Northwestern
PRITZKER SCHOOL OF LAW

NORTHWESTERN UNIVERSITY PRITZKER SCHOOL OF LAW
PUBLIC LAW AND LEGAL THEORY SERIES • NO. 22-32

Coding the Code: Catala and Computationally Accessible Tax Law

75 SMU L. Rev. 535 (2022)

Sarah Lawsky
Northwestern University Pritzker School of Law

SMU | Dedman School of Law

SMU Law Review

Volume 75 | Issue 3 Article 4

2022

Coding the Code: Catala and Computationally Accessible Tax Law

Sarah Lawsky
Northwestern University, Pritzker School of Law

CODING THE CODE: CATALA AND COMPUTATIONALLY ACCESSIBLE TAX LAW

*Sarah B. Lawsky**

ABSTRACT

"Caligula posted the tax laws in such fine print and so high that his subjects could not read them . . . . That's not a good idea, we can all agree. How can citizens comply with what they can't see? And how can anyone assess the tax collector's exercise of power in that setting?"[1]

TABLE OF CONTENTS

---

https://doi.org/10.25172/smulr.75.3.4.

* Stanford Clinton Sr. and Zylpha Kilbride Clinton Research Professor of Law, Northwestern Pritzker School of Law. Thanks first to rest of the Catala team: Denis Merigoux, Liane Huttner, and Jonathan Protzenko. Thanks also to Joshua Blank, Michelle Falkoff, Jacob Goldin, Ellen Lawsky, Mattea Lawsky, Ajay Mehrotra, Jim Speta, Dennis Ventry, and participants in the 2021 Seminar Series at the Future of Law Project, University of Pittsburgh School of Law, the 2021–2022 CS+Law Workshop, and the 2021–2022 Northwestern Pritzker School of Law Faculty workshop for helpful conversations and comments on earlier drafts.

1. Summa Holdings, Inc., v. Comm'r, 848 F.3d 779, 781 (6th Cir. 2017) (citing SUETONIUS, THE TWELVE CAESARS 220 (A.S. Kline trans.) (2010) ("He levied new and unprecedented taxes, at first through tax-collectors, and then, because the amounts were so vast, through the centurions and tribunes of the Praetorian Guard. No group of commodities or individuals escaped a levy of some kind . . . . Many offences were committed in ignorance of the law, since the new taxes were decreed but nothing was published to the public. After urgent representations, Caligula finally had a notice posted, in an awkward spot in tiny letters to make it hard to copy.")).

I. INTRODUCTION

WHILE lawyers, computer scientists, law professors, and technologists argue over whether formalizing law by translating it into computer code is practical, effective, useful, or desirable, tax law has been formalized for years.[2] Indeed, administering modern tax law would be essentially impossible without this formalization. Tax law, both in the United States and other countries, is complex and lends itself to being translated into formulas and algorithms—and tax authorities have done just this. Tax forms are formalizations.[3] Tax authorities enforce tax law using computational versions of the law.[4] Taxpayers pay for computer programs that prepare returns based on inputs from taxpayers.[5] The question is not whether to formalize tax law but rather how to formalize it.

The question of how to formalize has two parts: first, how the formalization would be accomplished, and second, the best version of the formalization itself. This Article takes as a given that the formalization will, at least for now, be accomplished by humans formalizing the Internal Revenue Code (Code), as opposed to computers analyzing large amounts of text to arrive at automated formalizations. While existing formalizations in use are often implemented by computers, they were not themselves created by computers. Indeed, many of the formalizations long predate the adoption of automated computing. Even today, automated formalizations such as natural language processing have yet to succeed in translating tax law into computer code or other formalizations.[6] Thus, any

2. To "formalize" something is to represent it in symbols using logical connectives or similar. *See generally* James G. Williams, *On the Formalization of Semantic Conventions*, 55 J. SYMBOLIC LOGIC 220, 220 (1990) (outlining "[v]arious formalization techniques . . . used with software systems based on traditional mathematical logic"). Computer code is a type of formalization that is accessible by computers, a computationally accessible formalization. *See id.*

3. Tax forms are formalizations of tax law, and in the United States alone, forms date back at least to the 1800s. *See* Sarah B. Lawsky, *Form as Formalization*, 16 OHIO ST. TECH. L.J. 114, 121 (2020); INTERNAL REVENUE SERV., DEP'T OF THE TREASURY, FORM 1040 (1864), https://www.irs.gov/pub/irs-prior/f1040—1864.pdf [https://perma.cc/9Z6L-8CNW].

4. *See, e.g.*, Cyrille Chausson, *France Opens the Source Code of Tax and Benefits Calculators to Increase Transparency*, JOINUP (Nov. 20, 2016), https://joinup.ec.europa.eu/collection/egovernment/document/france-opens-source-code-tax-and-benefits-calculators-increase-transparency [https://perma.cc/X2FP-PJDA] (describing source code used to calculate income tax in France); INTERNAL REVENUE SERV., PUBL'N. 5336: IRS INTEGRATED MODERNIZATION BUSINESS PLAN (2019), https://www.irs.gov/pub/irs-pdf/p5336.pdf [https://perma.cc/Q2FF-QXMV] (describing significant use of computation to monitor compliance with tax law).

5. The TurboTax tax preparation program is an example. *See* INTUIT TURBOTAX, https://turbotax.intuit.com [https://perma.cc/M8MB-XGV7].

6. *See, e.g.*, Nils Holzenberger, Andrew Blair-Stanek & Benjamin Van Durne, *A Dataset for Statutory Reasoning in Tax Law Entailment and Question Answering*, *in* 2D NAT. LEGAL LANGUAGE PROCESSING WORKSHOP (2020) (trying to remedy the lack of "any strong capabilities in automatic statutory reasoning"); Marcos Pertierra, Erik Hemberg, Una-May O'Reilly & Sarah B. Lawsky, *Toward Formalizing Statute Law as Default Logic Through Automatic Semantic Parsing*, *in* 2D WORKSHOP ON AUTOMATED SEMANTIC ANALYSIS INFO. LEGAL TEXTS (2017). This isn't surprising given the relatively

formalization would have to be tractable by humans. This Article, therefore, focuses on the second question: the best version of the formalization itself.

At the most abstract level, tax law should be formalized in a way that maximizes transparency, government accountability, and accuracy.[7] More specifically, this Article describes a new programming language, Catala, created by a team of computer scientists (Denis Merigoux and Jonathan Protzenko) and lawyers (Liane Huttner and Sarah Lawsky).[8] Catala provides a tractable and functional approach to coding U.S. tax law that offers a more transparent formalization and could potentially hold the government more accountable than the current patchwork of forms, worksheets, and secret programs.[9]

While this Article describes a particular programming language, key characteristics of this particular language could generalize to other programming languages that formalize the law. First, Catala is a domain-specific programming language designed specifically for formalizing tax law.[10] Second, computer code is created in Catala using a well-known approach in the field of computer science (though rarely mentioned in legal literature) called "pair programming," which, in this implementation, takes advantage of the knowledge of both lawyers and computer coders.[11] Third, Catala uses literate programming to create computer code that is, among other things, easier to read and that communicates the decisions behind the coding to the user.[12]

Formalizing tax law in this way won't eliminate its ambiguities. To the contrary, as this Article discusses, formalizing tax law using a domain-specific programming language, pair programming, and literate programming will help highlight ambiguities. Similarly, coding the Code with Cat-

small size of the Internal Revenue Code and tax regulations. This may be the only time that someone has complained the Code and regulations are too small, but consider the amount of text necessary to create, for example, accurate translations from English to French, and compare that with the mere four million words of the Internal Revenue Code and regulations. Moreover, the Code and regulations are highly stylized, and the style is not consistent between the Code and regulations, or even within the Code.

7. *See* Blake L. Currey, *A Shrouded Remedy: Increasing Transparency in the IRS Advance Pricing and Mutual Agreement Program by Releasing Redacted Advance Pricing Agreements and Increasing Administrative Disclosures*, 50 SAN DIEGO L. REV. 1005, 1025 (2013).

8. *See* Denis Merigoux, Nicolas Chataing & Jonathan Protzenko, *Catala: A Programming Language for the Law*, 5:77 ACM ON PROGRAMMING LANGUAGES 1, 6–7 (2021), https://dl.acm.org/doi/pdf/10.1145/3473582 [https://perma.cc/JCR5-ZD37]. The Catala language was designed by Merigoux, Huttner, Lawsky, and Protzenko, and the implementation was led by Merigoux. See *The Catala Language*, CATALA, https://catala-lang.org/en [https://perma.cc/GCY6-M9H2], for more information about Catala.

9. *See* Merigoux, Chataing & Protzenko, *supra* note 8, at 1.

10. Catala is effective for any statute that is drafting using general rules and exceptions. *See infra* Part III.A.

11. *See infra* Section III.B.

12. *See infra* Section III.C.

ala doesn't facilitate predictions of how cases will come out.[13] It isn't "machine learning" or "artificial intelligence." Yet formalizing tax law more completely and transparently will nonetheless have significant benefits.[14] But if the government does not join the private sector in making tax law computationally accessible, codifying the Code may further increase the availability of tax avoidance mechanisms.[15]

This Article first describes the current state of automating the application of the U.S. tax law.[16] It then explains the approach of the Catala programming language to coding tax law and the advantages of that approach for compliance and accountability.[17] The Article concludes by considering the next steps in formalizing tax law using Catala, including potential compliance issues that could arise from the formalization.[18]

II. THE STATE OF PLAY: ALGORITHMS, ABSTRACTIONS, AND ACCOUNTABILITY

Parts of tax law are already computationally accessible. For example, tax forms and worksheets create algorithms that allow taxpayers to comply with portions of the tax law. The vast majority of U.S. taxpayers prepare their returns using computer programs that are created by companies in the private sector, but the programs are based on government-created forms. As this Part shows, however, these formalizations do not track the structure of the underlying law and are not transparent; sometimes, they are even inaccurate.

Many people believe that tax law is already formalized in a way that parallels the underlying law.[19] For example, a recent article states, "TurboTax models real-world rules (i.e., the U.S. Internal Revenue Code), in a way that faithfully represents the logic and meaning of them."[20] This is almost certainly not correct. TurboTax and other similar programs do not represent the tax law in a way that faithfully represents the law's underlying logic.[21] Rather, tax preparation programs are intended to prepare tax forms, and therefore they encode tax forms and worksheets that are created by the government.[22]

---

13. *But see, e.g.*, Benjamin Alarie & Betinna Xue Griffin, *Captive Insurance Appeal in* Reserve Mechanical *Will Likely Fail*, 172 TAX NOTES FED. 1431, 1434 (2021) (using a machine learning approach to predict the outcome of a particular court case).

14. *See infra* Section IV.A.

15. *See infra* Section IV.B.

16. *See infra* Part II.

17. *See infra* Part III.

18. *See infra* Part IV.

19. *See* Joshua P. Davis & Anupama K. Reddy, *AI and Interdependent Pricing: Combination Without Conspiracy?*, 30 J. ANTITRUST, UCL & PRIV. SECTION CAL. LAWS. ASS'N. 1, 6 (2020).

20. *Id.* at 5.

21. Lawsky, *supra* note 3, at 116.

22. Anna Baluch & Eric Rosenberg, *How Does Tax Software Work?*, U.S. NEWS (Feb. 2, 2022), https://www.usnews.com/360-reviews/technology/tax-software/how-does-tax-software-work [https://perma.cc/J43P-M4YQ].

Forms and worksheets are accurate, for the most part, but they are far abstracted from the tax law. They do not track the underlying law, and untangling how they relate to that law can be difficult. Just as revealing formalizations increases transparency and government accountability,[23] the more obscure the formalization, the less the transparency and accountability. Some forms resolve structural ambiguity, hiding the resolution of an ambiguous legal questions within their drafting.[24] Even forms and worksheets that do not resolve ambiguities, however, can obscure the law. Indeed, there is a tension between writing a form that requires the fewest possible steps and is most efficient for a taxpayer to implement, and writing a form that accurately reflects the structure of the language of the law.

Similar issues arise with informal and automated guidance. As Joshua Blank and Leigh Osofsky have described, the U.S. government regularly provides automated tax guidance to taxpayers.[25] For example, the IRS provides the Interactive Tax Assistant, or ITA, an online tool that walks taxpayers through various questions and provides them with tax guidance based on their answers.[26] As the ITA website describes it:

> The Interactive Tax Assistant (ITA) is a tool that provides answers to several tax law questions specific to your individual circumstances. Based on your input, it can determine if you have to file a tax return, your filing status, if you can claim a dependent, if the type of income you have is taxable, if you're eligible to claim a credit, or if you can deduct expenses.[27]

The ITA is not tax preparation software, because it does not allow taxpayers to fill out forms or submit forms to the government.[28] Rather, the ITA answers questions about taxpayers' situations, and then taxpayers use that information to fill out the forms separately. In some sense, the ITA is a formalization of the law. It takes input from the taxpayer, aligns that input with underlying tax rules, and tells the taxpayer how the law applies to them. [29]

Blank and Osofsky show that the ITA is an example of "simplexity"[30]: guidance that "characterize[es] the tax law as clear and not contested, add[s] administrative gloss to the underlying tax law and fail[s] to fully

23. *See generally* ADA LOVELACE INST., AI NOW INST. & OPEN GOV'T P'SHIP, ALGORITHMIC ACCOUNTABILITY FOR THE PUBLIC SECTOR: LEARNING FROM THE FIRST WAVE OF POLICY IMPLEMENTATION 49–50 (2021), https://www.opengovpartnership.org/documents/algorithmic-accountability-public-sector [https://perma.cc/67Z8-B84L] (discussing releasing source code, including for tax law implementation, as a form of transparency).

24. See Lawsky, *supra* note 3, at 135, for an example of a form that resolves ambiguity.

25. Joshua D. Blank & Leigh Osofsky, *Automated Legal Guidance*, 106 CORNELL L. REV. 179, 184 (2020).

26. *Interactive Tax Assistant (ITA)*, INTERNAL REVENUE SERV., https://www.irs.gov/help/ita [https://perma.cc/8XK2-NAF3].

27. *Id.*

28. *See id.*

29. *See id.*

30. Joshua D. Blank & Leigh Osofsky, *Simplexity: Plain Language and the Tax Law*, 66 EMORY L.J. 189, 206–07 (2017).

explain the tax law, such as by omitting exceptions or specific requirements."[31] Blank and Osofsky show that, often, the answers that the ITA provides in automated tax guidance are incorrect or obscure the contested government decisions that led to the answers.[32] As they explain, "[T]he difficult questions underlying such guidance [are] hidden in programming decisions."[33] Therefore, the ITA's simplified formalization of the law not only misleads taxpayers with guidance that is frequently incorrect but also obscures the actual law, thus reducing government accountability.

III. A SOLUTION: THE CATALA PROGRAMMING LANGUAGE

As Part II of this Article explained, the U.S. government has already formalized tax law. But existing formalizations often oversimplify or translate the law incorrectly, fail to track the actual underlying law, and obscure the formalizers' judgments and decisions, which reduces government accountability to taxpayers.[34] Tax law can, however, be formalized and translated into computer code in a way that is tractable, functional, and transparent—or at least more transparent—to taxpayers. This Part introduces the new programming language, Catala, a programming language created specifically for the purpose of encoding tax law, as an example of how this can be done. [35] There are three elements to Catala that make it particularly well-suited to the task of formalizing tax law. [36]

First, Catala is a *domain-specific programming language*: a programming language created specifically for the purpose of coding the Code.[37] Second, law is translated into Catala using *pair programming*: the programming doesn't need to be accomplished by lawyers trained to be computer programmers or programmers in tax law, but rather can be implemented by a team that includes at least one lawyer and at least one programmer.[38] And finally, Catala uses *literate programming*: in addition to having a lawyer-friendly syntax such that the code itself is easily understandable by human readers, the code contains the language of the statute itself and highlights ambiguities both in the law and in the reasoning and decisions of the programming team.[39] This Part illustrates how each element, in turn, increases tractability, accuracy, and accountability when translating tax law into computer code.

---

31. Blank & Osofsky, *supra* note 25, at 207.
32. *See id.* at 208.
33. *Id.* at 185.
34. *See supra* Part II.
35. *See* Merigoux, Chataing & Protzenko, *supra* note 8, at 1.
36. For further discussion, see Liane Huttner & Denis Merigoux, *Traduire la Loi en Code Grâce au Langage de Programmation Catala*, *in* INTELLIGENCE ARTIFICIELLE ET FINANCES PUBLIQUES (Oct. 2020), https://hal.inria.fr/hal-03128248/document [https://perma.cc/3RUZ-2BFR].
37. *See infra* Section III.A.
38. *See id.*
39. *See infra* Section III.C.

A. Domain-Specific Programming Language

A domain-specific programming language is a programming language designed for a particular purpose or application. [40] Catala is a domain-specific programming language for the domain of coding tax law. This is in contrast to a general-purpose programming language, which is of general applicability and is not designed to apply in any particular area.[41] A programming language may be domain-specific in a range of ways. For example, the notations of the language may be designed for a particular application, or the underlying structure of the language may be consistent with the underlying structure of the source material.[42] A domain-specific language may actually be designed to run differently—and more efficiently—than a general-purpose language because its use cases are narrower, and it does not have to account for every possibility.[43]

Catala is domain-specific in a number of ways. First, a program in Catala, unlike most programming languages, mimics the structure of legal thinking by listing the relevant parties, structures, and variables first.[44] Items are named before they are used, which allows humans looking at the code to identify the relevant concepts before applying them.[45]

Second, Catala allows the programmer to define a scope—or range of coverage—for the rule that will be coded.[46] This tracks the Code's structure, in which various rules and definitions apply only to certain portions.[47] Thus, the Code will explicitly state that a rule or definition

40. *See* Federico Tomassetti, *The Complete Guide to (External) Domain Specific Languages*, Strumenta, https://tomassetti.me/domain-specific-languages [https://perma.cc/7JEY-ET4M].

41. A "high-level programming language," or just a "programming language," is how humans usually communicate with computers. *See generally* Noam Nisan & Shimon Schocken, The Elements of Computing Systems: Building a Modern Computer from First Principles ch. 5.1.3 (2005) ("High-level programs manipulate abstract artifacts . . . [that] become series of binary numbers, stored in the computer's data memory."). Java, C++, Python, and BASIC are all programming languages. *See, e.g.*, *Computer Programming Languages*, Computer Science.org (June 23, 2022), https://www.computerscience.org/resources/computer-programming-languages [https://perma.cc/UY6S-MLGZ]. Arguably, it's not possible to translate perfectly between natural languages (or maybe it is! This question is truly outside the scope of this paper), but it is unproblematic, and completely commonplace, to write identical instructions in different programming languages. The instructions from the high-level programming language are translated into machine language, which the computer then executes. This translation is called "compiling" the high-level language. *See id.*

42. *See* Marjan Mernik, Jan Heering & Anthony M. Sloane, *When and How to Develop Domain-Specific Languages*, 37 ACM Computing Surys. 316, 317–323 (2005).

43. *Id.* at 316. See Shrutarshi Basu, Nate Foster & James Grimmelmann, *Property Conveyances as a Programming Language*, *in* Onward! 2019: ACM SIGPLAN Int'l Symp. on New Ideas, New Paradigms, & Reflections on Programming & Software 128 (2019), for an example of another law-related domain-specific programming language.

44. *See* Huttner & Merigoux, *supra* note 36, at 7–8.

45. *See id.*

46. *See id.* For further discussion of the importance of scope within the Internal Revenue Code, see Sarah B. Lawsky, *Formalizing the Code*, 70 Tax L. Rev. 377, 380–81 (2017).

47. *See* Sarah B. Lawsky, *A Logic for Statutes*, 21 Fla. Tax Rev. 60, 74–76 (2017) (discussing the importance of the divisions of the Code).

applies, for example, only for purposes of "this subchapter,"[48] a particular named portion of the Code,[49] "this paragraph,"[50] and so forth. Catala can define terms and delimit rules accordingly.

Most critically, Catala is structured using *default logic*, a nonstandard logic that represents the underlying structure of the U.S. tax code more accurately than does standard logic. [51] Much of the U.S. tax code (and other tax statutes as well, including French tax law) is structured as general rules followed by exceptions.[52] This is apparent in § 121.[53] The general rule in § 61(a) is that all income, including gain from a sale, is included in gross income.[54] Section 121(a) provides an exception to § 61(a): it says that income from the sale of a principal residence is not included in gross income. [55] Then, § 121(b)(1) provides an exception (or limitation) to § 121(a): only gain up to $250,000 may be excluded.[56] Section 121(b)(2) provides an exception to § 121(b)(1): gain up to $500,000 may be excluded if the taxpayer is married filing jointly.[57] And so forth.

Because the statute is structured as general rules followed by exceptions, it is easiest to represent the statute using a logic that itself is structured as a "logic of exceptions": default logic. The basic idea of default logic is that rules are prioritized, and rules of higher priority override rules of lower priority.[58] As a result, one could reach a conclusion in default logic that is entirely logically supported and then, provided with more information, reject that conclusion.[59] This is in contrast to standard logic, where information can only be added and cannot be overridden once proven in the system.[60] Standard logic is thus "monotonic": the set

48. *E.g.*, I.R.C. § 708(a) ("For purposes of this subchapter, an existing partnership shall be considered as continuing if it is not terminated.").

49. *E.g.*, *id.* § 6513(a) ("For purposes of section 6511 any return filed before the last day prescribed for the filing thereof shall be considered as filed on such last day.").

50. *E.g.*, *id.* § 544(a)(2) ("For purposes of this paragraph, the family of an individual includes only his brothers and sisters . . . , spouse, ancestors, and lineal descendants.").

51. See Lawsky, *supra* note 47, at 77–80 for a more extensive discussion of default logic in general and this claim in particular. For a technical explanation of how Catala implements default logic, see Denis Merigoux, Raphaël Monat & Jonathan Protzenko, *A Modern Compiler for the French Tax Code*, *in* 30TH ACM SIGPLAN INT'L CONF. ON COMPILER CONSTR. 71–82 (2021), https://hal.inria.fr/hal-03002266v3 [https://perma.cc/9Y88-UE8N], and Merigoux, Chataing & Protzenko, *supra* note 8, at 3. The approach for implementing default logic in these articles builds upon that of Gerhard Brewka and Thomas Eiter. *See* Gerhard Brewka & Thomas Eiter, *Prioritizing Default Logic*, *in* INTELLECTICS & COMPUTATIONAL LOGIC 27 (2000).

52. The general rule-followed-by-exception approach is not limited to the U.S. tax code. This approach can be seen on inspection of various statutes, and it is explicitly recommended in both the House and Senate Legislative Drafting Manuals. *See* OFF. OF LEGIS. COUNS., H.R., HOUSE LEGISLATIVE COUNSEL'S MANUAL ON DRAFTING STYLE 23 (1995); OFF. OF LEGIS. COUNS., S., LEGISLATIVE DRAFTING MANUAL 9 (1997). For further discussion, see Lawsky, *supra* note 47, at 72–77.

53. *See* I.R.C. § 121.

54. *See id.* § 61(a).

55. *See id.* § 121(a).

56. *See id.* § 121(b)(1).

57. *See id.* § 121(b)(2).

58. *See* Lawsky, *supra* note 47, at 73–77.

59. *See id.* at 64.

60. *See id.*

of what's proven can only grow.[61] Default logic is "nonmonotonic": sometimes, what is proven with one set of information can be retracted given more information.[62]

This is, of course, how people reason in everyday life—by accepting and acting on conclusions but being open to rejecting those conclusions if they learn more information. The canonical example is that of Tweety, the bird. If someone tells you that Tweety is a bird and asks whether Tweety can fly (and demands that you answer yes or no), you will answer yes. But if the person adds that Tweety is a penguin, you will revise your answer. Default logic is thus sometimes called "the logic of everyday reasoning."[63]

Deciding which logic to use is a pragmatic choice. Default logic and standard logic can be used to represent the exact same law, both equally and accurately. Consider, for example, the following set of rules. Section 61(a)(11) provides the general rule that income from the discharge of indebtedness is included in gross income.[64] Section 108(a)(1)(B), however, provides an exception to that rule: income from the discharge of indebtedness is not included if the discharge occurs when the taxpayer is insolvent.[65] Section 108(a)(3) is an exception to the exception, or a limitation on the exception: the insolvency exception applies only to the extent that the taxpayer is insolvent (a taxpayer is insolvent to the extent that the taxpayer's liabilities exceed her assets immediately before the debt is discharged).[66]

Each of these rules can be formalized individually:

$\sigma_{3:}$ $\$X$ Income from Discharge of Indebtedness $\rightarrow$ Include $\$X$[67]

$\sigma_{2:}$ ($\$X$ Income from Discharge of Indebtedness *and* Insolvent) $\rightarrow$ Include $\$X$[68]

$\sigma_{1:}$ ($\$X$ Income from Discharge of Indebtedness *and* Insolvent) $\rightarrow$ Include ($\$X$ - Lesser($\$X$, (Liabilities - Assets)))[69]

This formalization is just one of many ways to represent these rules. The three rules track the relevant sections of the Code. Default logic now allows these rules to be ranked. Following the rule familiar to every lawyer that the specific controls over the general, $\sigma_{1,}$ the most specific rule, controls over $\sigma_{2,}$ which in turn dominates $\sigma_{3.}$

The same set of rules could be written in a single statement in standard logic (again, this is only one possible representation):

---

61. *See id.*
62. *See id.* It is useful to conceive of the law as nonmonotonic logic and defeasible reasoning; Thorne McCarty's work is foundational to this insight. *See, e.g.*, L. Thorne McCarty, *Some Arguments About Legal Arguments*, 6 INT'L CONF. ON A.I. & L. 215, 216 (1997).
63. *See* Lawsky, *supra* note 47, at 64.
64. I.R.C. § 61(a)(11).
65. *See id.* § 108(a)(1)(B).
66. *See id.* § 108(a)(3).
67. *Id.* § 61(a)(11).
68. *Id.* § 108(a)(1)(B).
69. *See id.* § 108(a)(3).

($*X* Income from Discharge of Indebtedness) → Include ($*X* - Lesser($*X*, Maximum(0, (Liabilities - Assets))))[70]

In terms of outputs given inputs, the default logic and standard logic representations are the same. But the representation in default logic has several advantages. First, the standard logic translation, in this case, is incomplete and therefore wrong. As drafted, it captures only one of many exceptions outlined in § 108.[71] Discharge of indebtedness also can be excluded if the discharge occurs in a Title 11 case,[72] if the discharge is "qualified farm indebtedness,"[73] if the discharge occurs within a certain date range and is related to a principal residence,[74] and so forth. To write the standard logic statement correctly, one would have to include *each* exception to the general rule that cancellation of indebtedness is income in § 108.[75] The standard formalization as drafted above is, therefore, incorrect. If a taxpayer is not insolvent, then the standard formalization says that the income must be included. Recall that standard logic is monotonic: once something is added to the knowledge base, later information cannot retract that knowledge. To formalize in standard logic, therefore, one must know all the relevant exceptions.

The default logic formalization, in contrast, is incomplete but not wrong. Other exceptions that override the initial rule could be added. No disclaimer would be necessary for the default logic formalization because the idea that future exceptions could provide different outcomes is built into the idea of default logic.

Second, the standard logic representation hides the law even as it implements the law. The relationship between standard logic representation and the law itself is entirely unclear from the face of the representation. The easiest way to satisfy oneself that the standard logic representation is accurate is not to inspect the statute but to verify the outputs given inputs. For example, consider three possible situations that exhaust all relevant possibilities: a taxpayer is not insolvent; a taxpayer is insolvent and the debt relief is less than the extent of their insolvency; or a taxpayer is insolvent and the debt relief is greater than the extent of their insolvency.

Recall that a taxpayer is insolvent if their liabilities exceed their assets. Accordingly, if a taxpayer is not insolvent, Liabilities - Assets is less than zero because the taxpayer's assets exceed their liabilities. The lesser of the amount discharged ($*X*) and zero is zero. The taxpayer includes $*X* - 0 = $*X*.

If the taxpayer is insolvent, Liabilities - Assets is greater than zero, given the definition of insolvency. If the debt relief is less than the extent

70. *See id.* §§ 61(a)(11), 108(1)(B), 108(a)(3).
71. *See id.* § 108(a)(1).
72. *Id.* § 108(a)(1)(A).
73. *Id.* § 108(a)(1)(C).
74. *Id.* § 108(a)(1)(E).
75. *Cf.* Ronald M. Dworkin, *The Model of Rules*, 35 U. CHI. L. REV. 14, 25 (1967) (stating that rules are "all-or-nothing" and that an "accurate" statement of a rule takes all exceptions in account and "legal consequences . . . follow automatically").

of insolvency, the lesser of $\$X$ and Liabilities - Assets is $\$X$. The taxpayer therefore includes $\$X$ - $\$X = 0$ in their gross income; that is, that taxpayer excludes all of the debt relief. If the debt relief is greater than the extent of the taxpayer's insolvency, Liabilities – Assets is less than $\$X$, and the taxpayer reduces the amount included in gross income by Liabilities - Assets; that is, the taxpayer excludes debt relief only to the extent of the excess of liabilities over assets.

The standard logic account thus hides core ideas of the statute: that there is a general rule that all income is included, including discharge of indebtedness income; that insolvency affects that general rule; and that there is a limitation on the extent to which insolvency changes the general rule.[76] Put another way, the actual statute conveys *ideas* separate from mere *outputs*. The statute identifies a baseline rule that income should generally be included. The statute identifies a characteristic of interest: insolvency. The statute conveys that insolvency is of interest, though, not as an on/off switch but as related to the relationship between the taxpayer's assets and liabilities. A statute (even a tax statute!) does more than just determine outputs given inputs.[77]

Third, and relatedly, translating the statute into default logic is easier than translating the statute into standard logic.[78] Because the underlying logic tracks the statute, the default logic translation can simply follow the statute itself.

Finally, if the statute changes, it will be easier to change the default logic formalization. Removing an exception in standard logic requires untangling a statement in which every word depends on every other word. Because default logic, by contrast, tracks the statute so closely and because default logic allows for exceptions that override more general rules, formalizing in default logic allows separate sections and subsections of the code to remain separate. Drafting is accordingly more modular and allows coders to swap new law in and old law out more easily.

The elements that make Catala domain-specific, including the preliminary definition of terms and concepts, the explicit statement of the scope of application, and the underlying structure of the language, increase the ease of coding and make the coding of the Code more accountable. As the next two Sections show, both who drafts the computer code and the code's contents also increase tractability and accountability.

B. Pair Programming

Turning legal code into computer code by hand requires expertise in both law and coding. One approach is to train individuals in both fields.[79] Law schools could include classes on coding, or there could be training

76. *See* I.R.C. §§ 61(a), 121(a), (b)(1)–(2).

77. Literate programming, another aspect of Catala, also helps the computer code convey meaning beyond outputs. *See infra* Section III.C.

78. *See* Lawsky, *supra* note 47, at 78.

79. *See id.* at 10.

classes for coders on tax law. The concern, of course, is that it is easy to train someone to be a poor coder, but becoming a skilled coder takes a significant amount of time and experience. Similarly, it is challenging to read tax statutes, let alone understand how the complex language is meant to interact with the rest of the statute. And, even if one had sufficient expertise in both tax law and coding, the task of formalizing statutes is sufficiently subtle that it should not be left to a single person.

Pair programming is the better approach to coding the Code, though not previously discussed in the U.S. legal literature.[80] Pair programming generally refers to two programmers working together side by side at the same computer to write computer code.[81] While pair programming isn't necessarily right for every project, evidence shows that pair programming can lead to fewer bugs and better designs.[82] In the usual pair programming, programmers have the same or similar skill sets and use the process to bounce ideas off each other, notice details they may have otherwise missed working alone, and share expertise.[83]

Pair programming in the context of coding the Code does not involve two coders with similar skills but rather a coder and a lawyer, people with radically different areas of expertise.[84] Pair programming in this manner (with a lawyer and a coder) captures many of the approach's usual advantages and allows the law to be encoded without the need for one person to be an expert in both tax law and coding. For pair programming to be most effective, the lawyer should know a little bit about coding (so they can read and evaluate the code the programmer is writing), and the programmer should know a little bit about tax law. A lawyer who took a coding class or two and a programmer who has done some reading about tax law are a good pair for these purposes.

To demonstrate how pair programming can work, this Section describes the actual experience and process of two individuals, a lawyer (Sarah Lawsky) and a coder (Jonathan Protzenko), pair programming § 61(a)(11) and § 108(a)(1)(B).

The lawyer provided a written description of the code section in ad-

80. Pair programming has been discussed in only one law review article as of this writing (although only in passing). *See* Shaanan Cohney & David A. Hoffman, *Transactional Scripts in Contract Stacks*, 105 MINN. L. REV. 319, 329 n.57 (2020).

81. *See generally* Stuart Wray, *How Pair Programming Really Works*, 27 IEEE SOFTWARE 50, 50 (2010) (providing an overview of the pair programming method).

82. *See, e.g.*, Andrew Begel & Nachiappan Nagappan, *Pair Programming: What's In It for Me?* 2 ACM-IEEE INT'L SYMP. ON EMPIRICAL SOFTWARE ENG'G & MEASUREMENT 120 (Jan./Feb. 2008); Laurie Williams, Robert R. Kessler, Ward Cunningham & Ron Jeffries, *Strengthening the Case for Pair Programming*, IEEE SOFTWARE 19, 19 (July/Aug. 2000).

83. Wray, *supra* note 81, at 51–55.

84. THE SERVICE INNOVATION LAB, N.Z. DEP'T OF INTERNAL AFFS., BETTER RULES FOR GOVERNMENT DISCOVERY REPORT 3–5 (2018), https://www.digital.govt.nz/dmsdocument/95-better-rules-for-government-discovery-report/html [https://perma.cc/X8YV-NXXX] (describing advantages of coding using an interdisciplinary team that includes both subject area experts and coders).

vance, including the formalization provided above.[85] At the meeting, the lawyer and coder looked at the language of the code sections and talked it through. The coder asked some questions, and the lawyer explained the section. The lawyer provided an example with numbers so the coder could understand the section better. This took about ten minutes total because § 61(a)(11) and the insolvency exception in § 108 are fairly straightforward;[86] for more complex sections, this initial process alone can take well over an hour.

The coder began by setting some terms upfront. He did not initially understand that the taxpayer was the one whose debt was forgiven rather than the one who repaid the debt, so he initially called the relevant variable "DEBT_REPAYMENT." The lawyer explained the statute again, and the coder changed the variable to "DISCHARGE_OF_INDEBTEDNESS_INCOME." The coder also initially wrote that DISCHARGE_OF_INDEBTEDNESS_INCOME increased the variable "INCOME," but the lawyer pointed out that it should increase the variable "GROSS_INCOME."

The coder knew there would be exceptions. So, consistent with the initial formalization provided by the lawyer, he labeled the first rule, based on § 61(a)(11), "delta_3," so that he could later code "delta_2" as an exception to delta_3, and thus override delta_3.

The coder and lawyer then turned to delta_2, which was intended to capture § 108(a)(1)(B). The coder initially defined the scope as 108a1b (lowercase "b"); the lawyer noted that this should be 108a1B (uppercase "B"), to track the statute. This provision, as described above, allows the taxpayer to exclude discharge-of-indebtedness income to the extent the taxpayer is insolvent.[87]

Both the coder and lawyer paused to reflect on a problem with the statutory language. Definitions in the Code are supposed to be extensional such that the definition of a term can be substituted anywhere the term appears: "when Congress inserts a definitional section, courts resort . . . to the statutory definition alone. Congress in effect replaces a complicated and fuzzy algorithm with a simple cut-and-paste function: Where one sees *X*, one shall read *Y*."[88]

While the approach of substituting a definition for a term *salva veritate* is an accurate description of how definitions should work, drafting does not always follow this definition. Section 108(d)(3) defines "insolvent" as follows (the exact language is important here): "For purposes of this section, the term 'insolvent' means the excess of liabilities over the fair market value of assets."[89] But § 108(a)(1)(B) says that gross income does not include any amount that would be includible by reason of discharge of

85. *See* text accompanying *supra* notes 67–69.
86. *See* I.R.C. §§ 61(a)(11), 108.
87. *See id.* § 108(a)(1)(B).
88. Nicholas Quinn Rosenkranz, *Federal Rules of Statutory Interpretation*, 115 HARV. L. REV. 2085, 2104 (2002) (internal quotations omitted).
89. I.R.C. § 108(d)(3).

debt if "the discharge occurs when the taxpayer is insolvent." [90] Congress clearly did not intend for income to be excluded from gross income only when "the taxpayer is the excess of liabilities over the fair market value of assets," and no one would ever interpret the statute that way. While the coding of the statute generally attempts to follow the language of the statute as closely as possible, it would be incorrect to define insolvent as the Code defines the term and then write computer code that excludes income when the statement "taxpayer = insolvent" is true. Here, therefore, the coder and lawyer chose to depart from the language of the statute. While this departure was necessary and reflects how this section is always implemented, it is indeed a departure. The discussion of literate programming below returns to this departure.[91]

The coder wrote delta_2 to say that the discharge-of-indebtedness income should not be included in gross income under the condition of insolvency. As stated, the computer code does not determine whether the statement "taxpayer = insolvent" is true. That is what the statutory language seems to suggest, but that would be incoherent. Instead, the computer code checks to see whether the statement "insolvency > 0" is true. That is, the computer code determines whether the taxpayer has an excess of liabilities over assets.

The coder and the lawyer focused on coding the definition of insolvency.[92] The program could ask the taxpayer to enter the extent to which he is insolvent, or it could ask the taxpayer to enter the fair market value of his assets and liabilities and calculate insolvency for him. The definition of insolvency is not intuitive, clear, or easy to code.[93] Therefore, the coder and lawyer decided to have the taxpayer enter the fair market value of their assets and the amount of their liabilities.

In tax law, the excess of *A* over *B* can only ever be a positive number or zero. The excess of $100 over $80 is $20, because $100 - $80 = $20. The excess of $80 over $100 is zero because $80 is smaller than $100.[94] The lawyer pointed out that, while this could be coded by excluding the lesser of Liabilities - Assets and zero, the "excess of" language occurred throughout the Code and would arise many other times. She suggested that the coder create an "excess_of" operator within the language, which he did. The extent of insolvency was then defined as "excess_of(assets, liabilities)," where the taxpayer is asked to enter amounts for assets and liabilities.

---

90. *Id.* § 108(a)(1)(B).
91. *See infra* Section III.C, for discussion.
92. *See* I.R.C. § 108(d)(3).
93. *See id.*
94. One can see this in, for example, INTERNAL REVENUE SERV., DEP'T OF THE TREASURY PUBL'N 4681 CANCELED DEBTS, FORECLOSURES, REPOSSESSIONS, AND ABANDONMENTS (FOR INDIVIDUALS) 7 (2022), in the worksheet for determining extent of insolvency. Line 38 directs the taxpayer to subtract the fair market value of total assets from total liabilities: "If zero or less, you aren't insolvent." *Id.* For further discussion of "excess of" in the Internal Revenue Code and regulations, see Sarah B. Lawsky, *Teaching Algorithms and Algorithms for Teaching*, 24 FLA. TAX REV. 1, 20–21 (2021).

The coder then coded "delta_1" as an exception to delta_2, indicating that delta_1 would override delta_2. This provision tracks § 108(a)(3), which limits the insolvency exclusion to the amount of insolvency.[95] This all took roughly an hour and resulted in a preliminary version that still needed the syntax fixed to match the requirements of Catala. (For example, the coder still had to actually define the "excess_of" operator within Catala.) The pair programming session ended, and the coder took the preliminary version to finalize on his own. Once the finalized code was ready, it was checked by running various inputs for assets, liabilities, and gross income to see whether it accurately captured the statutory rules.

C. Literate Programming

Catala code includes extensive comments aimed at the individual reader along with the language the computer interprets to determine what to do.[96] This approach follows the theory of literate programming.[97] Literate programming is the idea (first put forward by Donald Knuth) that a program should be thought of not only as communicating to a computer what to do but also as communicating to those reading the program what the program wants the computer to do.[98] Literate programming has become widely popular within math and science programming, for example, and permits computational analysis in those areas to be more easily explained, shared, and replicated.[99]

Finalized Catala code thus includes not only the functional code that the computer uses to execute the program but also the actual relevant statutory language.[100] The statutory language is included as a "comment," which means the computer ignores that portion of the program. The statutory language is there entirely for the individual reviewing the code.[101] Including the relevant statutory language is straightforward because Catala is built on default logic, which allows the program to track the structure of the statute.[102] Thus, the programmer may easily include with each snippet of code the specific subsection, paragraph, or subparagraph (for example) that the snippet encodes.

---

95. *See* I.R.C. § 108(a)(3) (2021).
96. *See generally* Merigoux, Chataing & Protzenko, *supra* note 8, at 3.
97. Literate programming has been discussed in two law review articles. Paul Ohm claims to enact literate programming in both a law review article and a computer program, but the article does not discuss the theory or reasoning behind literate programming. Paul Ohm, *Computer Programming and the Law: A New Research Agenda*, 54 Vill. L. Rev. 117, 117 (2009). Scott T. Luan discusses literate programming in the context of discussing what should be patentable. Scott T. Luan, *All That Is Solid Melts into Air: The Subject Matter Eligibility Inquiry in the Age of Cloud Computing*, 31 Santa Clara High Tech. L.J. 313, 350–51 (2015).
98. Donald E. Knuth, *Literate Programming*, 27 Computer J. 98, 99 (1984).
99. Mary Beth Kery, Marissa Radensky, Mahima Arya, Bonne E. John & Brad A. Myers, *The Story in the Notebook: Exploratory Data Science Using a Literate Programming Tool*, *in* CHI Conf. on Hum. Factors Comput. Sys. 1 (2018).
100. *See* Merigoux, Chataing & Protzenko *supra* note 8, at 1, 3.
101. *See id.* at 1.
102. *See id.* at 3.

Including the statutory language helps the reader of the computer code track exactly what the computer code is meant to do. But the comments can—and should—include at least two additional kinds of information related to the sources of law and to ambiguities. First, comments can include an explanation of the coding team's decisions, including extra-statutory legal support for those decisions. Second, comments can make explicit the ambiguities the coding team perceived and how and why the team chose to resolve the ambiguities as they did.

The Code is not, of course, a complete statement of U.S. tax law. Regulations provide an enormous amount of law, and indeed "coding the Code," as explained above, is better understood as "coding the Code and regulations." Sometimes regulations will be separately coded, but the regulations often provide details about how the Code is to be interpreted, which is necessary to even write code for the Code in the first place. For example, § 121 sometimes requires the taxpayer to prorate an exclusion based on how long the taxpayer used the house for a particular purpose.[103] The statute says that the proportion is the ratio of the relevant time period to two years.[104] How closely must one calculate the relevant time period? Should one round to the nearest minute? Hour? Six months? Year? What about leap days? The regulations make this statute administrable by saying that the calculation can be done using either months or days; if months are used, the denominator should be 24, and if days are used, the denominator should be 730.[105] When this section is coded using Catala, the computer code can use either days or months, and the comments can state that the regulation supports this approach.

Literate programming also allows the coder to note when they have departed from the language of the statute. Recall that the statutory definition of insolvent cannot possibly be the correct definition of insolvent,[106] as it conflates the amount of insolvency with whether a taxpayer is in the state of having more liabilities than assets. The coder can and should choose to write code reflecting the actual meaning of insolvency. The coder should also, however, explain the problem in the comments and explain why they chose to resolve the issue as they did, thus increasing transparency.

Literate programming can also highlight ambiguities and increase transparency around when and how these ambiguities are resolved. Some ambiguities can be maintained when coding. For example, "income" is an ambiguous term not directly defined in the Code.[107] Gross income includes all income, and income is, essentially, anything that makes the tax-

103. *See* I.R.C. § 121(c)(1).
104. *See id.*
105. *See* Treas. Reg. § 1.121-3(g)(1).
106. *See* discussion *supra* notes 89–90 and accompanying text.
107. For a vigorous discussion over how broadly the government has or should roam in its definition of "income," compare Alice G. Abreu & Richard K. Greenstein, *Defining Income*, 11 FLA. TAX REV. 295 (2011), with Lawrence Zelenak, *Custom and the Rule of Law in the Administration of the Income Tax*, 62 DUKE L.J. 829 (2012).

payer better off.[108] The computer code does not need to define income anywhere; it can ask for taxpayers to input amounts that are income and then run calculations on those amounts. It can essentially allow the ambiguity of the term to remain.

Other ambiguities, however, must be resolved in order for the tax law to be coded. A statute can leave the order in which calculations are to be run ambiguous, for example, with different ordering leading to different results.[109] The IRS has at times resolved these ambiguities within the forms themselves, consequently withholding from the taxpayer that any such ambiguity existed in the first place. Without resolving the ambiguity, the form could not exist, and neither could code that formalizes the statute.

I've suggested elsewhere that when a form hard-codes an ambiguity, the IRS could be explicit about the decision that the form has made regarding the ambiguity and allow the taxpayer to choose between different interpretive approaches—essentially, providing the taxpayer with two possible forms.[110] Code that uses the literate programming approach could explain the ambiguity in the comments, providing alternate code that could be used if the taxpayer wished to take a different approach than the lawyer and coder's, or even leave the choice of which code to use up to the user.

IV. THE FUTURE OF CODING THE CODE

Catala makes the task of translating tax law into computer code tractable. While translating the entire Code and regulations by hand is unlikely, at least in the near future, the task of translating portions of these materials could be manageable. Depending on the purpose of the translation, even translating a small subset of the Code could be highly useful. The partnership tax code and regulations, for example, are an almost entirely self-contained subset.[111] Significant and relevant portions of the Code may soon be encoded using pair programming. Catala does not create any of these possibilities, but having Catala, which is designed to code the tax law transparently and efficiently, makes computational access to the law more possible, along with all that would flow from that access. This Part looks at the potential advantages and risks of coding the Code, focusing on the potential distributive effects of formalization.

---

108. Comm'r v. Glenshaw Glass Co., 348 U.S. 426, 429–431 (1955) (income includes "accessions to wealth, clearly realized, and over which the taxpayers have complete dominion").

109. For an example of such ambiguity, see Lawsky, *supra* note 3, at 126–31 (describing an ambiguity in § 165(h) in the order in which certain calculations are to be performed).

110. *See id.* at 144–47 (describing a range of options, including, but not limited to, locked-in forms (the current approach); providing multiple forms; or providing no form at all).

111. The entirety of Subchapter K is devoted to tax partnerships. I.R.C. subt. A, ch. 1, subch. K.

A. Increasing Access and Efficiency

Formalizing the tax law has the potential to make tax law more easily administrable in ways that will redound to the benefit of middle- and low-income taxpayers. First, formalizing the law using Catala in a way that is faithful to the underlying structure of the law and open about the ambiguities that remain will obviate the need for other formalizations, including forms and "simplexified" automated guidance. [112] Instead of creating separate formalizations for forms (the ITA and so forth), the IRS could draw on the underlying formalization of the law itself. This would allow the IRS to provide more accurate guidance to taxpayers who cannot afford to hire a professional to assist them with tax returns. The IRS would also be largely spared from having to create forms—a time-consuming process that, for some forms, is an annual task.[113]

Second, the formalization will make it easier for the government to provide online tax preparation services, as opposed to outsourcing those services to for-profit entities. Although the vast majority of individual taxpayers file their taxes electronically,[114] the IRS does not currently provide an online interface for preparing tax returns. Rather, the IRS relies on for-profit companies (such as Intuit, which makes TurboTax, and H&R Block) to create online filing options, including free filing options for low-income taxpayers. [115] Indeed, until very recently, the IRS's contract with these companies barred the IRS from creating its own online filing option.[116] The online filing option was recently sent into disarray as reporters uncovered unethical behavior by for-profit companies involved in free filing, which caused the two largest companies involved in free filing to withdraw from the program.[117] The Taxpayer Advocate Service identified the state of the free filing program as one of the most serious problems facing the IRS.[118] Computationally accessible tax law will make the free filing program and other online filing options easier to improve. The government could provide online tax preparation options itself, and if the formalization of the tax law is public, it would be fairly straightforward for other not-for-profit entities to take advantage of the computational accessibility of the tax law to create or improve programs to allow online filing.

---

112. *See* Blank & Osofsky, *supra* note 25, at 179–80.

113. *See generally Prior Year Forms and Instructions*, Internal Revenue Serv., https://www.irs.gov/forms-pubs/prior-year [https://perma.cc/ANY5-EST7] (providing access to the 2020 and 2021 versions of various tax forms).

114. In fiscal year 2020, over 94% of individual tax returns were filed electronically. *See* Internal Revenue Serv., Dep't of the Treasury, 2020 Data Book 2 (2021).

115. Laurel Wamsley, *H&R Block, TurboTax Accused of Obstructing Access to Free Tax Filing*, NPR (May 7, 2019), https://www.npr.org/2019/05/07/720941665/la-city-attorney-sues-intuit-and-h-r-block-alleging-they-undermine-free-tax-fili [https://perma.cc/C74G-X5J2].

116. *See id.*; Dylan Matthews, *The IRS Has a Big Opportunity to Fix the Way Americans File Taxes*, Vox (Aug. 13, 2021), https://www.vox.com/policy-and-politics/22596072/irs-turbotax-hr-block-free-file-tax-return [https://perma.cc/3JRE-9VJS].

117. *See* Matthews, *supra* note 116.

118. *See* Nat'l Taxpayer Advoc., Annual Report to Congress 45 (2019).

B. Increasing Access to Inefficiency

Coding tax law could also provide a path to tax avoidance, however, making it especially critical that the government, as well as private taxpayers, have access to computationally tractable law. In particular, coding tax law could make it much easier to create certain types of tax shelters.[119]

Some tax shelters achieve results by giving legislatively unintended meanings to words.[120] Some tax shelters simply employ falsehoods—inflating the amount of a charitable contribution, for example.[121] But others work by chaining together disparate rules to create a tax avoidance situation.[122] While individuals may be adept at expanding the definition of terms or lying about a valuation, computers are far better than humans at looking at large numbers of permutations of increasingly complex rules and fact patterns and finding combinations of rules, structures, and facts that minimize the amount of tax owed.[123]

The transaction in *Summa Holdings, Inc. v. Commissioner* is an example of such a chaining tax shelter. [124] In that case, taxpayers used "domestic international sales corporations," or DISCs, to transfer money into their sons' Individual Retirement Accounts (IRAs), which subsequently accrued tax-free gains.[125] DISCs were designed by Congress to incentivize exporting goods.[126] A portion of a DISC's profits is deemed distributed each year. [127] An IRA may own an interest in a DISC but must pay an unrelated business tax on dividends from the DISC at the corporate tax rate.[128]

---

119. Michael J. Graetz, *100 Million Unnecessary Returns: A Fresh Start for the U.S. Tax System*, 112 Yale L.J. 261, 278 (2002).

120. For example, Notice 2015-73 describes basket option contracts, in which the taxpayer attempts to expand the concepts of "options" and "own" beyond a meaning acceptable to the IRS, even while working within the text of the statute. I.R.S. Notice 2015-73, 2015-46 I.R.B. 660.

121. For example, in syndicated conservation easement transactions, as described in Notice 2017-10, taxpayers claim a deduction based on "overvaluation of the [donated] conservation easement"—that is, they get an inflated charitable deduction. I.R.S. Notice 2017-10, 2017-4 I.R.B. 544.

122. *See* Graetz, *supra* note 119, at 277–78.

123. *See* Erik Hemberg, Jacob Rosen, Geoff Warner, Sanith Wijesinghe & Una-May O'Reilly, *Tax Non-Compliance Detection Using Co-Evolution of Tax Evasion Risk and Audit Likelihood*, *in* 15th Int'l Conf. A.I. & L. 79, 79–80 (2015) (using a computer program to simulate the co-evolution of tax evasion and auditing to demonstrate how tax shelters could be developed).

124. *See* Summa Holdings, Inc. v. Comm'r, 848 F.3d 779, 783–84 (6th Cir. 2017).

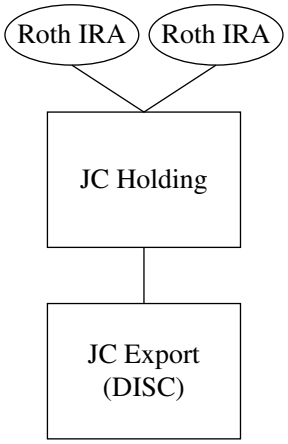
125. *See id.*

126. *See id.* at 782. The original version of the DISC was subject to an adverse ruling by the GATT and WTO on the grounds that the DISC was an export subsidy—the DISC described in Summa is a modified version of a DISC, what's known as an IC DISC. *See* United States Tax Legislation (DISC), L/4422–23S/98, Report of the Panel Presented to the Council of Representatives ¶¶ 3–5, 17 (Nov. 12, 1976). The profits of a DISC are not subject to corporate taxation, but rather are taxed to the DISC shareholders when the profits are distributed or are deemed distributed. *See Summa*, 848 F.3d at 783.

127. *See Summa*, 848 F.3d at 782.

128. *See id.*

The tax shelter worked as follows.[129]



The Benenson family owned Summa Holdings, the parent corporation of a manufacturing group of corporations.[130] The two children of the Benenson family each established a Roth IRA and put in $3,500.[131] The adults in the Benenson family could not have established a Roth IRA, as they earned far in excess of the amount above which one cannot contribute to a Roth IRA.[132] Each Roth IRA then paid $1,500 for shares of a new DISC, JC Export, before selling their shares of JC Export to a corporation, JC Holding.[133] After the sale, each Roth IRA owned 50% of JC Holding, and JC Holding owned 100% of JC Export.[134]

Due to a series of agreements between Summa Holdings and JC Export, Summa paid JC Export over $5 million over the course of six years.[135] JC Export distributed the money to JC Holding, which in turn paid the required 33% income tax on the dividends and distributed the balance to the Roth IRAs.[136] The Tax Court recharacterized these payments as dividends from Summa to its shareholders, followed by (excessive) contributions to the Roth IRAs.[137] The Sixth Circuit overturned the Tax Court and upheld the original characterization of the transactions.[138] The Benenson family used this structure to bypass the contribution limits on the Roth IRAs. [139]

This successful tax shelter was created not by exploiting ambiguity in a term or lying about valuation or some other fact but rather by "chaining"

129. *See id.* at 783–84.
130. *See id.* at 783.
131. *See id.*
132. *See id.* at 783–84.
133. *See id.* at 783.
134. *See id.*
135. *See id.* at 784.
136. *See id.*
137. *See id.*
138. *See id.* at 780–81.
139. *See id.*

apparently unrelated provisions of the Code to create an unintended tax savings opportunity—one that followed the letter of the law closely enough that a federal circuit court upheld the structure.[140] As the court in *Summa* described the shelter, this gap in taxation was created by the complexity of the Code itself.[141]

This returns to the epigram for this Article: Judge Sutton's statement in the *Summa Holdings* case, in which he says "we can all agree" that citizens should be able to "see" the tax law.[142] He continues, "The Internal Revenue Code improves matters in one sense, as it is accessible to everyone with the time and patience to pore over its provisions." [143] He then upholds the structure just described in *Summa* because the taxpayers "complied in full with the printed and accessible words of the tax laws. The Benenson family, to its good fortune, had the time and patience (and money) to understand how a complex set of tax provisions could lower its taxes."[144]

If the corpus of tax law is made available to computation, however, neither time, patience, nor money may be necessary to determine how a complex set of tax provisions can lower a taxpayer's taxes. A computer might be able to sort through various permutations of law, regulations, and fact patterns to find similar opportunities for chaining. A computer could check for opportunities for regulatory arbitrage—situations in which "[t]wo transactions with identical cash flows receive different . . . treatment" under the tax law—or it might be able to find situations in which "[t]he same transaction receives different . . . treatment [under the tax law] in the future than it does today."[145] If such a program is developed, the marginal cost of developing a tax shelter will drop to zero. As the IRS shuts down one shelter, another shelter will be marketed to take its place. And even if, in theory, anyone could generate a tax shelter, tax shelters benefit only the wealthy.[146]

Generating tax shelters from computationally accessible tax law has been proven possible.[147] Catala itself does not create that possibility, but it does make it tax law more computationally accessible, because it tracks the Code's structure closely. And once tax law is computationally accessible, tax shelters will be easier to generate.

---

140. *See id.* at 784.
141. *See id.* at 790.
142. *Id.* at 784.
143. *Id.* at 781.
144. *Id.*
145. Victor Fleischer, *Regulatory Arbitrage*, 89 TEX. L. REV. 227, 244 (2010).
146. *See generally* DAVID CAY JOHNSTON, PERFECTLY LEGAL: THE COVERT CAMPAIGN TO RIG OUR TAX SYSTEM TO BENEFIT THE SUPER RICH—AND CHEAT EVERYBODY ELSE *passim* (2005) (outlining the myriad tax loopholes used by large corporations and the super-rich); Henry Ordower, *The Culture of Tax Avoidance*, 55 ST. LOUIS L.J. 47, 123–24 (2010).
147. *See* Jacob Rosen, Computer Aided Tax Avoidance Policy Analysis (May 8, 2015) (M.S. thesis, Massachusetts Institute of Technology) (on file with the Massachusetts Institute of Technology); Hemberg, Rosen, Warner, Wijesinghe & O'Reilly, *supra* note 123, at 79–88.

If the IRS has its own computationally accessible tax law, it could use its own computer programs to create and flag tax shelters. The IRS already lists transactions it considers tax shelters, but this is done *ex post* based on shelters it has encountered from taxpayers. [148] Having its own computationally accessible tax law could permit the IRS to list problematic transactions *ex ante*.[149] Defining tax shelters is notoriously difficult.[150] But with computationally accessible tax law, instead of attempting to characterize or theorize the type of transaction not permitted, the IRS could essentially brute force the problem with its own program and provide a list of forbidden transactions. Allowing private individuals or industries to create computationally accessible tax laws without the government's also having such a corpus could be disastrous, as wealthy taxpayers could potentially identify and implement tax shelters at little or no marginal cost.

Ideally, the computer code of the tax law and the programs necessary to use this code would not be limited to the government but open-source and accessible to all. This openness would increase transparency and accountability, which are benefits on their own. Open source tax code would also give individuals and nonprofits the ability to flag tax shelters for the government. The tax bar has a long history of working with the government to improve tax law and administration.[151] Nonetheless, private tax lawyers work primarily for their clients, not for the government, and when it comes to tax enforcement in general and tax shelters in particular, private tax lawyers, as David Schizer has explained, in many ways

148. *See* Treas. Reg. § 1.6011-4(b)(2) ("A listed transaction is a transaction that is the same as or substantially similar to one of the types of transactions that the Internal Revenue Service (IRS) has determined to be a tax avoidance transaction and identified by notice, regulation, or other form of published guidance as a listed transaction."); I.R.C. § 6501(c)(10) (extending period of limitations with respect to undisclosed listed transactions); *id.* § 6707(a) (imposing penalties for undisclosed listed transactions). The reportable transactions regime, including listing specific transactions that the IRS or Treasury deems problematic, has its own problems. For example, the target activities are often either too broadly or too narrowly defined, and the concept of "similarity" can be difficult to capture. *See, e.g.*, Joshua D. Blank & Ari Glogower, *The Trouble with Targeting Tax Shelters*, 74 ADMIN. L. REV. 69, 77 (2022).

149. It may be impossible for the IRS to generate all possible shelters, because there may be no limit to how long the chain of statutes and regulations could be. (An interesting question, and again totally outside the scope of this Article, is whether it would ever be possible to show that no further shelters could be generated.) One could imagine a system in which the IRS classified transactions by number of statutory steps in the chain and banned all transactions with more than a certain number. This might allow an exhaustive listing of shelters, which in turn might be able to address some of the problems currently facing listed transactions as described in Blank & Glogower, *supra* note 148.

150. Michael Graetz has perhaps the best explanation: "It is easy to define a tax shelter for the press or the layman: Tax shelters are 'deal[s] done by very smart people that, absent tax considerations, would be very stupid.' But translating this definition into legislative language to defeat tax-shelter transactions and to justify enhanced penalties is another matter altogether." Graetz, *supra* note 119, at 278 (internal citations omitted).

151. For example, the American Bar Association Tax Section and the New York State Bar Association Tax Section regularly provide analysis and comments on proposed regulations. *See Section of Taxation*, AM. BAR ASS'N, https://www.americanbar.org/groups/taxation [https://perma.cc/K4UY-BHNR]; Tax Section, N.Y. STATE BAR ASS'N, https://nysba.org/committees/tax-section [https://perma.cc/ZX5X-EZB5].

"outmatch" the government in staffing, resources, and knowledge.[152] Open source code for the tax law could help others with an interest in locating and shutting down shelters balance that mismatch.

V. CONCLUSION

Tax law is already formalized, in a patchwork of inaccessible, obscure, and sometimes inaccurate formalizations, thus reducing both the general usefulness of the formalizations and government accountability. The new programming language Catala is designed to formalize tax law accurately and accountably. This formalization could have benefits for tax law administration and for government accountability, but formalizing tax law could also make tax shelters easier to create. Private individuals and businesses therefore have a significant incentive to use Catala or another domain-specific language to codify tax law to their own benefit. To affirmatively help taxpayers and tax administration, and to guard against increased tax avoidance, the government should act as soon as possible to create a computationally accessible version of tax law.

152. David M. Schizer, *Enlisting the Tax Bar*, 59 TAX L. REV. 331, 331 (2006).