Logique et linguistique de l'informatisation du droit

Baptiste Mélès

CNRS, Archives Henri-Poincaré, Université de Lorraine

École thématique CNRS
« Mathématiques et philosophie contemporaines XII »,
Saint-Ferréol, jeudi 26 juin 2025.

Outline

Plan de la section

Plan de la section

Le code fait loi I

Lawrence Lessig, « Code is law », 2000 [Lessig()]

- « Le code fait loi »
- À l'âge du cyberespace, les décisions techniques ne sont pas politiquement neutres
  - par exemple TCP/IP ne facilite pas la régulation des activités sur le web, ce qui favorise la liberté d'expression mais permet aussi des activités illicites ou nuisibles
- « Dans toutes ces circonstances, les architectures viendront garantir nos valeurs traditionnelles – ou pas. [. . .] Les choix concernant le code et le droit sont des choix de valeurs. »

Le code fait loi II

- « Nous vivons à une époque de scepticisme à l'égard de la démocratie. Notre époque est obsédée par la non-intervention. Laissons Internet se développer comme les codeurs l'entendent, voilà l'opinion générale. Laissons l'État en dehors de ça. »

Le code fait loi III

- Mais il est déjà trop tard :

  Ce n'est pas entre régulation et absence de régulation que nous avons à choisir. Le code régule. Il implémente – ou non – un certain nombre de valeurs. Il garantit certaines libertés, ou les empêche. Il protège la vie privée, ou promeut la surveillance. Des gens décident comment le code va se comporter. Des gens l'écrivent. La question n'est donc pas de savoir qui décidera de la manière dont le cyberespace est régulé : ce seront les codeurs. La seule question est de savoir si nous aurons collectivement un rôle dans leur choix – et donc dans la manière dont ces valeurs sont garanties – ou si nous laisserons aux codeurs le soin de choisir nos valeurs à notre place.

Le code fait loi IV

- « Nous devrions examiner l'architecture du cyberespace de la même manière que nous examinons le fonctionnement de nos institutions. »
- D'où la formule : « le code [informatique] fait loi » : les citoyens doivent avoir un contrôle sur les choix techniques qui décident de leur destin.

Plan de la section

La loi faite code I

Le constat dressé par Lessig sur le « cyberespace » vaut aussi pour certains domaines calculatoires du droit tels que l'impôt sur le revenu, les allocations familiales, les aides au logement, les retraites, etc. :

> *While lawyers, computer scientists, law professors, and technologistes argue over whether formalizing law by translating it into computer code is practical, effective, useful, or desirable, tax law has been formalized for years. [. . .] The question is not whether to formalize tax law but rather how to formalize it. Sarah Lawsky, « Coding the Code », 2022 [Lawsky(a), p. 536 et section II]*

La loi faite code II

Le malheur est que cette informatisation n'est pas fiable.

> *As Part II of this Article explained, the U.S. government has already formalized tax law. But existing formalizations often oversimplify or translate the law incorrectly, fail to track the actual underlying law, and obscure the formalizers' judgments and decisions, which reduces government accountability to taxpayers.*

Cahier des charges de l'informatisation du droit

- Que peut-on attendre de l'informatisation du droit ?
- Trois exigences naturelles :
  1. la publicité du code informatique : la possibilité pour chaque citoyen de le lire ;
  2. la fiabilité du code informatique : la certitude que le code corresponde au texte juridique ;
  3. la correction de l'interprète ou compilateur : la certitude que l'exécution du code se déroule sans bug.
- La principale difficulté est l'exigence 2, car il faut parvenir à mettre en correspondance le langage informel du code juridique avec le langage formel de la programmation : comment faire ?
  - La tâche préliminaire est l'analyse du droit, qui passe par une étude linguistique et logique.

Plan de la section

Plan de la section

Entre philosophie et sociologie

- Jean Ray, *Essai sur la structure logique du Code Civil français*, Paris, Alcan, 1926. [Ray()]
- Entre philosophie et sociologie :
  - dédicace « À la mémoire de mes Maîtres Émile Durkheim et Octave Hamelin » ;
  - publication dans la collection fondée par Durkheim, « Travaux de l'Année sociologique ».

Une étude logique et sociologique

Étude du Code civil de 1804 :

- ce n'est pas une étude génétique : ces considérations sont rejetées en annexe (cf. Annexe I) ;
- étude du Code Civil comme un fait social (Durkheim), qui se trouve être aussi un système (Hamelin).

Méthode logique et sociologique

Cette étude est essentiellement logique et linguistique : c'est par la linguistique que la logique devient observable (p. ix).

> Nous nous proposons d'étudier le Code civil comme une donnée : il constitue en même temps un fait intellectuel et un fait social. Ce double caractère répond aux deux tendances qui ont guidé les précurseurs : la tendance rationaliste représentée surtout par Domat, la tendance historique et sociologique représentée par Montesquieu. — Entre ces deux influences il n'y a pas l'opposition que l'on croit, et que souligna par exemple la controverse de Thibaut et de Savigny : la codification, ou plus largement la rationalisation du droit est elle-même une donnée de l'histoire, un fait social. Nous étudierons donc notre document d'un point de vue à la fois logique et sociologique. — C'est une analyse logique du Code qui nous fera pénétrer dans le mécanisme de cette forme intéressante de la pensée pratique.

Plan de la section

Affects

Ray montre les signes montrant l'enracinement du Code Civil dans une certaine société : ses affects (I, 3)...

CHAPITRE III

Une survivance caractéristique : des formules qui traduisent une attitude affective du législateur 60

On trouve dans le Code civil certaines propositions, qui sont fréquentes dans des droits moins évolués, et qui expriment par exemple la réprobation sociale. Mais en général ces marques affectives sont remplacées par des qualifications qui tirent leur force de leur complexité intellectuelle.

Conception de l'homme

sa conception de l'homme (II, 1)...

DEUXIÈME PARTIE

Les éléments de la construction

CHAPITRE PREMIER

De la représentation de l'homme et du monde dans le Code civil ; sa pauvreté 117

a) *La nature et les choses* 118
b) L' « *homo juridicus* » 120

La pauvreté de la représentation de l'homme et du monde vient en partie de son caractère traditionnel ; mais à son tour elle permet à la tradition de se maintenir : les représentations désuètes sont sauvées par leur insignifiance. Il ne faut pas exagérer cette permanence : le dernier siècle a vu naître bien des réglementations qui supposent une représentation de la vie moderne. Seulement le droit civil, qui reste le droit commun, demeure sous la dépendance d'une représentation schématique de la nature et de l'homme.

Conception de la nature I

sa conception de la nature (II, 1)...

A. La nature et les choses

Il n'y a absolument rien dans le Code civil qui fasse songer à ce que pourrait être une représentation scientifique de la nature ou des phénomènes.

Çà et là le législateur se trouve amené à faire allusion à des faits ; il se contente d'en emprunter la représentation à l'opinion courante. C'est ainsi que l'art. 1 se réfère à une conception mi-géographique, mi-administrative du territoire national ; ces notions sommaires de géographie politique se retrouvent en divers textes (art. 3, 8, 9, 10, 11... ; 55, 59...).

La théorie de l'accession tire certaines conséquences juridiques de faits naturels : production des fruits de la terre, croît des animaux (art. 547, 583), formation d'alluvions (art. 556), relais (557), modification des rives ou du cours d'une rivière (art. 559 sqq.), migration spontanée de certains animaux (art. 564).

Conception de la nature II

Plus souvent que les choses dans leur état naturel, le droit a l'occasion de traiter des choses modifiées ou fabriquées par l'homme : un grand nombre de textes supposent certaines formes de vie : la vie agricole et sédentaire ; certains procédés de culture ; certaines formes d'habitat et d'aménagement. Plusieurs textes entrent même à cet égard dans un grand détail. (théorie des servitudes : art. 640 sqq. ; réparations locatives, art. 1754). Et l'on peut, sans hardiesse, affirmer que si de telles questions étaient soulevées aujourd'hui, les changements matériels apportés aux conditions de notre vie donneraient lieu à des réglementations que le Code civil ignore, par exemple en ce qui concerne l'éclairage, le chauffage ; on trouve dans certains textes du Code civil allemand une confirmation de cette idée[2].

Notions générales I

des notions générales (II, 3)...

CHAPITRE III

Du rôle des notions générales, éléments de l'activité créatrice aussi bien que de l'activité représentative 143

Nombre, temps, espace. Qualités, relations. Cause, fin, personne.

Conception du temps I

une conception du temps (II, 4)...

CHAPITRE IV

Du rôle de la notion de temps 146

Le temps est une des bases essentielles de la structure logique du Code.

SECTION I

Des événements 147

Les événements ne sont pas toujours des accidents. Il y a des événements liés au développement normal des institutions dans le temps. Il y en a d'autres qui sont extérieurs, fortuits : mais, même alors, leur considération est liée à la construction de la notion. Le normal, *quod plerumque fit*. Parfois le législateur réserve l'infinie variété des événements : les « circonstances de la cause ». Détermination des « cas » en vue de l'énonciation des règles.

Conception du temps I

SECTION II

Conditions et effets 158

Le rapport de subordination, dont les termes sont les conditions et les effets, apparaît comme l'articulation essentielle de la plupart des institutions. Entre conditions et effets existe une relation logique qui remplit, toutes réserves faites des différences, une fonction comparable à celle qui caractérise le rapport d'extension à compréhension. Mais cette relation ne se conçoit que par rapport au temps, à la causalité, à l'action organisée.

SECTION III

Périodes. Statuts 163

L'écoulement même du temps importe : la détermination des délais est un mode important de la pensée juridique ; la prescription est la forme la plus frappante de la puissance créatrice de l'écoulement même du temps. Les périodes que le droit discerne sont, par cela seul, colorées, qualifiées, ou servent de base à des qualifications. Le passé et l'avenir : la rétroactivité. Le statut, ensemble des aptitudes juridiques de telles personnes ou de telles choses.

Conception de la personne I

une conception de la personne (II, 6).

Chapitre VI

De la notion de personnalité 185

Dans le Code civil, le mot de « personne » s'applique à des individus humains ; mais le Code, si individualiste qu'il soit, et si suspectes que lui aient paru selon la double tradition monarchique et révolutionnaire certaines personnes morales, reconnaît pourtant des « sujets de droits et d'obligations » autres que les individus. Considérée comme n'exprimant que cette aptitude à être « sujet de droits et d'obligations », l'idée juridique de personnalité apparaît à la fois dans sa généralité abstraite et dans son ancienneté. Tout en reconnaissant qu'elle joue un rôle comparable à celui de l'idée psychologique de personnalité, on voit qu'elle en est indépendante : et il paraît tout naturel qu'il y ait des sujets de droits qui ne soient pas des individus, comme il peut y avoir des individus qui ne soient pas sujets de droits.

Conclusion

- Jean Ray montre ainsi dans quelle société s'enracine le Code Civil. Il relève systématiquement les traces par lesquelles le Code trahit une certaine compréhension de la société et du monde qui l'entourent.
- Il dégage en quelque sorte l'« image inverse » de la société et du monde à partir du Code où ils ont été exprimés. En partant du constat qu'une société et un monde s'expriment dans ce texte, il montre *ce qui* transparaît précisément de cette société et du monde, c'est-à-dire à quoi se réduit l'image qui en est donnée.
- L'œuvre sociologique de Jean Ray porte ainsi sur les objets explicitement mentionnés dans le Code.

Plan de la section

Logique et linguistique I

L'œuvre sociologique de Jean Ray porte également sur la méthode de rédaction du Code : sa logique, telle qu'on peut l'observer linguistiquement (p. 23).

L'Expression

I.

N'ouvrons pas le Code civil en juriste accoutumé à son vocabulaire et à ses formules. Appliquons-nous à voir son apparence la plus extérieure. L'austérité du texte, la sécheresse dépouillée du style, l'absence presque complète d'ornements littéraires, d'images et de métaphores ne sont pas, comme on pourrait le croire, des caractères universels et nécessaires des prescriptions juridiques. Bien des règles de cet ordre se sont traduites en symboles, exprimées en anecdotes, fixées dans des formules vives et pittoresques, aux époques surtout où le droit était mal différencié.

Modalité I

On peut ainsi étudier la modalité logique à partir du mode verbal (I, 1)...

PREMIÈRE PARTIE

La règle

CHAPITRE PREMIER

Du fait que les dispositions du Code civil se présentent très souvent sous la forme énonciative et non pas sous une forme impérative 41

Ce fait ne correspond pas à la distinction classique des dispositions impératives et des dispositions supplétives ou déclaratives. Il ne correspond pas non plus à la distinction du principal et de l'accessoire. La prépondérance de la forme énonciative caractérise un degré avancé de l'évolution juridique : à ce stade la loi n'est plus seulement commandement, elle est effort organisateur, et elle se présente comme l'exposé d'une construction intellectuelle.

Modalité I

et à travers les verbes et formules modaux (I, 2)...

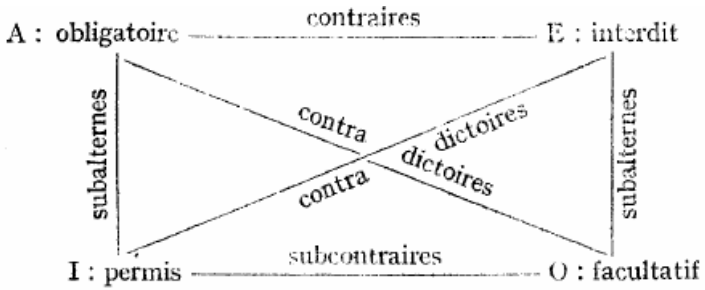
CHAPITRE II

Des formes sous lesquelles s'exprime directement la volonté de la loi : l'obligatoire, l'interdit, le permis, le facultatif. De leurs rapports logiques 50

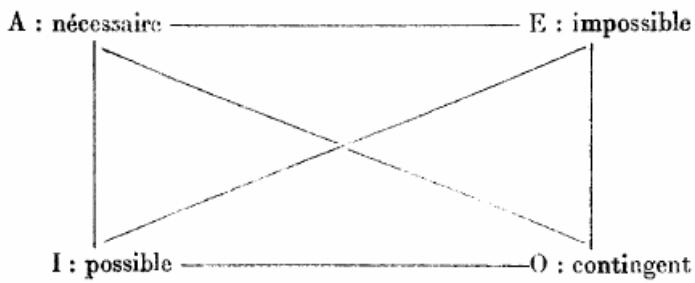
Les quatre propositions typiques apparaissent comme des modales d'une nature originale, mais correspondant au nécessaire, à l'impossible, au possible et au contingent. Elles se laissent ordonner en une table qui résume les mêmes possibilités d'inférence que la table classique des oppositions. Cette analogie s'explique si l'on se fait de la quantité logique une idée suffisamment compréhensive. Originalité des modales juridiques : elles sont orientées vers l'action.

Modalité I

en observant le parallélisme des logiques modales aléthique et déontique (p. 55) :



Cette table est comparable à celle que Pierre d'Espagne présentait pour la nécessité et la contingence[1] :



(en bas à droite : « pouvoir ne pas »)

Termes complexes, négations et restrictions I

Jean Ray étudie aussi les termes complexes, négations et restrictions (I, 5)...

Propositions complexes I

les propositions complexes : conjonction et disjonction, subordination (I, 5)...

SECTION III

La disjonction exprimée par la loi est d'autant plus rigoureuse qu'elle n'est pas constatée, mais créée.

L'essence du rapport de subordination est d'exprimer la dépendance de la principale à l'égard de la subordonnée : il implique donc considération de deux moments, au moins logiques.

Identité et équivalence I

et les notions d'identité et d'équivalence (I, 5).

CHAPITRE V

Du rôle des notions d'identité ou d'équivalence 168

SECTION I

Du passage du même au même ; du procédé d'assimilation ; du progrès logique 168

Il arrive que le législateur présente, outre le principe, ses conséquences ; a déduction est en réalité, en de tels cas, une façon de présenter une construction vraiment synthétique et partiellement arbitraire, quoique consolidée par les rapports logiques qui en lient les éléments. Plus souvent le législateur étend un principe à des matières connexes, ou à un domaine différent, mais analogue. Le procédé d'assimilation est même un des procédés habituels de la construction juridique. Du progrès logique.

Identité et équivalence I

Section II

L'affirmation de l'identité dans le temps, l'affirmation de l'égalité sont à la base même de notre droit ; elles expliquent, en même temps que sa substance intime, une multitude d'institutions. Mais tout spécialement leur influence se révèle lorsqu'il s'agit d'expliquer la rupture de l'identité ou de la permanence, la naissance d'un droit : dans le contrat, la cause est aussi essentielle que le consentement ; hors du domaine du contrat, la théorie de l'enrichissement sans cause est capitale ; la compensation du préjudice causé. Ce n'est pas seulement dans notre droit civil, c'est dans les systèmes juridiques les plus divers que les idées d'identité et d'équivalence sont au premier plan. Elles jouent dans la construction du droit un rôle comparable à celui qu'elles tiennent dans l'élaboration du savoir.

Conclusion

- Jean Ray étudie de façon systématique la logique du Code Civil, en passant en revue les notions élémentaires : modalités, termes complexes, négation, restriction, conjonction, disjonction, subordination, identité et équivalence, etc.
- C'est l'analyse linguistique qui lui permet de repérer les traces de ces notions logiques.

Plan de la section

Systèmes de propositions I

Par-dessus les propositions et les raisonnements, le Code forme un système (p. 12) :

> Enfin nous n'oublierons pas le caractère logique le plus apparent et le plus nouveau du Code ; il constitue un système, il est un ensemble, comprenant lui-même d'autres ensembles qui se subordonnent et se coordonnent. Les logiciens n'ont guère étudié au delà de la proposition qu'une forme plus complexe de la pensée : le raisonnement ; le Code nous donnera l'occasion de signaler l'existence et d'analyser la structure d'ensembles d'une tout autre sorte, mais doués vraiment de cohésion logique. Certes le Code ne présente point à cet égard, un cas isolé ; mais c'est du moins un cas typique.

Systèmes de propositions : holisme I

Comme en philosophie, le système codétermine le sens des propositions (III, 1) :

TROISIÈME PARTIE

L'architecture

CHAPITRE PREMIER

Valeur logique de certains ensembles autres que les raisonnements 193

Il y a, en dehors des raisonnements, des ensembles de propositions sur lesquels devrait s'arrêter l'attention du logicien, parce qu'ils réagissent sur la portée des propositions qui les composent. C'est là un fait général, mais le Code civil en fournit une illustration exceptionnelle, puisqu'il a voulu être un ensemble essentiellement ordonné et systématique.

Holisme : le tout détermine les parties.

Système de propositions : étude architectonique I

Étude de la structuration du Code (III, 2) :

CHAPITRE II

Les codes considérés comme des ensembles. Le cadre : plan, subdivisions, rubriques 196

Le Code civil est caractérisé par l'idée d'un traitement systématique d'une matière : il constitue, beaucoup plus que les compilations romaines et les coutumes, un ensemble. Le Code est devenu le type normal de la présentation du droit chez les peuples modernes. L'objet du Code civil est d'ailleurs imparfaitement défini : il tend à embrasser tous les principes de « droit commun » en matière de « droit privé ». La loi est considérée non seulement comme un ensemble, mais comme un ensemble complet, qui doit suffire à tout : l'article 4. Dans cet ensemble, les matières sont méthodiquement réparties et classées. On relève pourtant, jusque dans les grandes lignes du plan, l'influence de circonstances irrationnelles, par exemple le respect traditionnel du nombre 3, qui explique la choquante disproportion entre les divers livres du Code. Le classement d'une règle dans un chapitre et sous une rubrique doit guider l'interprète ; mais il ne lui impose pourtant pas une attitude rigide. Les opérations « mixtes ». La codification telle que l'a conçue notre législateur laisse au droit beaucoup de souplesse.

Systèmes de concepts I

Outre les systèmes de propositions, le Code construit aussi des classifications, c'est-à-dire des systèmes de définitions (III, 3) :

CHAPITRE III

Définitions et classifications..... 217

Les rédacteurs du Code, soucieux de laisser à la doctrine son rôle, ont voulu éviter les définitions qui sont comme l'œuvre suprême de la science ; en revanche ils se sont constamment servi des définitions dont l'objet est simplement de préciser ce dont on parle et de fournir la base de la construction. On trouve dans le Code beaucoup de définitions solidaires de classifications : par le genre prochain et la différence spécifique. Mais il y en a plusieurs autres sortes. La division. La présentation apparente des notions et de leurs rapports laisse subsister le double problème de l'unité réelle des institutions et de leur ordonnance profonde.

Plan de la section

Conclusion

- L'analyse de Jean Ray va vient au-delà de l'étude de la « structure logique » du *Code Civil* :
  1. étude sociologique et ontologique : étude de l'image inverse de la société ;
  2. étude logico-linguistique : modalités, négation, connecteurs, quantification, subordination. . . ;
  3. étude architectonique : systèmes de propositions, systèmes de concepts.

Conclusion

- Il montre ainsi les mécanismes de la « raison pratique » : l'intelligence non pas représentative, mais créatrice d'institutions. C'est la société se modelant elle-même (p. 256). « Le Code est un ensemble organique de directives. » (p. 260)
- Ce sont les ingrédients que l'on retrouve dans le projet Catala d'informatisation de certaines parties du droit.

Plan de la section

Plan de la section

Rappel des trois exigences sur l'informatisation du droit

- Rappel des trois exigences naturelles :
  1. la transparence : publicité du code ;
  2. la certitude que le code soit correct, c'est-à-dire corresponde au texte juridique ;
  3. la certitude que l'exécution du code produise le résultat escompté.
- Nous allons voir comment ont été réalisées les exigences 1 et 3, et comment faciliter la satisfaction de l'exigence 2.

Publication du code I

- Débats en France sur la publication des algorithmes de l'État : Parcoursup, etc. Les codes fournis sont parfois incomplets ou opaques.
- Première publication en avril 2016 du code de l'impôt sur le revenu dans le hackaton « #CodeImpot ».
- Loi du 7 octobre 2016 pour une république numérique
- Conséquence pour l'impôt sur le revenu : dépôt officiel des codes 2010-2017
  ⟨ dir:f796e6fda34e7b134e691f4ad6af24932b3149cd⟩

Analyse du code

Étude du code en langage M par Denis Merigoux
[Merigoux et collab.(a)Merigoux, Monat et Gaie],
[Merigoux et collab.(b)Merigoux, Monat et Protzenko]

- 4 000 variables ;
- plus de 1 000 règles de taxation numérotées de façon unique, ex. ⟨ cnt:c7c7fc93046eb41769efe93c29a86b16e2ff5c8c⟩ ;
- 48 fichiers, 92 000 lignes de code ; ex. la compilation de la version du 6 octobre 2017 ⟨ dir:ebe3348bc6b5209f7477e5cc04eac0ccca76bac0⟩ dont la compilation maison donne 535 000 lignes de C ;
- ajout en 1995 d'une rustine de 35 000 lignes de C ;
- mises à jour régulières pendant 30 ans ;
- batterie de tests ;
- deux mainteneurs, retraite imminente.

Amélioration du code par l'étude formelle I

Réduction vers un langage-noyau simplifié : μM

```
VAR0 = si (VAR1 > 0.0) alors - arr(VAR2 * 23 / 100) sinon 0.0;
TABLEAU[X; 10] = (3 * X * X + 2 * X + 1) * (1 - present(VAR1));
```

FIGURE 3 – Exemple de programme μM

Syntaxe formelle I

Définition rigoureuse de la syntaxe :

$$\langle programme \rangle \ ::= \ \langle commande \rangle \mid \langle commande \rangle \ \texttt{;} \ \langle programme \rangle$$

$$\begin{aligned}\langle commande \rangle \ ::= \ & \langle var \rangle \ \texttt{:=} \ \langle expr \rangle \\ \mid \ & \langle var \rangle \ \texttt{[X ;} \ \langle entier \rangle \ \texttt{]} \ \texttt{:=} \ \langle expr \rangle \\ \mid \ & \texttt{si} \ \langle expr \rangle \ \texttt{alors} \ \langle erreur \rangle\end{aligned}$$

$$\begin{aligned}\langle expr \rangle \ ::= \ & \langle var \rangle \mid \texttt{X} \mid \langle valeur \rangle \\ \mid \ & \langle expr \rangle \ \langle compop \rangle \ \langle expr \rangle \mid \langle expr \rangle \ \langle binop \rangle \ \langle expr \rangle \mid \langle unop \rangle \ \langle expr \rangle \\ \mid \ & \texttt{si} \ \langle expr \rangle \ \texttt{alors} \ \langle expr \rangle \ \texttt{sinon} \ \langle expr \rangle \\ \mid \ & \langle func \rangle \ \texttt{(} \ \langle expr \rangle, \ldots, \langle expr \rangle \ \texttt{)} \mid \langle var \rangle \ \texttt{[} \ \langle expr \rangle \ \texttt{]}\end{aligned}$$

$$\langle valeur \rangle \ ::= \ \texttt{indéfini} \mid \langle défini \rangle$$

$$\langle défini \rangle \ ::= \ \langle booléen \rangle \mid \langle flottant \rangle$$

Le type « flottant » vaut indifféremment pour les valeurs monétaires et pour les taux.
Pas (besoin) de `while` !

Syntaxe formelle

L'arithmétique juridique est pauvre en opérations :

$\langle compop \rangle$ ::= `<=` | `<` | `>` | `>=` | `==` | `!=`

$\langle binop \rangle$ ::= $\langle arithop \rangle$ | $\langle logicop \rangle$

$\langle arithop \rangle$ ::= `+` | `-` | `*` | `/`

$\langle logicop \rangle$ ::= `&&` | `||`

$\langle unop \rangle$ ::= `-` | `~`

$\langle func \rangle$ ::= `arr` | `inf` | `present` | `null`
| `max` | `min` | `abs` | `pos` | `pos_ou_nul`

Aucune bibliothèque n'étend cette arithmétique : impossible de calculer une racine carrée, un cosinus, etc.

Conditions d'un code informatique correct

Certifiabilité du code informatique en μM :

- étude du typage : booléens, flottants, règles de typage ;
- sémantique formelle ;
- formalisation du langage μM en Coq ⟨ cnt:c6c94fef61eea34cfe199db1cc917ec134cd5a5f⟩ (850 lignes) ;
- compilateur libre Mlang ⟨ dir:442ca9272e1343484da4e7f74a2d58e9aed27a3f⟩ en Ocaml, exécutant le code de la DGFiP ou le traduisant en Python ;
- institutionnellement : intégration de Denis Merigoux par l'équipe de développement.

Succès. . . I

Succès permis par la formalisation du langage M :

> L'application des méthodes formelles à l'artefact logiciel publié par la DGFiP ouvre des pistes très prometteuses quant à l'analyse de l'impact de l'impôt sur le revenu. Le langage M, bien qu'ancien, relève d'un choix éclairé d'architecture logicielle dans le contexte de l'administration publique des années 90. La centralisation de l'encodage du code des impôts à l'aide d'un DSL dans une implémentation unique et mutualisée entre les applications clientes nous a permis de concentrer efficacement notre travail de formalisation. La formalisation de M présentée dans cet article, permet également la compilation de M vers virtuellement n'importe quel autre langage, de manière à fournir une implémentation clés en main du code des impôts à toutes les applications qui en auraient besoin : analyses macroéconomiques, simulateurs en ligne, etc.

[Merigoux et collab.(a)Merigoux, Monat et Gaie, p. 15]

. . . et limites

Reste à satisfaire l'exigence 2 : la conformité du code informatique au texte de loi.

> Néanmoins, la correction du code M par rapport au code des impôts reste un sujet critique. En effet, un bug d'implémentation est potentiellement source de contentieux entre l'État et le particulier lésé par un calcul de l'impôt erroné. Le langage M n'a pas une structure permettant de se convaincre ou de vérifier modulairement cette correction fonctionnelle. La prochaine étape de ce travail est donc de créer un langage inspiré de M, mais qui serait utilisé pour annoter la loi dans un style de programmation littérale, article par article. Ces annotations tiendraient lieu de spécification formelle et pourraient aussi guider le législateur, à l'aide d'un outillage adapté mettant en valeur grâce à de l'inférence des incohérences ou des cas non-spécifiés.

[Merigoux et collab.(a)Merigoux, Monat et Gaie, p. 15-16]
Solution double : changement de style (programmation lettrée) et changement de flux de travail (programmation en binôme).

Plan de la section

Garantir la correspondance entre loi et programme

- Le principal problème n'est pas technique : la correspondance entre le texte de loi et le code informatique ;
  - ce n'est pas une tâche mécanisable : le texte de loi est informel, le code informatique est formel.
- La solution proposée est double :
  1. un style d'écriture : la programmation lettrée ;
  2. un flux de travail : la programmation en binôme.
- L'approche n'est donc plus simplement linguistique : elle est sociolinguistique. Elle se prête ainsi elle-même à une étude sociologique [Alauzen()].

Programmation lettrée I

- Programmation lettrée : les programmes comme œuvres littéraires.

> The past ten years have witnessed substantial improvements in programming methodology. This advance, carried out under the banner of "structured programming," has led to programs that are more reliable and easier to comprehend; yet the results are not entirely satisfactory. My purpose in the present paper is to propose another motto that may be appropriate for the next decade, as we attempt to make further progress in the state of the art. I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be *works of literature*. Hence, my title: "Literate Programming."
>
> Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do.

[Knuth(), p. 1]

Programmation lettrée : double texte, double compilation

- En programmation lettrée
  1. le code informatique contient le cahier des charges (« spécification ») et lui *subordonne* le code efficace ;
  2. l'ensemble du code est destiné à une double compilation : production d'un texte lisible par un humain (ex. PDF) et d'un code exécutable par une machine (ex. C, Python ou langage machine).

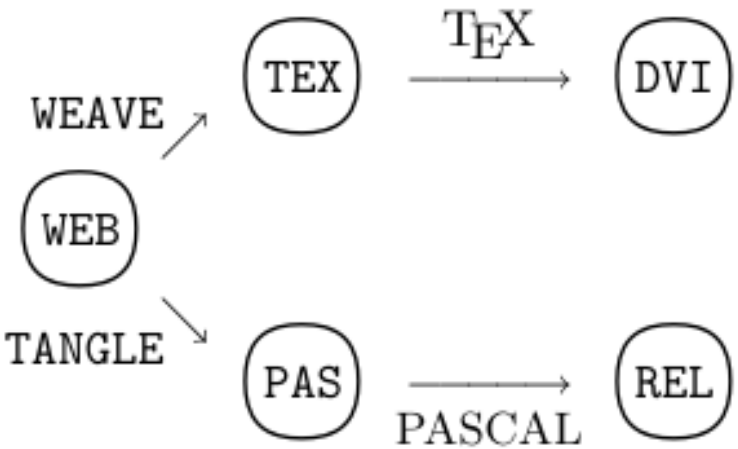


Figure 1. Dual usage of a WEB file.

Exemples

Exemple des aides au logement :

- code Catala
  ⟨ cnt:de051135a13365d2e36266537adb429186fc7ca7⟩
- compilation en PDF.

Programmation lettrée en Catala

- Les instructions par défaut de Catala sont du texte naturel, dans le langage de formatage de documents Markdown
  - le code efficace est compris entre des balises spéciales

```
9  ###### Section 1 : Règles de non-cumul
10
11 ######## Article R821-1 | LEGIARTI000038879023
12
13 En vertu de la règle énoncée à l'article L. 821-2 , une aide personnelle au
14 logement ne peut être attribuée, au profit d'une même personne ou d'un même
15 ménage, au titre de plusieurs logements.
16
17 ```catala
18 # À voir quelles sont exactement les condition pour la résidence principale,
19 # mais nous faisons ici l'hypothèse simplificatrice qu'il n'y a qu'une
20 # résidence principale. Si ce n'est pas le cas, alors l'utilisateur du programme
21 # pourra ne déclarer qu'une seule résidence principale (de son choix).
22 ```
```

⟨ cnt:de051135a13365d2e36266537adb429186fc7ca7⟩

Programmation lettrée : deux niveaux de commentaire I

Le code comprend donc deux niveaux de commentaire :

1. le commentaire externe au code, qui est la spécification informelle, l'objectif visé ;
2. le commentaire interne au code (introduit par #), qui détaille les choix techniques.

Programmation lettrée : deux niveaux de commentaire II

Article L822-5

Les aides personnelles au logement ne sont dues qu'aux personnes payant un minimum de loyer, compte tenu de leurs ressources et de la valeur en capital de leur patrimoine, lorsque cette valeur est supérieure à un montant fixé par voie réglementaire.

code_construction_legislatif.catala_fr

```
242 # Cet article est une déclaration d'intention qui donne des propriétée sur
243 # la formule de calcul. Dans le futur il serait interessant de vérifier que
244 # la formule est bien de cette forme là.
245 champ d'application ÉligibilitéAidesPersonnelleLogement:
246   définition patrimoine_total_demandeur égal à
247     demandeur.patrimoine.produisant_revenu_période_r822_3_3_r822_4 +€
248     demandeur.patrimoine.ne_produisant_pas_revenu_période_r822_3_3_r822_4
249
250   étiquette l822_5_1 définition patrimoine_pris_en_compte égal à
251     si patrimoine_total_demandeur >€ seuil_l822_5_patrimoine alors
252       patrimoine_total_demandeur
253     sinon
254       0€
```

Flux de travail

1. Écriture du code Catala : remplir un fichier d'extension .catala_fr ou .catala_en
2. Vérification du typage

   ```
   catala typecheck hello_world.catala_en
   ```

3. Exécution d'une portion de code

   ```
   catala interpret hello_world.catala_en --scope=HelloWorl
   ```

4. Compilation en un fichier PDF, lisible par des juristes : vérification manuelle de la conformité au texte de loi : programmation en binôme ;
5. Compilation en un code exécutable (partiellement) certifié en Python, C, etc.

Plan de la section

Types de base du droit

Types de base nécessaires dans ces textes juridiques :

- **booléens** ex. tester une condition ;
- **nombres entiers** ex. nombre d'enfants ;
- **nombres décimaux** ex. taux (avec précision arbitraire) ;
  - **argent** nombre à deux décimales ;
  - **date** ex. échéance ;
  - **durée** ex. période de validité ;
  - **liste**
  - **n-uplet**

Ces types correspondent à l'ontologie que Jean Ray essaie de dégager du droit : ce sont les types élémentaires d'entités sur lesquels on a besoin de raisonner juridiquement.

Types de base I

Langage fortement typé : toute valeur a un type explicite.

```
vrai                    faux                booléen
65536                                       entier
65536,262144            37%                 décimal
1 234 567,89€                               argent
|2024-04-01|                                date
254 jour      4 mois        1 an            durée
[ 12; 24; 36 ]                              liste de entier
(|2024-04-01|, 30€, 1%)                     (date,argent,décimal)
```

Types brillant par leur absence

- Quelques absences remarquables : les chaînes et même les caractères

> The absence of strings in Catala is *a feature, not a bug*. Catala is meant to be a domain-specific programming language for computations described in legal texts, that lawyers understand. If you find a legal text that requires actual string manipulation operations to be automated, please tell the Catala team! In absence of such a legal text, the decision was made to not include strings, for several reasons.

[Cat(), 4-2]

Structures (type produit, avec projections)

```
déclaration structure Struct1:
  donnée chp1 contenu entier
  donnée chp2 contenu décimal
```

```
# The name of the structure, "Individual", must start with an
# uppercase letter: this is the CamelCase convention.
declaration structure Individual:
  # In this line, "income" is the name of the structure field and
  # "money" is the type of what is stored in that field.
  # Available types include: "integer", "decimal", "money", "date",
  # "duration", and any other structure or enumeration that you declare.
  data income content money
  # The field names "income" and "number_of_children" start by a
lowercase
  # letter, they follow the snake_case convention.
  data number_of_children content integer
```

Énumération (type somme, avec filtrage par motif)

```
déclaration énumération Énum1:
  -- Cas1 contenu entier
  -- Cas2
```

```
# The name "TaxCredit" is also written in CamelCase.
declaration enumeration TaxCredit:
  # The line below says that "TaxCredit" can be a "NoTaxCredit"
situation.
  -- NoTaxCredit
  # The line below says that alternatively, "TaxCredit" can be a
  # "ChildrenTaxCredit" situation. This situation carries a content
  # of type integer corresponding to the number of children concerned
  # by the tax credit. This means that if you're in the
"ChildrenTaxCredit"
  # situation, you will also have access to this number of children.
  -- ChildrenTaxCredit content integer
```

Usage d'une énumération : filtrage par motif

```
match tax_credit with pattern
-- NoTaxCredit: $0
-- ChildrenTaxCredit of number_of_eligible_children:
  $10,000 * number_of_eligible_children
```

Combiner structure et énumération

```
declaration structure Individual:
  data date_of_birth content date
  data date_of_death content DateOfDeath

declaration enumeration DateOfDeath:
  -- Deceased content date
  -- StillAlive

# Declaration and definition of a constant value named test_person1
declaration test_person1 content Individual equals
  Individual {
    -- date_of_birth: |1981-10-05|
      # Date format is |YYYY-MM-DD|
    -- date_of_death: Deceased content |2012-05-12|
  }
```

Catala offre ainsi les outils pour définir tous les objets que demande de manipuler le Code des Impôts : types de base et types dérivés.

Plan de la section

Opérateurs

Catala offre également toutes les opérations exigées par le Code des Impôts :

```
non a          a et b                    Opérateurs logiques
a ou b         # "ou à défaut"
a ou bien b    # ou exclusif

- a           a + b        a - b        Arithmétique
a * b         a / b

a = b         a < b        a <= b       Comparaisons
a != b        a > b        a >= b

décimal de 44                           Conversions
argent de 23,15

arrondi de 9,99€                        Arrondis

accès_année de ...                      Éléments de dates
premier_jour_du_mois de ...

a +! b    a +. b    a +€ b    a +^ b    Opérateurs à types
# entier décimal   argent    durée      explicites
```

- Une bibliothèque arithmétique très limitée ! pas de racine carrée, etc.

L'arithmétique juridique

Règles et limites de l'arithmétique juridique :

- entier divisé par entier = décimal ;
- l'agent ne peut être multiplié par de l'argent, mais seulement par un décimal ;
- l'agent multiplié ou divisé par un décimal est arrondi au centime le plus proche ;
- l'argent divisé par de l'argent donné un décimal (non arrondi).

```
10 / 3 = 3.333333333...
$10 / 3.0 = $3.33
$20 / 3.0 = $6.67
$10 / $3 = 3.33333333...
```

L'arithmétique juridique des dates

L'arithmétique des dates est très compliquée [Monat et collab.()Monat, Fromherz et Merigoux] :

- ex. 31 janvier + 1 mois = 28 février, 29 février, 1er mars, 3 mars ?
- On ne peut convertir les jours en mois ou l'inverse : « comparing durations in days to durations in months is ambiguous and requires legal interpretations » (https://book.catala-lang.org/5-2-types.html#dates).

Plan de la section

Structures de contrôle juridiques

Catala offre les structures de contrôle exigées par le Code des Impôts, et seulement elles !

- `if` :

```
si ... alors ... sinon ...
```

- ni `while`, ni `for`, ni évidemment `goto` (proscrit par Dijkstra [Dijkstra()]).

Incomplétude au sens de Turing

- Böhm et Jacopini 1966 [Böhm et Jacopini()], les structures nécessaires à tout langage de programmation complet :
  1. séquence d'instructions : oui ;
  2. alternative (`if`) : oui ;
  3. répétition (`while` ou `for`) : non.
- Catala n'est donc pas complet au sens de Turing : il n'a pas la récursion générale (toutes les fonctions terminent)

compute a suitable dependency order for all the definitions in a given program. Fortunately, the law does not have general recursion, meaning that we do not need to compute fixed points, and do not risk running into circular definitions. Hence, our language is not Turing-complete, purposefully.

Champs d'application (*scopes*) I

- Catala est un langage de programmation fonctionnel : tout le code est dans une fonction.
- En Catala, les fonctions sont appelées champs d'application, et possèdent :
  1. un nom ;
  2. des variables d'entrée (arguments de la fonction) : définies à l'extérieur, ne peuvent être définies dans le champ d'application ;
  3. des variables internes (variables locales) : définies dans le champ d'application, invisibles à l'extérieur ;
  4. des variables de sortie (dont l'ensemble forme le type de retour de la fonction) : visibles de l'extérieur si l'on appelle la fonction.

Champs d'application (*scopes*) II

```
# Scope names use the CamelCase naming convention, like names of structs
# or enums Scope variables, on the other hand, use the snake_case naming
# convention, like struct fields.
declaration scope IncomeTaxComputation:
  # The following line declares an input variable of the scope, which is
akin to
  # a function parameter in computer science term. This is the piece of
  # data on which the scope will operate.
  input individual content Individual
  internal tax_rate content decimal
  output income_tax content money
```

Champs d'application (*scopes*) I

> **Article 1**
>
> The income tax for an individual is defined as a fixed percentage of the individual's income over a year.

```
scope IncomeTaxComputation:
  definition income_tax equals
    individual.income * tax_rate
```

Champs d'application (*scopes*) II

> **Article 2**
>
> The fixed percentage mentioned at article 1 is equal to 20 %.

```
scope IncomeTaxComputation:
  # Writing 20% is just an alternative for the decimal "0.20".
  definition tax_rate equals 20 %
```

- (Un champ d'application peut aussi être imbriqué dans un autre.)

Plan de la section

Un langage limité

- Catala n'est donc pas un langage généraliste ni complet comme C ou Python :
  - certains types très élémentaires sont absents : chaînes, caractères ;
  - les opérations sont en très petit nombre, et ne sont pas étendues par une bibliothèque mathématique ;
  - absence de récursion générale.
- Ces limites sont intentionnelles : Catala contient toutes *et seulement* les structures calculatoires exigées par le texte de loi.

Plan de la section

Plan de la section

Le traitement des exceptions

- Le Catala Book décrit « the killer feature of Catala when it comes to coding the law : exceptions in the definitions of variables ».

Traitement habituel des exceptions en informatique

- Habituellement, en programmation, les cas particuliers sont traités en premier et le cas général en dernier :

```
if (nombreEnfants == 0) { ... }
else if (nombreEnfants == 1) { ... }
else if (nombreEnfants == 2) { ... }
else { ... }
```

Traitement habituel des exceptions en logique

- Du point de vue logique, cela permet de construire progressivement une conjonction de formules sans contradiction
  - $(n = 0 \rightarrow ...)$
  - $(n = 0 \rightarrow ...) \wedge (n = 1 \rightarrow ...)$
  - $(n = 0 \rightarrow ...) \wedge (n = 1 \rightarrow ...) \wedge (n = 2 \rightarrow ...)$
  - $\ldots$
  - $(n = 0 \rightarrow ...) \wedge (n = 1 \rightarrow ...) \wedge (n = 2 \rightarrow ...) \wedge ... \wedge (n \neq 0 \wedge n \neq 1 \wedge n \neq 2 \wedge ... \rightarrow ...)$

Les exceptions en droit

- Mais dans les textes de droit, le cas général vient en premier :
  1. art. 2 : « Le taux fixé mentionné à l'article 1 est de 20 %. »
  2. art. 3 : « Si l'individu est en charge de 2 enfants ou plus, alors le pourcentage mentionné à l'article 1 est de 15 %. »
- Or, on ne peut pas écrire :
  1. $t = 20$
  2. $t = 20 \land (n \geq 2 \rightarrow t = 15)$

  car on pourrait avoir à la fois n = 20 et n = 15.
- Il faudrait donc écrire :
  1. $t = 20$
  2. effacer
  3. $(n < 2 \rightarrow t = 20) \land (n \geq 2 \rightarrow t = 15)$
- La formulation du cas général dépend donc de l'énumération de toutes ses exceptions ! Cela ne correspond pas à la notion intuitive de « cas général » telle qu'elle est utilisée en droit.

Plan de la section

La logique du raisonnement par défaut en droit

Sarah Lawsky, « A Logic for Statutes » [Lawsky(b)] :

- On pense souvent que la difficulté du raisonnement juridique réside dans le raisonnement à partir de cas, c'est-à-dire la jurisprudence (raisonnement par analogie, etc.), mais que le raisonnement à partir de règles ne pose pas de problème particulier : ce serait le raisonnement déductif habituel ;
- en réalité, le raisonnement à partir de règles aussi pose de gros problèmes : les textes de loi sont « défaisables », c'est-à-dire non monotones : une information ultérieure peut nous faire réviser une conclusion antérieure ;
- la logique du [raisonnement par] défaut permet de formaliser ce type de raisonnement en adoptant le même ordre que le discours juridique.

La logique du raisonnement par défaut : motivation I

Raymond Reiter : logique du raisonnement par défaut (*default logic*) [Reiter()] :

- vient du monde des bases de données ;
- une banque n'a pas de liste des non-membres : si une personne n'est pas dans la liste des membres, alors on va supposer qu'elle est non-membre (hypothèse du monde clos) ;
- cela peut changer, donc il faut une logique non monotone : possibilité de révision.

La logique du raisonnement par défaut : définition

Une logique du raisonnement par défaut contient :

1. un ensemble de formules logiques : les faits connus avec certitude
2. un ensemble de règles de défaut de la forme

$$\frac{A(x) : B(x), \ldots, C(x)}{D(x)}$$

   1. A(x) est le prérequis
   2. B(x), etc. sont les justifications : ce qui doit être consistant avec l'ensemble des croyances
   3. D(x) est le conséquent, que l'on peut dériver du prérequis sous réserve que les justifications ne soient pas transgressées (dans le cas « normal », conséquent = justification)

Si A(x) et que rien n'empêche B(x), etc., alors D(x). Dans le cas normal, B(x) = D(x).

La logique du raisonnement par défaut : exemple

Ex. « les oiseaux volent, mais pas les manchots » :

- W = {oiseau(pigeon), oiseau(manchot), ¬ vole(manchot)}
- D = {

$$\frac{\text{oiseau}(X) : \text{vole}(X)}{\text{vole}(X)}$$

}

- On peut dériver `vole(pigeon)` car cela n'entre pas en contradiction avec une connaissance tirée de W ;
- mais on ne peut pas dériver `vole(manchot)` car cela entre en contradiction avec une connaissance tirée de W.

La logique du raisonnement par défaut priorisée

- La logique du raisonnement par défaut priorisée ajoute un ordre dans les règles [Brewka et Eiter()].
- Ex. « une personne habitant les États-Unis sait lire, mais les enfants ne savent pas lire » :
  - W = {États-Unis, Jeune}
  - $D = d_1, d_2$
  - d_1 : États-Unis $\rightarrow$ Lit
  - d_2 : Jeune $\rightarrow \neg$ Lit
  - $<$: $d_1 < d_2$
  - donc un habitant des États-Unis sait lire, sauf si c'est un enfant — car c'est *plus* un enfant qu'un hébitant des États-Unis.

Plan de la section

Conflit entre définitions I

Il arrive souvent qu'un article redéfinisse une variable déjà définie plus tôt dans le même champ d'application, ce qui pourrait mener à des conflits entre définitions :

> **Article 1**
>
> The income tax for an individual is defined as a fixed percentage of the individual's income over a year.

```
scope IncomeTaxComputation:
  definition income_tax equals
    individual.income * tax_rate
```

Conflit entre définitions

> **Article 2**
>
> The fixed percentage mentioned at article 1 is equal to 20 %.

```
scope IncomeTaxComputation:
  # Writing 20% is just an alternative for the decimal "0.20".
  definition tax_rate equals 20 %
```

> **Article 3**
>
> If the individual is in charge of 2 or more children, then the fixed percentage mentioned at article 1 is equal to 15 %.

```
scope IncomeTaxComputation:
  definition tax_rate under condition
    individual.number_of_children >= 2
  consequence equals 15 %
```

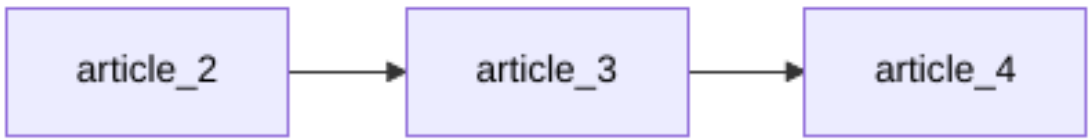
Solution : déclarer l'exception

Il faut donc déclarer les exceptions les unes après les autres :

> **Article 3**
>
> If the individual is in charge of 2 or more children, then the fixed percentage mentioned at article 1 is equal to 15 %.

Defining an exception for a variable

```
scope IncomeTaxComputation:
  exception definition tax_rate under condition
    individual.number_of_children >= 2
  consequence equals 15 %
```

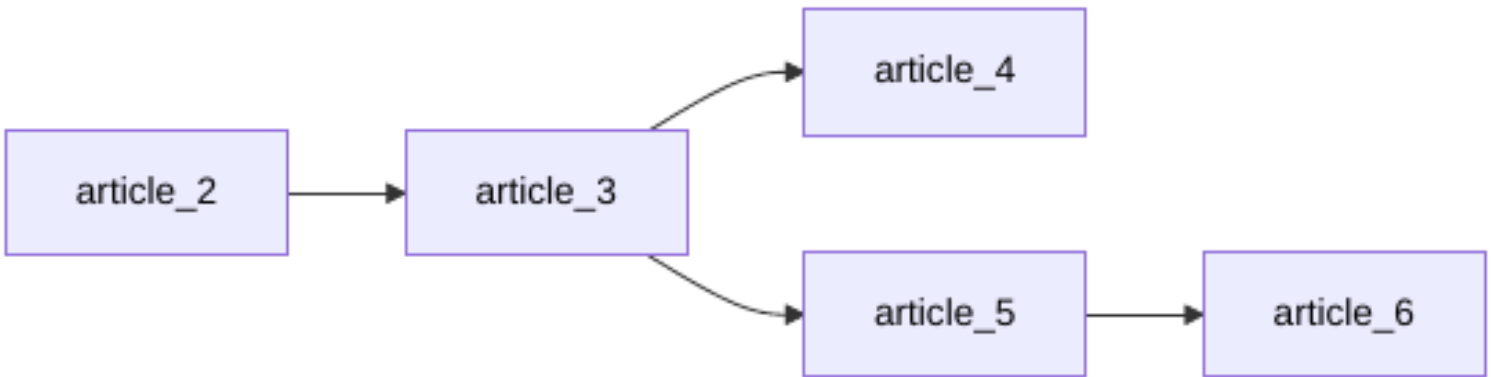


Gestion d'exceptions arborescentes

On peut aussi gérer des arbres d'exception :

> **Article 4**
>
> Individuals earning less than $10,000 are exempted of the income tax mentioned at article 1.

```
scope IncomeTaxComputation:
  exception definition tax_rate under condition
    individual.income <= $10,000
  consequence equals 0 %
```



Gestion d'exceptions arborescentes

Article 2

The fixed percentage mentioned at article 1 is equal to 20 %.

```
scope IncomeTaxComputation:
  # The keyword "label" introduces the name of the label itself, here
  # "article_2".
  label article_2 definition tax_rate equals 20 %
```

Article 3

If the individual is in charge of 2 or more children, then the fixed percentage mentioned at article 1 is equal to 15 %.

```
scope IncomeTaxComputation:
  # This definition is preceded by two indications:
  # * it has its own label, "article_3";
  # * this definition is an exception to the definition labeled
"article_2".
  label article_3 exception article_2
  definition tax_rate under condition
    individual.number_of_children >= 2
  consequence equals 15 %
```

Plan de la section

Plan de la section

Conclusion

- L'informatisation du droit mobilise des outils logiques et linguistiques :
  - pour informatiser le droit, il faut le formaliser ;
  - pour le formaliser, il faut analyser son langage.
- Cette tâche suppose, en amont, une interaction entre juristes et informaticiens comme le permet le couplage de la programmation lettrée et de la programmation en binôme.
- Cette tâche logique, linguistique et sociale est une condition préalable à la satisfaction d'exigences citoyennes telles que la transparence de la loi, de ses applications et de ses méthodes d'application.

Bibliographie I


« The Catala domain-specific programming language », https://book.catala-lang.org/.


« Software heritage », https://www.softwareheritage.org/?lang=fr.


Alauzen, M. « L'altération informatique du droit. Une sociologie du passage du droit aux droits », vol. 117, n° 2, p. 293–315, https://droit.cairn.info/revue-droit-et-societe-2024-2-page-277?tab=texte-integral.

Bibliographie II


Brewka, G. et T. Eiter. « Prioritizing Default Logic », dans *Intellectics and Computational Logic : Papers in Honor of Wolfgang Bibel*, sous la direction de S. Hölldobler, Springer Netherlands, p. 27–45, ISBN 978-94-015-9383-0, doi : 10.1007/978-94-015-9383-0_3.


Böhm, C. et G. Jacopini. « Flow diagrams, turing machines and languages with only two formation rules », vol. 9, n° 5, p. 366–371, doi : 10.1145/355592.365646.


Di Cosmo, R. et S. Zacchiroli. « Software heritage : Why and how to preserve software source code », dans *iPRES 2017-14th International Conference on Digital Preservation*, p. 1–10.

Bibliographie III


Dijkstra, E. W. « Go To Statement Considered Harmful », dans *Pioneers and Their Contributions to Software Engineering*, sous la direction de M. Broy et E. Denert, Springer Berlin Heidelberg, p. 297–300, ISBN 978-3-540-42290-7 978-3-642-48354-7, doi : 10.1007/978-3-642-48354-7$_1$2.


Knuth, D. E. « Literate programming », vol. 27, n° 2, p. 97–111, https://academic.oup.com/comjnl/article-abstract/27/2/97/343244.


Lawsky, S. B. a, « Coding the Code : Catala and Computationally Accessible Tax Law », https://papers.ssrn.com/abstract=4291177.

Bibliographie IV

- Lawsky, S. B. b, « A Logic for Statutes », doi : 10.2139/ssrn.3088206.
- Lessig, L. « Code is Law », https://framablog.org/2010/05/22/code-is-law-lessig/.
- Merigoux, D., R. Monat et C. Gaie. a, « Étude formelle de l'implémentation du code des impôts », https://inria.hal.science/hal-02320347.
- Merigoux, D., R. Monat et J. Protzenko. b, « A Modern Compiler for the French Tax Code », ACM, p. 71, doi : 10.1145/3446804.3446850.

Bibliographie V


Monat, R., A. Fromherz et D. Merigoux. « Formalizing Date Arithmetic and Statically Detecting Ambiguities for the Law », dans *Lecture Notes in Computer Science*, *Lecture Notes in Computer Science*, vol. 14577, sous la direction de S. Weirich, Springer Nature Switzerland, *Lecture Notes in Computer Science*, vol. 14577, p. 421–450, doi : 10.1007/978-3-031-57267-8$_1$6.


Ray, J. *Essai sur la structure logique du Code civil français*, F. Alcan.


Reiter, R. « A logic for default reasoning », vol. 13, n° 1–2, p. 81–132, https://www.sciencedirect.com/science/article/pii/0004370280900144.