Experience report: implementing a real-world, medium-sized program derived from a legislative specification

Denis Merigoux

▶ **To cite this version:**

Denis Merigoux. Experience report: implementing a real-world, medium-sized program derived from a legislative specification. Programming Languages and the Law 2023 (affiliated with POPL), Jan 2023, Boston (MA), United States. hal-03933574

HAL Id: hal-03933574
https://inria.hal.science/hal-03933574

Submitted on 10 Jan 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Experience report: implementing a real-world, medium-sized program derived from a legislative specification

DENIS MERIGOUX, Inria, France

Additional Key Words and Phrases: legal expert systems, catala, hosuing benefits, legal formalization

1 INTRODUCTION

Implementing computer programs from legislative specifications has been a longstanding academic endeavor [Allen 1956; Bench-Capon et al. 1987; Dewey 1924; McCarty 1977, 1989, 1997; Sergot et al. 1986]. However, few of these experiments has so far sought to *replicate* a computer program that is already in production in public administration; most of them have created *new* programs or formalizations that cover sections of the law not previously automatically enforced by public administration. A somewhat dated but accurate state of the art of the use of legal expert systems in government agencies was compiled by Oskamp and Lauritsen [2002].

Building on the work around the Catala domain-specific language [Huttner and Merigoux 2022; Merigoux et al. 2021a], we have chosen to engage in an exercise of replication of the existing IT system that computes the French housing benefits (and various other benefits) within the CNAF administrative agency, CRISTAL. More general context and main non-technical findings of this replication exercise can be found in Merigoux et al. [2023]; this presentation will focus on the challenges and lessons learned about the programming act itself, in an effort of consolidation of knowledge for this line of research.

Housing benefits in France are distributed to approximately 5 million households, for a yearly total amount of about 18 billion euros. Each household must declare its family and housing situation to the CNAF or CCMSA administrative agencies, that are tasked with computing and distributing the correct amount of housing benefits entitled to this household. The rules for computing the amount of benefit entitled from the family and housing situation are scattered in various legislative and regulatory documents, ranging from the Construction and Housing Code to the executive order of September 27$^{\text{th}}$, 2019 relative to the computation of personal housing benefits.

As permitted by French law, we asked the source code of CRISTAL in 2020 and obtained a CD-ROM containing a snapshot of COBOL sources for approximately 6 million lines of code, similarly to the previous experiment of Berne [2018]. These COBOL files, lacking any documentation about how to compile or run them, appear to be generated by a Computer-Aided Software Engineering (CASE) tool, whose high level sources have not been published. In other terms, what the CNAF published is merely the low-level source code, not meant to be read or edited by humans, while the real high-level source edited through a CASE tool is probably still hidden behind the CNAF curtains. Moreover, we received in November 2021 during subsequent discussion with the CNAF another excerpt from the source of the housing benefits computation. This excerpt was written using Oracle's proprietary Oracle Intelligent Advisor (OIA) programming language, as it appears that the CNAF had recently undertaken a rewrite of the housing benefits computation algorithm using this new technology, as a part of a grand scheme to modernize the CRISTAL system. We asked for the rest of the source code written in OIA but got no answer.

Author's address: Denis Merigoux, denis.merigoux@inria.fr, Inria, Paris, France.

Being unable to exploit the existing official codebase, we decided to replicate it. The motivation for tackling this particular piece of legislation comes from frequent media occurrences of dysfunctions in the CRISTAL system [Knaebel 2022; Monnet 2022; Zerouala 2021]. We believe that the replication of this system using the Catala language and methodology can help improve the transparency and correctness of the existing system, by comparing its output with the existing CRISTAL system. We have already collected our recommendation to the CNAF regulators in an administrative report [Merigoux 2022].

The contributions of this presentation are the following: a detailed account of the programming methodology used, a summary of the legal difficulties encountered, and a deep focus on the semantic and language constructs needed to correctly express the computational content of the French housing benefit legislation.

2 PROGRAMMING METHODOLOGY

In absence of an exploitable codebase, we decided to start the programming from scratch and from the official legal sources describing the computation of the housing benefits, following the Catala methodology mentioned in Huttner and Merigoux [2022]. The main authors of the program include a programmer very proficient in the Catala language (A) and a lawyer initially specialized in intellectual property law (B). Another programer (C) switched with A for a small part of the development. None of the three had any prior experience of the French housing benefits computation.

The development effort spanned 9 months from December 2021 to August 2022, but the developers were only working part-time on the implementation during this period, as administrative and publication-related tasks take up a significant amount of time in a research lab. The time spent in programming and programming-related tasks has been self-reported in order to produce accurate estimates of the effort involved in the project. These estimates do not include the research and development time invested in the building of the Catala compiler and tooling; the goal is to measure the marginal cost of implementing a real-world program derived from legislative specifications.

The first step of the development was to gather all the legislative and regulatory sources describing the computation of the French housing benefits. This was done by B in approximately 20 hours of work, resulting in a 106-pages document compiling the (almost, see §3) complete specification.

Then, the bulk of the programming was performed in 22 sessions of pair programming totalling 50 hours. In each session, A or C was paired with B. During these sessions, intense cross-disciplinary discussion was necessary to build a shared knowledge of the effects of the law and the Catala program. But we can say in retrospect that the majority of the time was spent on actually deciphering and understanding the text of the law, as the drafting style is very compact and sometimes arcane. The Catala code produced amounts to about 7,000 lines of code. The ratio of lines of code per hour of programming is very high because of several factors. First, the code contains a lot of copy-pasting that mirrors the duplications in the law. Second, a lot of boilerplate code comes from the translation of huge tables of values, contained in an executive order assigning threshold amounts of benefits. The logic of the table had to be expressed in terms of conditionals and pattern matching in a very repetitive fashion. Third, programmer A being the very proficient the language, this scenario is one of maximum programming productivity; the project time is likely to go up with another programmer less accustomed with the language.

Next, approximately 50 hours were spent testing and debugging the resulting program. Some tests were crafted manually by B who invented fictional households and manually computed the housing

benefits amounts by applying the rules in the legislative sources. Other test cases included expected outputs were collected from the official explainer document for the housing benefits computation from the government. These official test cases yielded some differences with our computations, so we investigated with the authors of the official government document, the DGALN/DHUP/FE4 administrative office in the Ministry responsible for Housing. These investigations around the test cases revealed a number of bugs in our implementation, as well as some factual errors in the official test cases that were consequently fixed. The total number of test cases involved in the qualification of the program is quite low, around 20. Indeed, crafting a new test case is very time consuming due to the complexity of the input space and the tedious nature of the housing benefits computations that involve many steps when done by hand. We do not believe the current level of testing to be sufficient and a doubling or tripling of the testing time would be absolutely necessary to reach a minimum level of assurance.

Additional software validation was performed using the initial features of the Catala proof platform [Delaët et al. 2022], including the formal verification of the well-behaved execution of the program (there is always one rule that apply, no two exceptions overlap). Even though the proof platform is not fully implemented, with only a Z3 backend and a crude encoding that produces a lot of false positives, it found a bug. Finally, we developed a tool that scans our codebase and compares the legislative text with the official version stored on the Légifrance database. This tool helped us locate and fix a few copy-pasting errors (especially in tables) that were mirrored in the Catala code.

3 LEGAL CHALLENGES

As reported by [Merigoux et al. 2023], we found a number of factual errors in the text of the executive orders describing the computation parameters for the housing benefits. The errors included two typos in numbers contained in a table of parameters directly determining the maximum amount of benefits entitled in a corner case (“59.47” instead of “459.47” and “47.99” instead of “347.99”), and more problematically a whole table of income brackets was missing for a mode of computation (although it was published in the official, non-legal explainer document from the government). Those factual errors have been fixed by the government with a new executive order amending the previous one.

Another legal challenge we encountered was the lack of clarity in the drafting style of executive orders. Indeed, as reported by the the DGALN/DHUP/FE4 administrative office, the executive orders describing the French housing benefits computation had undergone in 2019 a massive restructuring and reform, that led to a coexisting of old and new texts in the current version. The articulation between the old and new texts had not yet been perfectly consolidated at the time of our development, and we sent a list of a dozen questions to the DGALN/DHUP/FE4 office to clarify the interpretations. DGALN/DHUP/FE4 answered promptly and later requested our input for a coming “grooming” rewrite of these texts, which has not yet been enacted at submission time.

The computer replication of the French housing benefits computation, conducted with a formal and detailed-oriented methodology, led to two interesting findings at the intersection between law and computation. First, as detailed by Merigoux [2022], a computational trick in the final steps of the computation only works if the rate of the CRDS tax is below 1%. If the rate is ever changed to a value superior to 1% (currently it is 0.5%), the formula written in the executive order will result in rounding errors potentially detrimental to the beneficiaries. This finding stresses the importance of making computational invariants explicit, even in legislative texts. Second, the computation distinguishes between two exceptional cases for the housing benefits amount: the case where the housing is a room within a larger housing unit, and the case where the housing is

shared with flatmates. However, ambiguity prevails when the two situations happen at the same time. Confronted with that ambiguity, DGALN/DHUP/FE4 answered: "As this situation is far from reality, it has not been dealt with". This reminds us of a delayed form of decision that clashes with the nature of computer programs [Diver 2021; Hildebrandt 2020]. To be respectful of that delayed decision, the best option could be for the program to crash and instruct a human review of the case.

The last but not least of the legal challenges was the computation of the housing benefits in the case of split custody. Merigoux [2022] provides an in-depth explanation of the thorny situation on this issue: pressed by past court decisions to split the benefits between the two households of the split custody child, the administrative agencies delay the application of this measure because of a technical blocking point. This technical blocking point is in fact related to the capacity of the IT system to compute the benefits twice for the same household, with or without the children in split custody. The ability to call the computation twice requires proper computer programming abstractions, like functions or classes, that have not yet been identified as a key requirement for legal expert systems (see §4). Our Catala replication has proven flexible enough to express the exact computation of the benefits split, therefore providing a technical path to resolving the stillstand that has been ongoing since 2017. Since then, only households that file an official complaint to the administration have their benefits split by hand, the others cannot have their split custody child accounted for properly.

4 LANGUAGE DESIGN CHALLENGES

In this section, we will discuss the language design decisions of Catala with respect to the empirical needs of the French housing benefits computation.

Defeasability. The need for a form of defeasible logic when formalizing legislation has been identified for a long time, as summarized by [McCarty 1997]. Catala embarks a very restricted form of defeasible logic in its default calculus [Merigoux et al. 2021a], corresponding to a subset of prioritized default logic [Brewka and Eiter 2000]. We found that this restricted form of defeasible logic was sufficient to correctly express all the exceptional case structures found in the French housing benefits computation. The most complex exceptional case structure involved multiple groups of rules, each group piecewise defining the value of a variable, being arranged in a tree-like structure where a group of rules can have multiple groups of rules as exceptions to it, and the exceptions can also recursively have exceptions. We found the lean semantics and runtime for dealing with exceptions in Catala to be sufficient for our needs.

Precision. The computation of the French housing benefits involves amounts of money being multiplied and divided multiple times. Particularly, the computation of the "rent equivalence" involves up to 6 income brackets and a division by twelve that is sufficiently complex to raise the usual problems of computing precision stemming from the binary representation of numbers in the machine. Learning from the mistakes of the French tax authority that chose floating-point numbers to represent all variables, including amounts of money [Merigoux et al. 2021b], Catala features a rich typology of values where amounts of money, integers and decimal numbers each have different representations. In particular, the current Catala semantics represents money as an integer number of cents; multiplying a money by a decimal yields a new amount of money rounded to the nearest cent. Hence, in an expression like $(m \times a) \times b$ where m is an amount of money and a and b two decimals, two roundings to the nearest cent happen when computing with Catala. In the detailing of the official tests cases for French housing benefits as published by DGALN/DHUP/FE4, all intermediate amounts of money are also rounded to the nearest cent. But when confronted with the question of computing precision, DGALN/DHUP/FE4 later clarified that the rounding to

the nearest cent was merely an "explanation artefact", and that large formulae in the computation should only round at the end and not in intermediate values. We had to change our Catala code (switching from the money type to decimal) to account for that behavior, that could change by up to 1 euro the amount of benefits effectively received. More generally, the question of computing precision for taxes and social benefits computation seems to be largely under-considered by both lawyers and programmers, compared to the potentially massive cumulated effects it can have.

Modules and functions. The computation of the French housing benefits, with 7,000 lines of code, is big enough to trigger some software engineering challenges. First, we felt the need for modules representing separate compilation units to better structure our code. This limitation is being addressed with a planned module system for Catala. Second, there are six different modes of computation with four main formulae that can interact or reference each other. Because of these interactions, being able to encapsulate these modes of computation and formulae into functions of our language was paramount. To that end, Catala features the concept of "scope", that corresponds to a function with some defeasible capabilities builtin. The current Catala syntax only allows a static declaration of the scope call graph, which proved to be a limitation since we want certain sub-scopes to be called only if a condition is filled. The addition of a reified scope call syntax that lifts this limitation is underway. The functional nature of Catala with immutable values lead to a certain amount of boilerplate "plumbing" code to fill the inputs of sub-scopes with values that are themselves inputs of the parent scope. However, this explicitation of the input values of the scope has proved comfortable when dealing with subtle counter-factuals in the law that mandate a certain computation to be made again with slightly different arguments. The usefulness of functions for formalizing legislation is somewhat under-represented in the literature, and our findings here are consistent with previous studies on the French tax code implementation [Merigoux et al. 2021b].

Higher-order. Also somewhat under-represented in the literature, the usefulness of higher-order functions has been empirically validated in the French housing benefits computation. Used in many places, higher-order functions felt natural to the lawyer that reviewed the code. In Catala, functions only have one argument so we don't run into partial application or curryfication problems. Even though a higher-order function usually can be replaced by a refactoring, they were critical in a high-stakes situation. For splitting benefits in the case of split custody, what has to be split is a "gross" amount of benefits, on which several post-processing steps have to be applied to reach the final amount. These steps depend on the mode of computation used (6 in total), but we didn't want to have 6 copies of the splitting operation that is inserted before the post-processing steps. By reifying the post-processing steps with a higher-order function, we were able to avoid code duplication and correctly express the computation.

5 CONCLUSION

This case study on the French housing benefits has been conducted not only with the objective of producing a piece of software, but also to interrogate the ways in which this particular piece of law was written and enforced by the government. The numerous findings it yielded can inform both lawyers and computer scientists working at the intersection of their fields.

This replication provides a starting point for a potential replacement as the production system for computing the French housing benefits. Even though the choice of replacing a production system is consequential and involves a lot of parameters, we believe that our replication and the underlying Catala technology have the potential of being production-ready for government systems within a few years.

ACKNOWLEDGMENTS

Many thanks to Lilya Slimani who has been my pair programming partner for the development of the French housing benefits codebase, and to Alain Delaët for the occasional help. Thanks to the DGALN/DHUP/FE4 administrative office for their very reactive feedback and cooperation, which helped us debug our program and somewhat improve the redaction of the executive orders describing the computation of the French housing benefits.

REFERENCES

Layman E Allen. 1956. Symbolic logic: A razor-edged tool for drafting and interpreting legal documents. *Yale LJ* 66 (1956), 833.

Trevor JM Bench-Capon, Gwen O Robinson, Tom W Routen, and Marek J Sergot. 1987. Logic programming for large scale applications in law: A formalisation of supplementary benefit legislation. In *Proceedings of the 1st international conference on Artificial intelligence and law*. 190–198.

Xavier Berne. 2018. Les Allocations familiales nous ouvrent le code source de leur calculateur d'aides. *Nextimpact* (2018). https://www.nextinpact.com/article/28136/106298-les-allocations-familales-nous-ouvrent-code-source-leur-calculateur-daides

Gerhard Brewka and Thomas Eiter. 2000. Prioritizing default logic. In *Intellectics and computational logic*. Springer, 27–45.

Alain Delaët, Denis Merigoux, and Aymeric Fromherz. 2022. Turning Catala into a Proof Platform for the Law. In *POPL 2022 - Programming Languages and the Law*. Philadelphia, United States. https://hal.inria.fr/hal-03447072

John Dewey. 1924. Logical method and law. *Cornell LQ* 10 (1924), 17.

Laurence Diver. 2021. Interpreting the Rule(s) of Code: Performance, Performativity, and Production. *MIT Computational Law Report* (15 7 2021). https://law.mit.edu/pub/interpretingtherulesofcode https://law.mit.edu/pub/interpretingtherulesofcode.

Mireille Hildebrandt. 2020. Code-driven Law: Freezing the Future and Scaling the Past. In *Is Law Computable? : Critical Perspectives on Law and Artificial Intelligence*, Simon Deakin and Christopher Markou (Eds.). Hart Publishing, Oxford, 67–84.

Liane Huttner and Denis Merigoux. 2022. Catala: Moving Towards the Future of Legal Expert Systems. *Artificial Intelligence and Law* (Aug. 2022). https://doi.org/10.1007/s10506-022-09328-5

Rachel Knaebel. 2022. « Une galère pas possible » : quand la Caf refuse de prendre en compte la résidence alternée. *Basta!* (2022). https://basta.media/RSA-APL-temoignage-une-galere-pas-possible-quand-la-caf-refuse-de-prendre-en-compte-la-residence-alternee

L. Thorne McCarty. 1977. Reflections on "Taxman": An Experiment in Artificial Intelligence and Legal Reasoning. *Harvard Law Review* 90, 5 (1977), 837–893. http://www.jstor.org/stable/1340132

L. T. McCarty. 1989. A Language for Legal Discourse I. Basic Features. In *Proceedings of the 2nd International Conference on Artificial Intelligence and Law* (Vancouver, British Columbia, Canada) *(ICAIL '89)*. Association for Computing Machinery, New York, NY, USA, 180–189. https://doi.org/10.1145/74014.74037

L Thorne McCarty. 1997. Some arguments about legal arguments. In *Proceedings of the 6th international conference on artificial intelligence and law*. 215–224.

Denis Merigoux. 2022. *Observations sur le calcul des aides au logement.* Research Report RR-9485. Inria Paris. 27 pages. https://hal.inria.fr/hal-03781578

Denis Merigoux, Marie Alauzen, and Lilya Slimani. 2023. Rules, Computation and Politics: Scrutinizing Unnoticed Programming Choices in French Housing Benefits. *Journal of Cross-disciplinary Research in Computational Law* (2023). https://hal.inria.fr/hal-03712130 (forthcoming).

Denis Merigoux, Nicolas Chataing, and Jonathan Protzenko. 2021a. Catala: A Programming Language for the Law. *Proc. ACM Program. Lang.* 5, ICFP, Article 77 (Aug. 2021), 29 pages. https://doi.org/10.1145/3473582

Denis Merigoux, Raphaël Monat, and Jonathan Protzenko. 2021b. A Modern Compiler for the French Tax Code. In *Proceedings of the 30th ACM SIGPLAN International Conference on Compiler Construction* (Virtual, Republic of Korea) *(CC 2021)*. Association for Computing Machinery, New York, NY, USA, 71–82. https://doi.org/10.1145/3446804.3446850

Nicolas Monnet. 2022. "Qui ne demande rien n'a rien" : de mauvaise volonté, la CAF vous prive peut-être injustement de ces prestations. *Midi Libre* (2022). https://www.midilibre.fr/2022/06/11/qui-ne-demande-rien-na-rien-de-mauvaise-volonte-la-caf-vous-prive-peut-etre-injustement-de-ces-prestations-10353023.php

Anja Oskamp and Marc Lauritsen. 2002. AI in Law Practice? So far, not much. *Artificial Intelligence and Law* 10, 4 (01 Dec 2002), 227–236. https://doi.org/10.1023/A:1025402013007

M. J. Sergot, F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammond, and H. T. Cory. 1986. The British Nationality Act As a Logic Program. *Commun. ACM* 29, 5 (May 1986), 370–386.

Faïza Zerouala. 2021. La réforme des APL vire au cauchemar pour les allocataires et ses agents. *Mediapart* (2021). https://www.mediapart.fr/journal/france/190621/la-reforme-des-apl-vire-au-cauchemar-pour-les-allocataires-et-ses-agents