When Law is Code

Author : James Grimmelmann

Date : July 31, 2024

Sarah B. Lawsky, Coding the Code: Catala and Computationally Accessible Tax Law, 75 **SMU L. Rev.** 535 (2022).

Sarah B. Lawsky's *Coding the Code: Catala and Computationally Accessible Tax Law* offers an exceptionally thoughtful perspective on the automation of legal rules. It provides not just a nuanced analysis of the consequences of translating legal doctrines into computer programs (something many other scholars have done), but also a tutorial in *how* to do so effectively, with fidelity to the internal structure of law and humility about what computers do and don't do well.

Coding the Code builds on Lawsky's previous work on formal logic and its advantages for statutory interpretation. (Formal logic, sometimes called "symbolic" or "mathematical" logic, involves the precise and rigorous analysis of symbolic expressions representing arguments, such as "p & ¬q" to mean "p is true and q is not true".) In her 2017 *A Logic for Statutes*, she observed that many statutory provisions have a characteristic structure: rules subject to exceptions. A typical rule says that WHEN certain conditions are satisfied, THEN certain consequences follow, UNLESS one of several exceptions applies. Exceptions have exceptions of their own: interest payments are deductible, unless they are personal, unless they are mortgage payments.

Lawsky's great insight about law and logic is that this characteristic structure of nested exceptions is most naturally modeled using a branch of formal logic called "default logic." Default logic, unlike standard "monotonic logic," allows for tentative conclusions. On the basis of what I know now, this is a nondeductible personal interest payment, but let me investigate further, and oh, I see that this is qualified residence interest, so I am withdrawing my tentative conclusion and replacing it with another tentative conclusion that the payment is deductible. And so on, until there are no more clauses of the statute to check, no more exceptions to explore, and the most recent tentative conclusion becomes a definitive one. It is a process of successive refinement, converging on certainty. Monotonic logic, by sharp contrast, requires ruling out all possibilities before drawing a conclusion, which remains valid for all time once drawn.

Default logic is not more powerful than standard logic, but for some kinds of reasoning it is cleaner, and Lawsky's point is that the back-and-forth of exceptions and subexceptions in statutory analysis maps naturally onto default logic's structure of defaults and defeats. A formal logician applying a default logic's inference rules follows a reasoning process that naturally corresponds to the reasoning process followed by a lawyer working through a statute.

Default logic is also a good tool for programming. (It is a formal logic, after all.) Once a human has translated a natural-language statute into a formal-logic representation, it becomes possible to reason automatically and algorithmically about the statute and how it treats various fact patterns. In 2021, a trio of computer scientists—Denis Merigoux, Nicolas Chataing, and Jonathan Protzenko—published *Catala: A Programming Language for the Law*, which turned Lawsky's default-logic analysis of statutes from an abstract formalism useful for pencil-and-paper analysis into a concrete implementation useful for programming. (Lawsky herself is now a co-designer of the Catala language.)

As someone who sank years of his life into programming a body of law, I can say that Catala is the cleanest and most broadly useful advance towards making law programmable I have ever seen. A 29-page paper filled with equations and code blocks may be quite daunting (our research group took several weeks to read through the formalisms together in detail), but the basic idea of what it does is beautifully simple and clear. Catala allows a programmer to write the way that lawyers think: by laying out rules that trigger consequences, together with the exceptions that can prevent the consequences from happening.

Tax law in particular has two advantages that make it well-suited for this kind of formalization. First, it depends on—and attempts to produce—clear and determinate answers. Everything comes down to, or should, a specific amount due. And second, much of tax law is what a programmer would "declarative" rather than "imperative"; instead of telling people what to do, it describes the consequences of what they have already done. The Internal Revenue Code, as Lawsky has shown, comprises declarative provisions that are particularly clean to implement in a Catala-style language that uses defaults and exceptions. Taking advantage of this affinity between tax law and programming languages, Merigoux and Protzenko, along with Raphaël Monat, have been developed a toolchain to help the French tax authority modernize its antiquated systems. Their work puts directly into practice Lawsky-ian ideas about the value of clean formal reasoning to improve the application of tax statutes.

Coding the Code is in many ways the *summa* of Lawsky's project over the last decade. After an accessible introduction to default logic, other portions of *Coding the Code* draw on what Lawsky has been up to lately: actually using Catala to code up tax law. She and her collaborators have approached the task with humility and care—virtues that Lawsky describes the need for in interdisciplinary collaborations in the recently-published *Computational Law and Epistemic Trespassing*. One approach they use is "pair programming": a lawyer and a computer scientist sit side by side at one computer, discussing a statutory section and making sure that they agree on its translation into code. Another is "literate programming", in which code is interwoven with comments that document what each part of it is doing. For statutory translations, these comments can include the statutory text itself, making the isomorphism between specification (the statute) and implementation (the code) wholly explicit. Neither pair programming nor literate programming directly affects what the code-ified version of the law does; instead, they are tools to make sure that the people who do the translation do so faithfully, in a way that others who come later can recognize as correct. (Lawrence Lessig, patron saint of code-as-law, would approve.)

Coding the Code, like the rest of Lawsky's work, stands out in two ways. First, she is actively making it happen, using her insights as a legal scholar and logician to push forward the state of the art. Her Lawsky Practice Problems site—a hand-coded open source app that can generate as many tax exercises as students have the patience to work through—is a pedagogical gem, because it matches the computer science under the hood to the structure of the legal problem. (Her Teaching Algorithms and Algorithms for Teaching documents the app and why it works the way it does.)

Second, Lawsky's claims about the broader consequences of formal approaches are grounded in a nuanced understanding of what these formal approaches do well and what they do not. Sometimes formalization leads to insight; her recent Reasoning with Formalized Statutes shows how coding up a statute section can reveal unexpected edge cases and drafting mistakes. At other times, formalization is hiding in plain sight. As she observes in 2020's Form as Formalization, the IRS already walks taxpayers through tax algorithms; its forms provide step-by-step instruction for making tax computations. In every case, Lawsky links carefully links her systemic claims to specific doctrinal examples. She shows not that computational law will change everything, but rather that it is already changing some things, in ways large and small.

It is unusual for an established law professor to go back to school for a PhD. In philosophy. With a dissertation on formal logic. But *Coding the Code*, published five years after Lawsky submitted her (highly technical) thesis, shows the great value for legal scholars of the approach she developed in her PhD. It refines her distinctive approach to statutory analysis—which mixes careful legal reading with technical tools from formal logic and computer science—in a way that has great potential to help other lawyers and legal scholars be more precise about what tax laws say. All they need to do is talk to computer scientists, and Lawsky provides a roadmap for how. There is no epistemic trespassing in Sarah Lawsky's work. Everywhere she goes, she is a welcomed guest.

Cite as: James Grimmelmann, *When Law is Code*, JOTWELL (July 31, 2024) (reviewing Sarah B. Lawsky, *Coding the Code: Catala and Computationally Accessible Tax Law*, 75 **SMU L. Rev.** 535 (2022)), https://cyber.jotwell.com/when-law-is-code/.