that, in the long run, often produce higher cost and complexity than a timely patch.

[3] One cannot destroy an Act of Parliament by destroying a specific scroll, book, or group of judges. It exists as information and will continue existing as 'the law' provided a record remains.

[4] For example, the rules defining a valid Act of Parliament are found... in an act of parliament; in England, the rules of precedent are found in... legal precedents. All that is held in collective consensus is a vague gesture to the sources themselves, which then impose structure internally

Despite this complexity, the 'software' of the legal system is managed very informally. There is no manual of best-practice for developing the law, nor is there a deep understanding of how legal rules work. Legal philosophers can still barely agree on what the correct description of a 'law' is! This helps explain the many ways legal systems fail *as a tool for information management.*

# 3 Hardware Bugs

A typical hardware failure is human cognitive error, such as forgetfulness, stupidity, and various cognitive biases related to the innumeracy of the judiciary. A famous example is a group of South African judges who accidentally awarded over 100 percent in compensatory damages through incompetent addition. These are analogous to ordinary malfunctions, Perhaps surprisingly, these bugs also arise in the case of formal legal texts. For example, the land registry in England misfiled its case reports on its online website. Lawyers could not find these cases and, as a result, reasonably assumed there were no binding precedents. As a result, judges decided cases inconsistently, leading to three different case lines on the same point.

More obliquely, the reporting process in England is not 1:1. Many cases are left unreported and someone has to choose which cases are selected. There was a famous case of a 19th century reporter hiding cases which he disagreed with in a drawer titled 'bad law.' More recently, a number of 1950s House of Lords decisions were suppressed due to political sensitivity and remain unreported despite their binding force.

Finally, a fuzzier class of 'hardware bugs' arises in organic storage. From the perspective of a system designer, the rapid and large expansion of extra-textual law, disrupting the consistent application of rules and replacing it with local justice, is a bug. A clear example is 1950s Brazilian legal culture known as the 'Jeito'. An example of Jeito is deliberately mis-entering a job in an immigration application to expedite the process. If enough of these bodges are engrafted on to the formal rules, the legal system ceases to process information - or, in juristic terms - becomes 'uncertain.'

Treating the legal system this way hints towards a more ambitious legal science. Legal reform could be treated as a matter of performance enhancement. How can we make information retrieval systems faster, more accessible, and cheaper? Methods of storage could be diversified: we could model, for example, the correct balance between textual and extra-textual storage. It will be increasingly important to think in terms of hardware as the legal system is digitized. The right balance of human-text-machine in the system can be reconceptualized as a matter of hardware design: what is the optimal arrangement of media and processing? There is currently no easy way to carry out hardware diagnostics

in the legal system: we lack both the concepts and the data to do so.

# 4 Software Bugs

Traditionally, legal software flaws are the domain of Legal Science. It is the ancient role of the legal scientist to close legal loopholes and prevent unwanted blips in the mutual interaction of legal rules. A recent, somewhat esoteric example is the interaction between privity and commercial subsidiarity. Say parent company **A** makes a contract with landlord **B** in which **B** is to let some land for use by **A**'s subsidiary **C**. If B breaches the contract, **A** has suffered no loss, whilst **C**, who has suffered loss, is not part of the contract so cannot sue according to the rules of privity. This is known as a 'legal black hole' and various bodges are used to evade it. Like most bodges, they have not stuck, and the scenario continues to reach the top appellate court. [5]

Updating and cleaning up the legal codebase is essential because, like computer code, law deprecates. Unfortunately, those responsible - legal scientists - use methods which date from the 18th century and can no longer manage the complexity of the modern state. Lawyers have a fuzzy, informal idea of how rules behave. They are like technicians maintaining an immense machine they did not design or build, reduced to knacks, rituals, and obscure maxims.

Legal-engineers must find ways to model their systems at multiple levels of abstraction. They should map legal areas; their rate of change and development; the distribution of compensation being awarded; areas of stability and instability; and the functioning of the runtime environment. This has to come alongside knowledge of the 'hardware' of the legal system. They also need to actively remove complexity from the system. This means clearing out old code and investing in long-term updates to the law-base. Judges and lawyers have a blurry conception of local complexity, and certainly no sense of a general, cross-system policy of reducing it. Remarkably, law-makers also lack the software engineering concepts of cohesion and coupling. Links between different legal areas are allowed to multiply randomly with no awareness of the emerging structure.

# 5 Cyber-Legal Viruses

What is a 'cyber-legal' virus? A set of legal rules designed to produce results contrary to the purpose or goals of the system operators. [6] In practice, such

[5] https://www.supremecourt.uk/cases/uksc-2025-0081

[6] My definition of 'pure execution' alone is narrow; I exclude straightforward legal reform which is designed to produce an obvious, intended result. This is just updating the system.

viruses are exceptionally rare. 'Code' cannot be run on the legal system secretly, even if massive quantities of information can be held in inorganic storage. The requirement for human cognitive contact makes it almost impossible for a virus to run undetected.

The ideal theoretical 'legal virus' would work like this: an adverse actor introduces into a legal system - either through lobbying or a precedent - a series of unobjectionable laws. These laws then trigger a series of chain-reactions at a wide enough scale that the legal system's operators are unable to detect the final destination before it becomes inevitable. Designing such a legal virus is currently impossible because no-one - virus developers nor lawyers - understands the legal system deeply enough. Fortunately for lawyers, there appears to be an asymmetry insofar as designing legal viruses is substantially harder than solving even the trickiest doctrinal puzzles. All existing viruses imperfectly approximate this ideal.

In 18th century England, several courts ran concurrently before fusing, and legal doctrines from one sometimes infiltrated and threatened to alter the outcomes of the others. The complex nature of legal rules permits this sort of 'virus.' Lawyers may be well-aware of Rule A and its application; that does not mean they are aware of how it operates with Rules B, C, D, and the emergent effects from their mutual interaction. In this way, large sets of foreign rules - referred to as 'legal irritants' by sociologists - can lodge in a legal system and produce awkwardness and undesirable outcomes for remarkably long periods.

Another possible example is the introduction of fake legal sources into the legal system. The source would have to remain undetected in the system sufficiently long as to become fully enmeshed in other rules, such that undoing it would be practically impossible. Such a 'legal virus' came into existence in the 15th century when 'Fake Statutes' were accidentally introduced from personal collections into more official statute collections. By the time this was realized, the fake statutes had percolated widely across the system.

Finally, the German sociologist Niklas Luhmann believes that a real, widespread virus currently infects most legal systems: morality. According to his 'Systems Theory', every societal sub-system has its own algorithm and binary code. The legal system uses 'conditional programs' (IF-THEN) and codes situations 'legal/illegal.' Morality threatens to break this algorithm down, replacing the conditional program with (WHATEVER IS MORAL), and the code with moral/immoral. The results are systemic breakdown: the legal system is no longer predictable, nor can it be easily integrated into other systems (POLITICS; NEWS; ECONOMY; RELIGION).

---

My definition of 'purposes' is focused primarily on the legal system's operators themselves: the courts and legislature

# 6 Hacking a Legal System

Is it possible to hack a legal system? Unlike a virus, legal hacking could cover a range of adversarial activities. Abusing legal loopholes [7] are the quintessential hack and have existed for hundreds of years. In the 15th and 16th centuries lawyers evaded entails and feudal taxes using trusts, just as 'tax-efficiency' specialists do today. Nonetheless, more interesting legal hacks can be found.

One is to take advantage of procedural glitches. Until quite recently, if you filed a lawsuit involving parties with multiple nationalities in a particular national court, any hearing on the transfer of jurisdiction had to occur in that country's courts. This led to the so-called 'Italian Torpedo'[8], whereby a party rapidly filed a claim in a country with enormous procedural delays (e.g., Italy), thereby paralysing its progress.

Another hack is broadly called 'lawfare.' [9] The best known example is the PRC's construction of artificial islands in the South China sea to generate territorial claims under international law; for similar reasons, the PRC has also set up fake research facilities across Antarctica. At a smaller scale, lawfare typically involves the abuse of procedural cost - a sort of DDOS attack - whereby an enemy is buried in paperwork and complex, ongoing litigation.

Both lawfare and procedural exploits take advantage of the relative 'blindness' of the legal system: courts typically refuse to take notice of their own enormous delays or the wider political effects of their decisions. The cognitive screening necessary for predictability and scale also makes the legal system vulnerable.

A third sort of hack involves path-dependency in legal and financial systems. In 2006, a Calabrian mafia organisation securitised its extortion business and sold the ensuing debt packages - constructed with the help of their lawyers - in the bond market. [10] By the time it was detected, these packages had percolated so widely that unwinding them became impossible. This is possible because legal *instruments* are often operated on - i.e., bought, sold, packaged, securitised - without close inspection of their contents. The same rarely happens with legal *rules* because, by their nature, rules need high cognitive contact to be applied.

[7] https://www.schneier.com/blog/archives/2023/01/kevin-mitnick-hacked-california-law-in-1983.html

[8] https://en.wikipedia.org/wiki/Italian_torpedo

[9] https://en.wikipedia.org/wiki/Lawfare

[10] https://www.ft.com/content/bcebd77c-057b-4fd0-bd99-b97e0e559455

# 7 A Digital Legal System

Natural language legal systems are inherently difficult to scale. Human beings have limited storage space and can only operate for a few hours a day before getting tired. They also have sharp limits in their input and communication speed. Yet society - especially its economy and technology - continues to accelerate. The legal system can no longer keep up.

Large language models are extremely well-suited to replacing lawyers. Every year, the ratio of organic to inorganic storage moves in favour of the latter. Only the former offers any obstacle to automation. Storage has already been taken off-head; doing the same for legal Processing will make the system fully scalable. It will also open up new avenues of exploitation and malfunction. New institutions, norms, and ontologies will be needed if the legal system is to be brought into the 21st century.