Catala : formaliser la loi grâce à un langage de programmation dédié

Denis Merigoux [1] et Liane Huttner [2]

Ce texte est adapté d'une présentation orale donnée au séminaire Codes Sources du 10 mars 2022.

Introduction : programmer la loi ?

Transformer les textes de loi en programmes informatiques est une tâche cruciale dans l'administration publique, nécessaire à l'application massive du droit pour produire les millions de décisions individuelles que constituent le prélèvement des impôts ou la distribution des aides sociales. C'est cette tâche de transformation à laquelle nous nous intéressons du point de vue des méthodes formelles. Prenons tout de suite un exemple pour cadrer notre sujet d'étude :

> **Code Général des Impôts, Article 197, I, 3, b, 3° (circa 2019)**
>
> Le taux de la réduction prévue au premier alinéa du présent b est de 20 %. Toutefois, pour [...], le taux de la réduction d'impôt est égal à 20 % multiplié par le rapport entre :
>
> — au numérateur, la différence entre 20 500 €, pour [...], ou 41 000 €, pour [...], et le montant des revenus mentionnés au troisième alinéa du présent b, etc. ;
>
> — au dénominateur, 2 000 €, pour [...], ou 4 000 €, pour [...].

1. Inria Paris, denis.merigoux@inria.fr.
2. Université Panthéon-Sorbonne, liane.huttner@univ-paris1.fr.

DOI:10.48556/SIF.1024.20.77 **1024** – Bulletin de la SiF, numéro 20, Novembre 2022, pp. 77–84

Le texte ci-dessus est constitué d'extraits du code général des impôts (CGI), qui regroupe tous les textes législatifs et réglementaires qui décrivent entre autres la formule de calcul de l'impôt sur le revenu à partir de la déclaration de revenus. On peut remarquer que cet article de loi est assez différent de l'idée reçue que l'on peut se faire de ce genre de texte. Plus précisément, cet article présente une étape d'une décision se faisant sans intervention humaine, sans ambiguïté (apparente), et traitant des données quantitatives. En résumé, cet extrait du CGI décrit une étape d'un algorithme. D'ailleurs, s'il fallait traduire le texte ci-dessus en pseudo-code, cela donnerait un programme pas si trivial que cela :

```
TAUX = si (non [...(1)...]) alors 20 % sinon
  20 % * (
    ((si [...(2),(3)...] alors 20 500 € sinon 41000 €)
      - REVENUS) /
    (si [...(4),(5)...] alors 2000 € sinon 4000 €))
```

Ainsi, la loi fait parfois office de spécification algorithmique. Attention, toutes les lois ne sont pas des algorithmes ! Il est évidemment hors de question de programmer des notions fortement dépendantes au contexte et nécessitant le jugement humain comme la justice ou l'égalité. Mais certains domaines juridiques sont particulièrement adaptés à une interprétation algorithmique, comme le calcul de l'impôt ou des prestations sociales. Concrètement, voici un petit tour d'horizon des programmes informatiques en production dans les administrations, et qui appliquent des algorithmes spécifiés dans la loi :

Nom	Langage	Taille (lignes)
Impôt sur le revenu	M et C	100 000
Impôt sur les sociétés	Java	10 000
Cotisations sociales	SQL	20 000
Prestations sociales	COBOL	6 900 000
Allocation chômage	Java	1 300 000
Retraites	?	?
Droits de succession	?	?

Outre la taille tout à fait conséquente des programmes, on peut également remarquer que ceux-ci sont souvent écrits dans des langages de programmation *legacy* comme COBOL, ou inhabituels (20 000 lignes de SQL !) ou encore spécifiques à l'administration comme le langage M de la DGFiP [9]. Enfin, certains programmes sont toujours secrets, malgré l'obligation qui est faite aux administrations de publier leur code source depuis la loi pour une République numérique de 2016.

Pour autant, même publiés, ces programmes informatiques ne sont pas forcément lisibles et compréhensibles. Voici par exemple un extrait (lignes 347 à 363) du code de la DGFiP qui a calculé l'impôt sur les revenus 2020 [3] :

3. Publié sur `https://gitlab.adullact.net/dgfip/ir-calcul`, fichier `chap-cmajo.m`.

```
RDSTARDIF_DEF = max(0,
  FLAG_RETARD * (FLAG_TRTARDIF * RDBASE_MAJO
    + FLAG_TRTARDIF_R * SUPRD[00] + FLAG_TRTARDIF_F *
      (positif(TARDIFEVT2) * positif(RDSTARDIF_A +
        CODCOR-CODCOR_A - RDBASE_MAJO) *
      (positif(FLAG_RECTIF) * min(SUPRD[2],RDBASE_MAJO)
        + (1 - positif(FLAG_RECTIF)) *
          min(max(0,RDBASE_MAJO-SUPRD[00] *
            (1 - positif(FLAG_RETARD0718))),
              RDSBASE_REF)) +
      (1 - positif(TARDIFEVT2) * positif(RDSTARDIF_A
        + CODCOR-CODCOR_A- RDBASE_MAJO)) *
      (positif(FLAG_RECTIF) * min(SUPRD[00],RDBASE_MAJO) +
        (1 - positif(FLAG_RECTIF)) * RDSBASE_REF)) +
    (1 - positif(FLAG_TRTARDIF + FLAG_TRTARDIF_R +
      FLAG_TRTARDIF_F + FLAG_TRMAJOP)) *
    (positif(FLAG_RECTIF) * SUPRD[00] +
      (1 - positif(FLAG_RECTIF)) *
        (RDSTARDIF_A +CODCOR-CODCOR_A))));
```

Comment retrouver quel article de loi cette expression est censée modéliser ? Mission impossible si l'on est pas dans la tête de celui ou celle qui l'a écrit...

La situation française n'est pas une exception ; les problèmes de ce genre sont même parfois pires ailleurs. Aux États-Unis par exemple, le calcul de l'impôt sur le revenu repose toujours sur un programme écrit dans l'assembleur spécifique à un ordinateur des années 1960 [13] ! Par contre, il existe en France un exemple connu d'échec industriel en partie lié à la difficulté de traduire la loi en code informatique. Il s'agit du système de paie Louvois de l'armée, renommé en SourceSolde. Selon la presse [10, 12], l'État a profité de l'informatisation du calcul de la paie pour supprimer 750 postes au ministère des armées. Cependant, avec plus de 174 primes et indemnités différentes, impossible de définir des critères simples de calcul. S'en sont suivis des bugs catastrophiques avec des soldats et leurs familles privés de solde pendant plusieurs mois. Selon le rapport d'une mission d'information de l'Assemblée Nationale [1], « *pour certains, les défaillances du calculateur lui-même constituent le problème majeur à la source des dysfonctionnements de l'écosystème Louvois. Le chef d'état-major des armées, devant la commission, a ainsi déclaré : j'en viens à penser que c'est le cœur du système Louvois qui constitue le fond du problème : vraisemblablement, il a été mal, ou insuffisamment, spécifié* ». Le problème de la spécification de ces logiciels devant appliquer des textes de loi ou des règlements est donc cruciale. Mais alors, comment vérifier que ces spécifications, et le logiciel lui-même, sont fidèles à la loi ?

Des programmes fidèles à la loi ?

Une première solution évidente est de valider la spécification et le logiciel à l'aide de tests, écrits à la main par des juristes spécialistes du droit en question. Cependant, plusieurs raisons structurelles rendent très coûteux le maintien d'une telle base de tests. D'abord, l'espace des entrées des programmes basés sur la loi souffre souvent

d'explosion combinatoire. En effet, avec plus de 1500 variables d'entrées dans la déclaration de revenus, il faudrait un nombre astronomique de tests pour couvrir toutes les possibilités (certainement plus que de foyers fiscaux en réalité !). Ensuite, les juristes n'ont pas l'habitude d'écrire des tests et il est difficile de leur inculquer la notion de couverture de code ; dans les études de droit, on apprend à appliquer la loi, pas à inventer à l'avance des situations insolites pour en tester les recoins sombres, puisque ces situations sont produites gratuitement par la réalité. Enfin, la loi change régulièrement ; tous les ans ou plus pour la loi fiscale par exemple. Il faut donc remettre à jour toute la base de tests très régulièrement.

Pour toutes ces raisons, la traduction de la loi en code nécessite une méthode de validation complémentaire à celle des tests qui est trop coûteuse pour atteindre un niveau maximum d'assurance sur le logiciel.

Mais quelle méthode utiliser ? Naïvement, on pourrait se dire que le plus simple serait que les spécifications et le logiciel soient directement écrits par les experts du domaine, c'est-à-dire les juristes. Mais comment rendre cela possible en pratique ? Il semble difficile de demander à des juristes d'écrire des logiciels de plusieurs dizaines ou centaines de milliers de lignes de code. Au lieu de cela, est-il possible pour les juristes de relire le code source de ces logiciels afin de détecter des erreurs d'interprétation de la loi ? La réponse à cette question est compliquée car plusieurs obstacles se placent sur la route. Premièrement, un juriste peut-il lire du code informatique ? Le droit fait appel au raisonnement logique et on ne voit pas pourquoi un juriste ne pourrait pas comprendre ce qu'est une conditionnelle, ou même une fonction. Empiriquement, nos discussions interdisciplinaires ont fait ressortir que le problème n'était pas là, car fondamentalement, le raisonnement logique basique qui sous-tend un programme informatique est tout à fait accessible aux juristes.

Par contre, un juriste peut-il lire la syntaxe d'un langage de programmation ? Cela dépend du langage de programmation, et la syntaxe est effectivement une barrière à l'entrée qui rebute nombre de non-informaticiens. Cependant cet obstacle peut être levé soit avec une formation, soit par la conception de syntaxes adaptées et proches du langage naturel [3].

Finalement, le plus sérieux obstacle à ce qu'un juriste puisse relire du code informatique concerne l'organisation du code. En effet, le juriste se trouve dans la loi grâce aux numéros d'articles, d'alinéas, à la structuration en différents titres et chapitres des codes législatifs et réglementaires. Or, le code informatique possède sa propre logique d'organisation par unité computationnelle qui est souvent radicalement différente de celle du texte de loi. La correspondance entre article de loi et bout de code informatique qui le traduit est anarchique : parfois, un article de loi est traduit par une incise à dix endroits différents dans le code ; parfois, c'est l'inverse et un bout de code modélise en même temps dix articles de loi. Ainsi, quand un juriste relit du code informatique, il est très souvent perdu car il ne retrouve plus la structure familière du texte de loi qu'il est censé appliquer. Pour remédier à ce problème, les

administrations ont la plupart du temps recours à la rédaction d'un document intermédiaire entre le texte de loi et le code informatique, parfois appelé *spécification*. Il s'agit d'un texte écrit en français naturel, décrivant l'effet computationnel d'un texte de loi, et organisé *comme un programme*. Sauf que, quand les juristes *analystes* chargés de rédiger cette spécification sont peu familiers avec la programmation, cette spécification n'est souvent ni assez détaillée ni assez bien organisée pour être traduite telle quelle en code informatique...

Mais pourquoi donc est-il si difficile de faire coïncider la structure du code informatique et celle du texte de loi ? Il se trouve qu'il existe une discordance profonde, et de nature logique. Sarah Lawsky [4], rappelant récemment des découvertes de la communauté de la programmation logique dans les années 1980 [5], met en évidence la dichotomie entre cas de base et exceptions dans les textes du droit fiscal américain. En effet, la plupart des textes de loi énoncent d'abord un principe général (cas de base), puis une multitude d'exceptions venant nuancer l'application de ce principe suivant les situations. En reprenant l'exemple de l'article 197 du CGI, on peut surligner en gras des conséquences introduites par deux justifications en italique, une pour le cas général et l'autre pour une exception :

> *Le taux de la réduction prévue au premier alinéa du présent b* **est de 20 %**. *Toutefois, pour les contribuables dont les revenus du foyer fiscal, au sens du 1° du IV de l'article 1417, excèdent 19 176 €, pour la première part de quotient familial des personnes célibataires, veuves ou divorcées, ou 38 352 €, pour les deux premières parts de quotient familial des personnes soumises à une imposition commune, ces seuils étant majorés le cas échéant dans les conditions prévues au même premier alinéa,* **le taux de la réduction d'impôt est égal à 20 % multiplié par le rapport entre : [...]**

Computationellement, cette structure en cas de base et exception est difficile à représenter car avant d'évaluer le cas de base, il faut vérifier que l'on ne déclenche aucune exception. En informatique, le cas de base se trouve à la fin de la conditionnelle dans le `else` alors qu'il se trouve énoncé en premier dans le texte de loi. De plus, des structures logiques usuelles du droit plus compliquées comme des exceptions d'exceptions deviennent compliquées à exprimer avec des conditionnelles dans le code informatique. Si l'on veut que la structure du code informatique suive celle du texte de loi, il est alors impératif d'utiliser un langage de programmation embarquant une fonctionnalité inspirée de la logique *par défaut*, une forme de logique non-monotone [11]. Or, presqu'aucun langage de programmation ne présente de construction permettant d'encoder naturellement une telle logique par défaut.

Catala, une méthodologie et un langage

Partant de ces observations et de l'état de l'art, nous proposons d'abord une méthodologie pour la production et la maintenance des programmes informatiques dérivant d'une spécification législative ou réglementaire. Le but de cette méthodologie est de garantir la correction du programme par rapport à la loi, ou plutôt par rapport à l'interprétation de la loi choisie par les auteurs du programme [6].

Le sujet central de la méthodologie que nous proposons est le cadre de production ou de modification du code. Plutôt que d'être fait isolément par un programmeur à partir d'un document tiers écrit par un juriste (cycle en V), nous préconisons d'utiliser deux techniques issues des méthodes agiles : la programmation en *binôme* et la programmation *littéraire*.

Dans le premier cas, l'objectif est de faire dialoguer un duo juriste-programmeur autour d'une même source dans laquelle les deux parties peuvent se retrouver. Concrètement, une séance de traduction de la loi en code se passe de la façon suivante : dans une salle, deux personnes sont réunies — un juriste et un programmeur. Le juriste a préparé la séance à l'avance en réunissant tous les textes de loi et règlements qui encadrent le dispositif cible ; par exemple le calcul d'une prestation sociale. Idéalement, le juriste a compilé ces textes en les copiant-collant dans un fichier texte que le programmeur va récupérer.

Ensuite, dans le cas de la programmation littéraire, le programmeur traduit linéairement, en suivant l'ordre du texte de loi, article par article. À chaque alinéa de chaque article, une discussion s'engage entre le programmeur et le juriste : qu'y a-t-il à calculer ici ? Est-ce que cet article décrit une étape d'un algorithme qui nous intéresse ? Si la réponse est non, le programmeur laisse un commentaire et passe à l'alinéa suivant. Si la réponse est oui, le programmeur écrit une première version du code informatique correspondant à cet alinéa, directement en dessous du texte de loi dans le même document. Une fois la première version écrite, voire pendant l'écriture, le juriste relit le code et intervient : qu'est-ce qu'on calcule ici ? Que veut dire cette instruction informatique ? Est-ce que le code gère cette interprétation de la loi ? Le programmeur répond et la discussion se poursuit : comment fait-on pour gérer ce cas-limite ? Faut-il rajouter un champ d'entrée pour prendre une décision dans le calcul à cet endroit là ? Quelles seront les conséquences de nos choix pour les utilisateurs ?

Toutes ces questions et réponses mènent à la prise de micro-décisions qui, dans leur ensemble, reflètent une certaine interprétation du droit qui ne dit généralement pas son nom [7]. Il est donc crucial, et c'est le dernier point de notre méthodologie, que le code source résultant du processus de traduction de la loi en code soit *open source* afin que tous ceux qui sont concernés par l'application de cette loi puissent aller constater les choix d'interprétation de l'administration, et si besoin les contester. En cela, nous ne faisons que recommander de préserver la pratique administrative

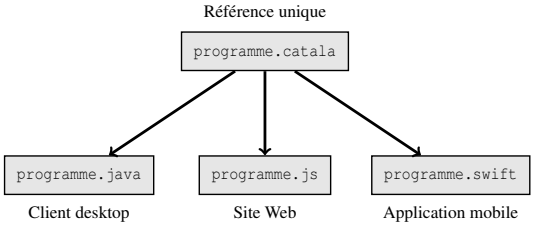
de publication des interprétations (par exemple au bulletin officiel des finances publiques).

Cette méthodologie est toutefois incomplète sans l'outil qui la rend possible. Il faut en effet disposer d'un langage de programmation qui permette de respecter la structure logique de la loi, qui soit compréhensible par le juriste, et qui dispose de suffisamment de capacité d'abstraction pour gérer des programmes de plusieurs centaines de milliers de lignes. C'est pour satisfaire ce cahier des charges que nous avons développé le nouveau langage de programmation spécialisé Catala [8, 2]. Nous ne décrirons pas les fonctionnalités du langage ni sa formalisation en détail ici, préférant vous renvoyer au papier technique sur le sujet [8]. Mais voici un petit exemple de traduction en Catala d'un article du code de la sécurité sociale décrivant une étape de calcul des allocations familiales :

> **Code de la sécurité sociale – Article L521-3** Toutefois, les personnes ayant un nombre déterminé d'enfants à charge bénéficient de ladite majoration pour chaque enfant à charge à partir de l'âge mentionné au premier alinéa.

```
champ d'application CalculAllocationsFamiliales :
  règle majorations_allocations_familiales.droits_ouverts
    de enfant
  sous condition
    (enfant dans ménage.enfants) et
    (nombre de ménage.enfants >= minimum_l521_3_alinéa_2) et
    (enfant.âge >= âge_limite_l521_3_alinéa_1 de enfant)
  conséquence rempli
```

Le code source Catala est ensuite compilé en passant par différentes représentations intermédiaires, qui vont jusqu'à des langages traditionnels en passant par du λ-calcul. L'idée est de disposer d'une référence unique pour toutes les implémentations de la loi dont auraient besoin les diverses applications informatiques d'un système d'information typique d'un grand ministère :



L'interopérabilité par livraison de code source directement dans le langage cible est plus contraignante qu'une interopérabilité par API, ou par *Foreign Function Interface* (FFI), mais c'est sans doute celle qui s'adapte le mieux à des environnements *legacy* où les technologies peuvent être très hétérogènes. De plus, la maîtrise de la chaîne de compilation permet de générer du code cible qui s'adapte parfaitement aux particularités de l'environnement où l'on veut le déployer.

Conclusion

Traduire la loi en code est une tâche ardue, méticuleuse mais néanmoins nécessaire et vitale pour le fonctionnement de nombre de services administratifs. Nous proposons une nouvelle méthodologie de production et de maintenance pour les programmes informatiques qui ont pour but d'implémenter une interprétation de la loi. Cette méthodologie s'appuie sur la programmation en binôme, la programmation littéraire, l'*open source* et un nouvel outil, le langage de programmation Catala [4].

Références

[1] Geneviève Gosselin-Fleury and Damien Meslot. Mission d'information sur la mise en œuvre et le suivi de la réorganisation du ministère de la Défense, 2013.

[2] Liane Huttner and Denis Merigoux. Catala : Moving Towards the Future of Legal Expert Systems. working paper or preprint, January 2022.

[3] Robert Kowalski. Logical english. *Proceedings of Logic and Practice of Programming (LPOP)*, 2020.

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

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

[6] Denis Merigoux. The Specification Problem of Legal Expert Systems. working paper or preprint, January 2022.

[7] Denis Merigoux, Marie Alauzen, and Lilya Slimani. Scrutinizing unnoticed programming choices in french housing benefits implementation. Submitted, 2022.

[8] Denis Merigoux, Nicolas Chataing, and Jonathan Protzenko. Catala : A programming language for the law. *Proc. ACM Program. Lang.*, 5(ICFP), August 2021.

[9] Denis Merigoux, Raphaël Monat, and Jonathan Protzenko. A modern compiler for the french tax code. https://doi.org/10.1145/3446804.3446850.

[10] Jacques Monin. Louvois, le logiciel qui a mis l'armée à terre, 2018.

[11] R. Reiter. Readings in nonmonotonic reasoning. chapter A Logic for Default Reasoning, pages 68–93. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[12] Philippe Reltien. Des milliers de soldats français endettés à cause de louvois, leur logiciel de paie. *France culture*.

[13] Tom Temin. IRS programming mystery continues. *Federal News Network*, 2020.

4. https://github.com/Catalalang/Catala.