Turning Catala into a Proof Platform for the Law
Alain Delaët, Denis Merigoux, Aymeric Fromherz

To cite this version:
Alain Delaët, Denis Merigoux, Aymeric Fromherz. Turning Catala into a Proof Platform for the Law. POPL 2022 - Programming Languages and the Law, Jan 2022, Philadelphia, United States. hal-03447072

HAL Id: hal-03447072
https://inria.hal.science/hal-03447072
Submitted on 24 Nov 2021

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.
The Catala Toolchain

Legal statutes contain descriptions of algorithmic decision processes. In that regard, certain domains of the law are thus more “algorithmic” than others. For instance, tax law describe the computation of the amount of taxes owed by an individual, depending on her income and family situation. More interestingly, these legally-defined algorithms are usually enforced by computer programs called legal expert systems, that government agencies or large organizations have been using for decades. Merigoux et al. [2021] cast light on the issues related to the maintenance and production of these legal expert systems, and their faithfulness with respect to the corresponding legislative specification. To tackle those issues, a novel domain-specific language, Catala, was described and formalized in the same paper. By design, Catala programs closely follow the logical structure of legal statutes. Thus, Catala enables a pair programming and literate programming methodology that raises the level of assurance of legal expert systems, while providing a usable toolchain amenable for production deployments.

Under the hood, Catala relies on an intermediate representation, called default calculus, which is a λ-calculus augmented with default logic terms. Default logic was introduced by Reiter [1987] and is known to be well-suited for legal reasoning, as shown by Lawsky [2017, 2018]: legal statutes define quantities and computations using a base case/exceptions structure often scattered across multiple articles of law. For instance, in the US Internal Revenue Code, sections 61 to 68 define what is considered as income for an individual in general terms, then sections 71 to 291 refine this definition with exemptions, tax cuts and specific inclusions. Encoding such a structure using only conditionals is uneasy, which is why we introduce a default logic term. Formally, this default term is represented as an expression \( \langle e_1, \ldots, e_n | e_{\text{just}} : - e_{\text{cons}} \rangle \) where \( e_{\text{just}} : - e_{\text{cons}} \) is the base case and \( e_1, \ldots, e_n \) are the exceptions. The semantics of the default term is as follows: first, each of the exceptions \( e_i \) is evaluated; if two or more are valid, a conflict error \( \ast \) is raised. If exactly one exception \( e_i \) is valid, the final result is \( e_i \). If no exception is valid, we evaluate the base case: if \( e_{\text{just}} \) evaluates to \( \text{true} \) then the result is the default consequence \( e_{\text{cons}} \), otherwise the result is \( \emptyset \). Using this formalism, we can encode sections 61 to 68 as a base case and sections 71 to 291 as exceptions.

A more in-depth description of Catala, including a full formalization of the language and a part of the section 121 of the US Internal Revenue Code, can be found in the original paper of Merigoux et al. [2021].

Properties of Catala Programs

In this paper, our goal is to verify properties about Catala programs. These properties mostly are of two different kinds, which we present here.

Well-defined executions. We want to prove that the code generated by the Catala compiler is free of crashes. This includes traditional properties like zero-division errors, but is not limited to it. Indeed, the default-logic term introduces interesting new properties to check. First, we need to ensure the absence of conflict (\( \ast \)) errors, i.e., no two exceptions are valid inside the same default
term during evaluation. In legal terms, it corresponds to the unambiguous of law: no two legal exceptions can occur at the same time. Conversely, we must ensure that the law covers all cases, i.e., every default term must reduce to a value different from $\emptyset$.

Those properties are paramount for the use cases that Catala aims to tackle. For instance, we want the automatic enforcement of tax law to be deterministic and applicable to every possible profiles.

Higher-order properties. Additionally, we want to prove more complex and richer properties that reflect the intent of the legislator. For instance, in 2012, the French Supreme Court ruled against marginal tax rates greater than 70% [French Constitutional Council 2012]. The marginal tax rate can be defined as follows: if the income of a household increases by $\Delta I$, then its benefits will decrease by $\Delta B$ and its income tax will increase by $\Delta T$. Using these notations, the marginal tax rate is defined as $T_{\text{eff}} = \frac{\Delta T - \Delta B}{\Delta I}$. In this case, the property we want to guarantee is that, “for any household, $T_{\text{eff}} \leq 70\%$”.

In the next section, we propose to extend the Catala toolchain to model and verify such properties.

3 ARCHITECTURE OF THE VERIFICATION FRAMEWORK

Program simplification by refinement. From the two sorts of properties laid out in the previous section, the higher-level ones are of course more interesting. However, proving those properties on a pristine Catala program might be challenging as they require an analysis of the whole program. The complexity of the proof may arise from the corner cases of the default logic structures that obscure the functional meaning of the code, hence significantly raising program verification costs.

But interestingly, if we first prove that the execution of a Catala program is well-defined, then we can rule out most of these default logic corner cases, and perform semantics-preserving rewritings that simplify the program. For instance, by proving the absence of conflict errors in a default term $e$, we can remove the runtime mechanism that counts the number of exceptions triggered and simply use nested conditionals to express $e$. Thus, we propose a methodology for proving both the well-definedness of executions and higher-level properties on Catala programs. We summarize this methodology in Figure 1.

![Fig. 1. Overview of the proof by refinement architecture, transforming a program $P_0$ into a simplified program $P_n$, under verification conditions $c_1, \ldots, c_n$.](image)

Starting from a Catala program, we apply a series of program rewritings, which must be proved semantics-preserving. By transitivity, the resulting program will be equivalent to the original one, but without most of the complexity due to the default logic encoding. The objective is to simplify the program to enable the proof of higher-level properties inside proof assistants.

Syntax-directed automated simplification. In a sense, our methodology is similar to program verification efforts that use proof by refinement to phase out a problematic background semantics issue before hitting the functional core of the proof. Examples of this approach abound in the literature [Gu et al. 2016; Lorch et al. 2020], but so far each simplification step and intermediate
Turning Catala into a Proof Platform for the Law

3

program needs to be manually written down and proved. For Catala, we want to leverage the
domain-specific nature of the language to propose a heuristics-based, syntax-directed automated
procedure to identify potential simplifications in the program, directly embedded as a Catala
compiler pass.

Most heuristics will be focused on the default logic term and its corner cases, as stated earlier.
But additional domain-specific program simplifications could be detected and applied using the
syntax-directed matching. For instance, the law often defines a concept or quantity piecewise
depending on the current date, e.g., minimum salaries indexed on the yearly inflation. In Catala, the
piecewise definitions are stitched together in a single default term where each piece is an exception
coming with a triggering precondition comparing the current date to fixed interval bounds. Such a
pattern can be globally matched and transformed into a more idiomatic interval check, as long as
the sub-intervals form a partition of the input space.

Of course, all of these simplifications are contingent on verification conditions justifying their
semantic-preserving nature. Therefore, in addition to the syntax-directed simplification pass,
the Catala compiler will incorporate a system for collecting and distributing those verification
conditions to various proof backends. We sketch such a system in the next section.

4 CATALA AS A PROOF PLATFORM

As the syntax-directed simplification procedure described in the last section operates, it will generate
verification conditions backing the validity of each proposed simplification. Because the proposed
simplifications may depend on each other, these verification conditions will have to be dealt with
(proved or rejected) in a specific order, prompting the user for proof input. We aim at providing
connections from Catala to multiple proof backends: SMT solvers, symbolic execution engines,
and proof assistants. Following the methodology of Why3 [Filliâtre and Paskevich 2013], we want
to let the user control how each verification condition should be discharged. Hence, we might
design a tactics language that the programmer could use to give hints about the proof strategy for a
specific verification condition. For instance, a user could indicate that a specific variable should be
treated symbolically (instead of relying on its definition), or provide a cut proof goal and different
strategies for the two resulting verification conditions after the cut split.

By allowing the user to control the proof strategies, we aim to reach a tradeoff between a fully
automated proof approach that might fail often and leave the user powerless, and a completely
manual approach that would likely require a very high effort. We believe that our approach should
lead to dozens of verification conditions for simplifying a medium-sized Catala program, and that
most of these could be proven by an automated backend with little to no user guidance. This plan
might be altered by the data coming back from real-world case studies; overall, we plan to take a
pragmatic, application-oriented approach to this Catala proof platform.

REFERENCES

Jean-Christophe Filliâtre and Andrei Paskevich. 2013. Why3—where programs meet provers. In European symposium on
programming. Springer, 125–128.


Certikos: An extensible architecture for building certified concurrent {OS} kernels. In 12th USENIX Symposium on


Jacob R Lorch, Yixuan Chen, Manos Kapritsos, Bryan Parno, Shaz Qadeer, Upamanyu Sharma, James R Wilcox, and Xueyuan