Catala: Moving Towards the Future of Legal Expert Systems

Denis Merigoux, Liane Huttner

CATALA: Moving Towards the Future of Legal Expert Systems

Denis MERIGOUX [a], Liane HUTTNER [b]
[a] *Inria, Paris*
[b] *Université Panthéon-Sorbonne, Paris*

Abstract Software called legal expert systems are used around the world by private and public organizations to compute taxes. A bug in such programs can lead to tax miscalculations and heavy legal and democratic consequences. Yet, increasing evidence suggests that some legal expert systems may not meet satisfying criteria to be in compliance with the law. Moreover, they are difficult to adapt to the continuous flow of new legislation just by using traditional software development process. To prevent further software decay and reconcile these systems with the growing demand for algorithmic transparency, we argue that there is a need for a new development process for legal expert systems. As such, we present a solution built by lawyers and computer scientists : a new programming language coupled with a pair programming development process.

Keywords. legal expert systems, formal methods, literate programming, algorithmic transparency, tax law, social benefits

1. Introduction

How do governments compute taxes and social benefits for their citizens? Many have chosen to solve this complicated task by using computer programs. They have created software, modeled on the law, which computes the amount of tax due and the social benefits to be paid to each citizen.

These software, called "legal expert systems", are based on the law. In other words, the program and its reference statute ought to be functionally equivalent. This means that any result output by the computer program should match a correct legal reasoning based on the statute. A breach of functional equivalence in such programs, which we call a bug, places the entity running the program in legal jeopardy.

Two questions arise from this situation. The first one is legal and social: what are the consequences of bugs in programs computing taxes? The second is technical: how to decrease the number of bugs in those programs, and increase the confidence that they faithfully implement the statutes they use as a reference?

We will answer those questions with a survey of the state of the art of the computerized implementation of tax-related statutes, with a focus on French examples (§2). Then, we will present the legal and democratic implications of bugs in legal expert systems (§3). Finally, we will introduce Catala, a new programming language created by lawyers and computer scientists for quantitative statute formalization. The goal of Catala is to provide a systematic way to produce bug-free programs from tax-related statutes, that can be deployed and executed on virtually any digital infrastructure, including legacy. Thanks to an *ex ante* systematic human review, Catala-created software will also comply with data protection law (§4).

This initiative belongs to the broader "rules as code" movement. However, we deliberately omit from this article considerations on co-designing legal statutes and computer programs. Rather, we take existing statutes for granted. Exploring the effectiveness of Catala as a co-design tool is left for future work.

2. Existing Algorithmic Implementations of Statutes

Drawing from US and French Law, this section will provide an overview of usages and failures of algorithmic implementation of statutes.

2.1. Examples in US and French Law

Although there has been a boom in studies on the use of artificial intelligence tools by governments, algorithmic tools implemented in legal expert systems such as models or scoring systems have been relatively neglected [1]. However, these tools are widely used by governments and private companies to calculate social benefits and tax. Two examples, one drawn from US law and the other from French law, will show how public and private entities use algorithms for administrative purposes.

In the US, taxes are collected by the Federal government as well as by individual States. It is the responsibility of the taxpayer to compute for themselves the correct amount of taxes that they owe, depending on their income. Because of the complexity of this task, private companies have developed legal expert systems in order to help individuals fill out their tax forms. The most widely used program is *TurboTax* [2]. There's reason to believe that this software is based on the tax forms prepared by the government [3]. Hence, it is on the basis of these forms, themselves based on the US Tax Code, that an individual's tax is computed. The IRS (US Tax Agency) also operates an internal legal expert system to check tax returns once they are submitted[1].

In France, taxes are also computed by a legal expert system. Unlike in the US, though, the French government itself develops and maintains the software which computes the amount of taxes due. Led by the Directorate of Public Finances (DGFiP), some fragments of this software have been rendered public and are available online. Social benefits algorithms, however, are not widely published, though they can theoretically be accessed upon request.

Even if the size and responsibilities of the public sector for computing taxes is greater in France than in the US, some tax-related algorithms are commonly operated by the private sector. This is the case of private sector employees' payroll taxes and contributions to Social Security, computed by closed-source legal expert systems of companies such as ADP or PayFit. In both the US and France, software is used as a means to simplify tax collection and the distribution of social benefits. But as we will see next, current legal expert systems have some major issues and may even fail to comply with the law. In the next section, we will use a French example to illustrate the consequences of bugs.

2.2. Legacy Code and Industrial Failures: the French Example

The legal expert systems responsible for tax computation in large organizations sometimes correspond to what is called "legacy code". Legacy code is a term coined by the software industry to designate large, complex systems whose lifespan has exceeded the tenure of its original programmers [4]. According to scholars, legal expert systems used in large financial audit firms were created around 1990 [5, 6, 7] and are still in use today in at least some private companies[2]. For the public sector, the French systems for income tax [8] and the family benefits computation[3], were both created in 1990. The IRS system is even older and it is still operating 1950-era software and hardware, in spite of multiple failed modernization attempts [9].

Legal expert systems quickly suffered from the discontentment of users because of their poor usability [10]. After more than 30 years of existence, they have now acquired

[1] Source: Internal Revenue Manual.

[2] Source: representatives from Ernst & Young, PwC, Deloitte and KPMG at the "Machine Intelligence and the Future of Professional Tax Services" panel during the 2020 UC Irvine Tax Symposium.

[3] Source: French Commission for Accessing Administrative Documents, notice n°20181891, 2019

all the general characteristics of legacy code: use of obsolete technologies no longer taught in university courses, loss of expertise on critical portions of the source code, as original programmers retire, and a "plaster on a wooden-leg" approach to modification and maintenance [4].

These characteristics jeopardize the ability of the system to be adapted to new functional requirements. For example, in the case of systems implementing a statute, any modification of the statute entails a corresponding modification of the software. But because of the complexity and the fast-paced nature of tax law reforms, updates have to be frequent. As a consequence, maintenance of the systems becomes very difficult and costly.

The traditional solution to legacy code is migration [11]. Migration boils down to creating a new system from scratch using modern technologies. During the migration, both systems have to run in parallel to ensure that they reach the same results[4] [12]. However, in France, several migrations attempts have led to a number of high-profile industrial failures.

More precisely, two migrations for public-sector legal expert systems have had catastrophic consequences for their users : the Louvois[5] army payroll computation system and the CIPAV[6] "auto-entrepreneur" pension rights computation system. In both cases, the State contracted a private company to undertake this task, but it failed to implement the legal specifications correctly. After years of maintenance and bug-fixing, these two migrated systems were still producing unreliable outputs, requiring extensive human supervision. This suggests that the complexity of the legal landscape has increased since 1990 [13]. So much so that implementing correctly a legal expert system from scratch seems to be now out of reach from 1990-era traditional development methods, still widely used in the software industry [14].

As these examples show, the current situation of legal expert systems is paradoxical. Even though their use is pervasive and critical in many large public or private organizations, they have become legacy code increasingly hostile to maintenance and migration. While it seems that there hasn't been any recent survey about the correctness of legal expert systems, the facts presented above should cast serious doubts on whether these systems are functionally equivalent to their reference statutes. And, as we will show in the following part of the article, bugs have serious legal and democratic implications.

3. Issues in Algorithmic Implementations of Statutes

Incorrect implementations of statutes have legal and democratic implications. Some solutions exist but they fail to address all of the implications.

3.1. Legal and Democratic Implications

The shortcomings of legal expert systems have two important consequences, which we will examine in the context of French and EU law. First, bugs can lead to miscalculations, and such miscalculations are in breach of the French Constitution. Second, because of the enormous scale of the tax collection, it is impossible for an individual to systematically review the results generated by these algorithmic tools. Such a barrier to human intervention might be a breach of data protection law.

To begin with miscalculations: bugs in legal expert systems used by governments or private companies can lead to errors in the computation of taxes and social benefits. For example, in 2009, a French retirement scheme miscalculated pensions. For several

[4] Any disagreement should correspond to a bug in the old system.

[5] Source: France Inter, Jan. 2018

[6] Source: France Inter, Jan. 2016. Hundreds of subscribers were threatened to lose their pension rights.

years, the program failed to provide the correct amount of money to its beneficiaries[7]. However, because such errors are sensitive matters, governments rarely acknowledge such bugs. As a consequence, literature on the subject is difficult to find. Nonetheless, miscalculation of taxes and social benefits is a breach of the French Constitution. For example, concerning tax law, article 34 of the Constitution states that "Statutes shall determine the rules concerning the base, rates and methods of collection of all types of taxes; the issuing of currency". Known as the principle of *tax legality*, this constitutional principle is interpreted in a broad manner. As a result, the French Parliament has authority not only to determine general principles of tax law but also to decide on every detail of tax collection. The executive power and the administration therefore have almost no authority over tax law. Thus, when the administration is responsible for miscalculations resulting from their use of algorithms, they are in breach not only the law but also of the Constitution. This is, of course, a serious democratic issue.

Secondly, if the results of the legal expert system are not reviewed by a human, they might be in breach of International and European Law. Indeed, this would fall under article 9 of the Convention 108+ of the Council of Europe, which gives every individual the right not to be subject to a significant decision based solely on automated processing of data[8]. Similar protection can be found under article 22 of the GDPR. Article 22 of the GDPR states that individuals have the right not to be subject to a decision based solely on automated processing which produces legal effects. While there are discussions on the exact meaning of article 22, there is a consensus that it refers to decisions excluding human involvement. In other words, an automated decision happens when no human reviews the results[9].

There are exceptions to this right. For example, Member States can grant specific exemptions. Nevertheless, even when such exemptions are applicable, suitable measures must be put in place to safeguard the data subject's rights, freedoms, and legitimate interests. In France, for example, such an exemption was put in place in 2018 for administrative purposes. As a result, the government can now use automated tools to make decisions, including the calculation of taxes[10]. But since legal expert systems are created to automate tax calculations, there is, by definition, no human involvement and no human verification of the results, meaning that, if these algorithms malfunction or lack suitable safeguards against negative impacts on individual rights, freedoms, and legitimate interests, they are in breach of article 22 of the GDPR.

Breaching the Constitution and data protection law have severe consequences. Tax collection which does not comply with the Tax Code can be nullified. As for data protection, infringements of article 22 of the GDPR can lead to administrative fines up to 20 million euros.

3.2. Problems with Existing Solutions

There are several existing solutions to prevent the legal and democratic consequences of bugs. First, preventing bugs from happening in the first place through case-based testing. Second, improving transparency of algorithms and make them more accessible for maintenance. The two solutions are difficult to implement, as we will see, because of a host of political and technical difficulties.

3.2.1. First Solution : Case Based Testing

Let's begin with case-based testing. Like any piece of software, legal expert systems programmers take measures to locate and fix bugs. Finding a bug in a legal expert system

[7] Source: press release of the French national pension agency (CNAV), May 13th, 2009.

[8] Convention 108+ for the protection of individuals with regard to the processing of personal data.

[9] Article 29 Data Protection Working Party. Guidelines on Automated individual decision-making and Profiling for the purposes of Regulation 2016/679, 3 October 2017.

[10] Article 21, *Loi n°2018-493 du 20 juin 2018 relative à la protection des données personnelles*

requires interaction between lawyers and programmers. The process is the following[11]. First, lawyers create a virtual test case or pick one from production data (*e.g.* a household fiscal data). This will constitue the set of test cases. Afterwards, they manually compute the expected output of the legal expert system based on the reference statute. They then compare the output of the system with the lawyer's expected output. In the case of a disagreement, lawyers and programmers discuss how they got to their results. When the discrepancy is located, the software is updated to output the correct result. The most important phase of the process is the discussion between programmers and lawyers. This is the phase where the legal requirements, with all their subtleties and varying interpretations [15], are confronted to unambiguous computer code.

While this process is effective in improving the quality of the legal expert system, it suffers from the same limitations software testing is generally subject to. Indeed, program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence [16]. But most importantly, testing efficiency is tied to the quality of the set of test cases used. For testing to be efficient, the set of test cases has to be diverse and numerous. It's only in this condition that the integrality of the code can be checked for correctness. This poses various issues.

Firstly, in the case of legal expert systems, creating new tests is costly since it requires legal expertise. A legal expert system typically handles thousands of distinct situations. To reach full coverage, a test base should be able to deal with all of these situations. While it is difficult to get accurate and recent data on legal expert systems test bases, the French income tax computation system uses around 500 tests (for approximately a hundred thousand lines of code)[12]. This suggests that legal expert systems are currently under-tested.

Moreover, the set of test cases is updated along with the software as the legal requirements change. This means that for each statute modification, each test of the set should be manually reviewed by a lawyer to determine whether it is affected or not by the modification. For this reason, keeping a complete and correct test set up to date is very costly.

Beyond mismanagement and poor software quality, it seems that there is currently no systemic incentive for large public or private organizations to maintain a diverse and numerous test set for their legal expert systems. If an individual contests the result of the algorithms, the organization simply provides for a manual human review of their case. In the best scenario, the case is turned into a test for the system[13]. But most of the time, it seems that the software is never updated and human agents have to manually correct the output in future occurrences of the buggy situation.

Overall, case-based testing is a good way to trigger interaction between programmers and lawyers, leading to a better legal expert system. But the high cost of maintenance for a good test set, coupled to low incentives for covering corner cases correctly, have *de facto* led to low levels of confidence for legal expert systems. Moreover, case-based testing does not help to achieve algorithmic transparency, especially if the test set contains production data that cannot be revealed.

3.2.2. Second Solution : Transparency

The second solution, algorithmic transparency is often seen as a strong safeguard against issues arising from automated decision-making [17, 18]. As a consequence, European and French law have flirted with the idea of making such transparency a legal obligation. Under the GDPR, articles 13, 14 and 15 give every individual the right to access mean-

[11] Source: author's private discussions with public sector French legal expert systems programmers and publicly available beta.gouv.fr blog post (Feb. 2020)

[12] Source: information personally transmitted by the French Directorate of Public Finances (DGFiP) to the authors.

[13] This behavior can lead to privacy violations when the data is covered by tax secrecy for instance.

ingful information about the logic involved in automated decision-making. While there has been debate about the meaning of this right, it is generally accepted that individuals have the right to ask for access to information on algorithmic processing [19, 20, 21]. This, however, is far from a general transparency requirement. This holds true in France, as well. Administrations have to give access to meaningful information on the logic of administrative algorithms[14]. Every individual can ask to access these algorithms, under the supervision of an independent commission. As a consequence, the government created an online platform where source code can be found.

But the publication of algorithms is scarce and complicated. The source code for income tax computation, for example, was published in an incomplete form [8]. Access to the source code for family benefits computations has been denied because "since the legal expert system is old and complex, the extraction of the source code for publication is not technically possible without disproportionate effort". As a testament to these difficulties, a French Member of Parliament was recently appointed to investigate the issues with administrative source code publication[15]. The US situation is better in that regard, since the IRS regularly publishes draft versions of its tax forms for public review.

4. CATALA as a New Solution for Algorithmic Implementations of Statutes

As we've seen, current legal expert systems are outdated and suffer from a lack of confidence on the correctness of their results, as well as general opacity that conflicts with the growing demand for algorithmic transparency. In this section, we introduce a new production process for legal expert systems along with appropriate tooling that answers to the issues raised above. This production process is based on two complementary concepts : formal methods and literate pair programming.

4.1. The Use of Formal Methods

Formal methods are a subdomain of computer science, sharing close ties with mathematics. Its premise is to consider computer programs as mathematical objects [22], on which theories can be applied and properties, such as correctness or safety, proven. Formal methods have been deployed in critical industrial sectors like avionics [23] or nuclear energy production [24]. Indeed, these techniques are able to completely rule out entire classes of bugs from computer programs, including memory safety bugs or software crashes.

Another achievement of formal methods is the ability to prove functional equivalence between a program and its specification, assuming that both can be expressed using formalized languages, *i.e.* whose behavior is described precisely using mathematical terms. This ability is of high interest for legal expert systems. Given a formal specification of a legal statute, it is possible to use formal methods to produce an executable implementation that is guaranteed to behave in the exact same way. An analogous process has been used for software controlling critical industrial facilities in real time [25].

More generally, formal methods shift the discussion about correctness from individual test cases to the source code of the program itself, whose behavior is considered for all possible inputs. We believe that reference statutes have to be considered as the ultimate specification of legal expert systems. Hence, the correctness problem of legal expert systems boils down to agreeing on a reference formal specification of the statutes. Optimized and executable implementations should be derived from this specification using advanced compilation techniques [26] that preserve the functional equivalence between the statute and the software that is produced from it.

[14] Article L.311-3-1 of the French code on relations between the administration and individuals

[15] Source: French decree of June 22th, 2020.

Interestingly, formal methods have already been used to formalize part of statutes [27]. Consequently, the question raised sporadic academic interest, but has never led to large-scale deployments in the public sector. The most advanced project in that category is certainly the French-led OpenFisca[16]. OpenFisca now features a comprehensive legal expert system able to compute the amount of almost all French taxes and benefits. However, OpenFisca does not have a formal grounding and the system has yet to be deployed in a government agency responsible for the collection of taxes or distribution of benefits.

Building on formal methods, our solution also features literate pair programming.

4.2. The Use of Literate Pair Programming

While literate pair programming is a clear advantage for the implementation of statutes, it faces a particular challenge when it comes to law. This challenge originates from the special structure of law. One of the most important feature of CATALA is, consequently, to adress this challenge.

4.2.1. Presentation and General Advantages

Our solution combines two software development processes: literate programming and pair programming. To begin with literate programming, the source code of a program is annotated line by line with a textual description of what the program is supposed to do [28]. This a systematic approach to documentation, and a good fit for programs whose behavior is subtle or difficult to infer just by looking at the code.

On the other hand, pair programming is part of the agile process of software development [29]. It consists of pairing two programmers when producing software. While one of the programmers is busy writing the code, the other programmer can think about more high-level aspects of the software, or catch bugs as they're being written.

Both literate and pair-programming are used as *ex ante* and systematic ways of increasing the quality of software. It is relevant to combine these two concepts and use them for legal expert systems.

First, let us examine the advantages of literate pair programming in the context of legal expert systems. We have shown that interaction between lawyers and programmers is crucial for debugging legal expert systems. During this interaction, both parties play a crucial and complementary role: lawyers ensure that the specification reflects lawful interpretations of the statutes, while programmers ensure that the specification is completely unambiguous, and can be turned into an executable program.

However, this systematic interaction cannot be achieved with traditional waterfalls or V-shaped software development processes. With such processes, lawyers produce first a verbose natural language specification document from the statutes. The programmers then translate this verbose specification document into code. We identity three pitfalls. First, the lawyers don't know whether their specification document is sufficiently unambiguous to be turned into code. Second, when confronted to ambiguous or imprecise specifications, the programmers make arbitrary decisions that may correspond to unlawful code. Third, there is no direct and systematic connection between any piece of the source code and the piece of the statute that justify it.

Literate pair programming solves all those pitfalls: a lawyer and a programmer can produce together (pair programming) the legal expert system by gradually annotating the law with code translation (literate programming). When each line of statutory text is annotated with a line of code that translates its meaning, lawyers and programmers can have a local discussion with a visual support about a specific legal requirement. This format should foster mutual understanding, and eventually build cross-competence for both the lawyer and the programmer. With this method, interdisciplinary interaction is systematic and *ex ante*, by contrast to case-based interaction which is limited and *ex*

[16] `openfisca.org`

post. Using agile methods can also significantly decrease the cost of software production. The French portal for social benefits computation, mes-aides.gouv.fr, has been developed from scratch using agile methods for a total cost of 1.25 million euros (over 5 years). When its maintenance was transferred to a private company using more traditional development processes, the cost skyrocketed to at least 2 million euros annually[17].

4.2.2. The Particular Challenge for Law

However, to use literate pair programming for legal expert system production, adequate tooling is needed. Indeed, the structure of statutes is not adapted to traditional literate programming. This is demonstrated by [30] in the case of the US Tax Code, but we've empirically observed that the results apply for French statutes, because of similarities in the drafting style. The crux of the issue is a antagonism between the structure of law and the structure of computer programs. Indeed, normal programming goes from the most special case to the most general case. Statutes do the opposite. Because of this, implementations of statutes need to resort to impractical encodings based on nested conditionals, obscuring the behavior of the code.

This main hurdle, coupled with the lack of existing tooling solving it, has led us to create a new programming language designed to enable statute literate programming: CATALA[18]. Equipped with this appropriate tool, we ideally envision for CATALA statutes specifications to be published as open-source software, complementing existing publications lof legislative texts. Coupled with state-of-the-art, formalism grounded, open-source compilation and interpretation tooling, this new method of producing legal expert systems could help to solve both the correctness and explainability issues of current solutions while driving legal expert systems maintenance costs down. All these features are present in CATALA[19].

4.3. The Technical Overview of CATALA

A complete technical description of the CATALA language and compilation ecosystem is beyond the scope of this paper, and is left to a future computer science-focused article. However, we can give here a brief overview of the key technical decisions and their consequence.

Concerning formal methods, CATALA enjoys a design process guided by the best practices of programming languages research. More particularly, it has a formal semantics [32], comprised of three complementary items: a description of the syntax of CATALA programs, a typing judgment that rules out ill-formed programs, and an operational semantics describing how valid programs execute. The core semantic feature of CATALA is its use of default logic [33] to enable defining a variable multiple times with preconditions. Multiple conditions can be triggered at the same time, for instance when the law defines two exceptional cases that overlap. In that case, the programmer has to specify (and justify by law) a priority between the conflicting exceptions. If no such priority is given, the execution will report an error to the user, indicating a black spot requiring legal interpretation.

Because of its formal grounding, existing off-the-shelf static analysers and automatic proves could be used on CATALA code. They could be used to check the coherence of the statute (no conflict between exception) or whether the statute is valid with respect to requirements of another statute, higher in the hierarchy of statutes. For instance, article L521-1 of the French Social Security Code mandates that the amount of family benefits decreases with the household's income. The formula giving the amount, defined in several other articles, can be formally checked to satisfy this requirement.

[17] Source: beta.gouv.fr evaluation of the project.

[18] Pierre Catala is, together with Lucien Mehl, a pioneer of French legal informatics [31].

[19] `catala-lang.org`

The use of formal methods also enable the application of compilation techniques that can translate CATALA to virtually any language. This is where CATALA differs from existing propositions of so-called "rules engine". These rules engine, like Flora2 [34] or Prolog [35] (recently used in a tax-related context by [36]), are based on logic programming and executed using interpreters. This means that to use *e.g.* a Prolog program in your application written in Java or C++, you have to call an outside program that will interpret the Prolog program and store its outputs in a file or database, then load back those outputs in the context of your language. Moreover, interpreters are generally inefficient and can lead to performance losses that are unacceptable when computing taxes for millions of individuals. By using advanced compilation techniques, CATALA programs can be compiled (translated) to any general-purpose programming language, including legacy languages like COBOL or Fortran. This interoperability scheme is much more efficient, both in terms of program performance as well as development overhead. It also allows for separating the tax computing logic from the other parts of the system.

Last benefit of formal methods and compilation techniques: explainability requirements can also be addressed by a special compilation scheme that insert logging of the program's execution. Each log entry corresponds to a source code line, and therefore the statutory text provision that it annotates. This scheme would address the needs for both individual and global algorithmic explainability, as long as the source code is open-source.

Concerning literate pair programming, CATALA has been designed with pair programming and lawyers' review in mind. As such, it enjoys syntax with natural language keywords that can be adapted to different countries. Right now, CATALA supports French and English inputs. The CATALA compiler is distributed under the Apache2 license and freely available on GitHub. Future work will include case studies on the French family benefit computation as well as more content from the US Tax Code.

5. Conclusion

CATALA provides a solution for many of the problems adressed in this article. It reduces bugs and improves transparency in legal expert systems, and is, as such, in compliance with legal obligations. It also answers questions arising from changes in socio-economical context. To address the consequences of this change, organizations that operate legal expert systems should proactively seek to modernize their software infrastructure. As we have shown, it is likely that using traditional development processes will lead to industrial failures. But in the setting of an agile development process, CATALA would be the perfect tool with which to build correct and explainable new legal expert systems.

References

[1] Cary Coglianese and Lavi Ben Dor. AI in adjudication and administration: A status report on governmental use of algorithmic tools in the United States. *U of Penn Law School, Public Law Research Paper*, (19-41), 2019.

[2] JB Ruhl and Daniel Martin Katz. Measuring, monitoring, and managing legal complexity. *Iowa L. Rev.*, 101:191, 2015.

[3] Sarah B Lawsky. Form as formalization. *Ohio State Technology Law Journal*, 2020.

[4] Belfrit Victor. Revisiting legacy systems and legacy modernization from the industrial perspective. Master's thesis, 2013.

[5] Donald A. Waterman, Jody Paul, and Mark Peterson. Expert systems for legal decision making. *Expert Systems*, 3(4):212–226, 1986.

[6] Michael Z. Bell. Why expert systems fail. *Journal of the Operational Research Society*, 36(7):613–619, 1985.

[7] C.E. Brown and David Murphy. The use of auditing expert systems in public accounting. *Journal of Information Systems*, pages 63–72, 01 1990.

[8] Denis Merigoux, Raphaël Monat, and Christophe Gaie. Étude formelle de l'implémentation du code des impôts. In *31ème Journées Francophones des Langages Applicatifs*, Gruissan, France, January 2020.

[9] Andrew Whitmore, Eliot Rich, and Mark R Nelson. Building on shifting sands: The structure of repetitive it project escalation, crisis, and de-escalation. In *Proceedings of the 26th International Conference of the System Dynamics Society, Athens, Greece*, 2008.

[10] Philip Leith. The rise and fall of the legal expert system. *International Review of Law, Computers & Technology*, 30(3):94–106, 2016.

[11] A. Sivagnana Ganesan and T. Chithralekha. A survey on survey of migration of legacy systems. In *Proceedings of the International Conference on Informatics and Analytics*, ICIA-16, New York, NY, USA, 2016. Association for Computing Machinery.

[12] Christophe Gaie. From secured legacy systems to interoperable services (the careful evolution of the french tax administration to provide new possibilities while ensuring the primary tax recovering objective). 2020.

[13] Stephane Cottin. Computer-assisted law-making-process: Information technologies and legal certainty – French experiments. In *International Conference, The State and the Legal System – Institutional Contemporary Transformations*, 2006.

[14] Phillip A. Laplante and Colin J. Neill. The demise of the waterfall model is imminent. *Queue*, 1(10): 10–15, February 2004.

[15] Sarah B. Lawsky. Formalizing the Code. *Tax Law Review*, 70(377), 2017.

[16] Edsger W Dijkstra. The humble programmer. *Communications of the ACM*, 15(10):859–866, 1972.

[17] Alyssa M Carlson. The need for transparency in the age of predictive sentencing algorithms. *Iowa L. Rev.*, 103:303, 2017.

[18] Sandra Wachter, Brent Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harv. JL & Tech.*, 31:841, 2017.

[19] Sandra Wachter, Brent Mittelstadt, and Luciano Floridi. Why a right to explanation of automated decision-making does not exist in the general data protection regulation. *International Data Privacy Law*, 7(2):76–99, 2017.

[20] Emre Bayamlıoğlu. Transparency of automated decisions in the GDPR: an attempt for systemisation. *International Data Privacy Law*, 2017.

[21] Bryan Casey, Ashkon Farhangi, and Roland Vogl. ethinking explainable machines: The GDPR's "right to explanation" debate and the rise of algorithmic audits in enterprise. *Berkeley Technology Law Journal*, 34, 2019.

[22] William A Howard. The formulae-as-types notion of construction. *To HB Curry: essays on combinatory logic, lambda calculus and formalism*, 44:479–490, 1980.

[23] Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. Design and Implementation of a Special-Purpose Static Program Analyzer for Safety-Critical Real-Time Embedded Software. In Mogensen, T., Schmidt, D.A., Sudborough, and I.H., editors, *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, volume 2566 of *Lecture Notes in Computer Science*, pages 85–108. Springer, 2002.

[24] Pascal Cuoq, Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski. Frama-c. In George Eleftherakis, Mike Hinchey, and Mike Holcombe, editors, *Software Engineering and Formal Methods*, pages 233–247, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[25] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. Lustre: A declarative language for programming synchronous systems. In *In 14th Symposium on Principles of Programming Languages (POPL'87). ACM*, 1987.

[26] Xavier Leroy. Formal certification of a compiler back-end or: Programming a compiler with a proof assistant. *SIGPLAN Not.*, 41(1):42–54, January 2006.

[27] Jason Morris. Spreadsheets for legal reasoning: The continued promise of declarative logic programming in law. *Available at SSRN 3577239*, 2020.

[28] D. E. Knuth. Literate Programming. *The Computer Journal*, 27(2):97–111, 01 1984.

[29] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for agile software development. 2001.

[30] Sarah B. Lawsky. A Logic for Statutes. *Florida Tax Review*, 2018.

[31] Pierre Catala, Lucien Mehl, and Edmond Bertrand. Constitution et exploitation informatique d'un ensemble documentaire en droit (droit de l'urbanisme et de de la construction).

[32] A.K. Wright and M. Felleisen. A syntactic approach to type soundness. *Information and Computation*, 115(1):38 – 94, 1994.

[33] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1):81 – 132, 1980. Special Issue on Non-Monotonic Logic.

[34] Guizhen Yang, Michael Kifer, and Chang Zhao. Flora-2: A rule-based knowledge representation and inference infrastructure for the semantic web. In Robert Meersman, Zahir Tari, and Douglas C. Schmidt, editors, *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, pages 671–688, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[35] Alain Colmerauer and Philippe Roussel. *The Birth of Prolog*, page 331–367. Association for Computing Machinery, New York, NY, USA, 1996.

[36] Nils Holzenberger, Andrew Blair-Stanek, and Benjamin Van Durme. A dataset for statutory reasoning in tax law entailment and question answering. *arXiv preprint arXiv:2005.05257*, 2020.