A Formal Study of the French Tax Code's Implementation

Denis Merigoux

Prosecco, Inria

Feb 18th, 2020

Law and algorithms

Code Général des Impôts, Article 197, I, 3, b, 3°

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, et ;
— au dénominateur, 2 000 €, pour [...], ou 4 000 €, pour [...].

Law and algorithms

Code Général des Impôts, Article 197, I, 3, b, 3°

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, et ;
— au dénominateur, 2 000 €, pour [...], ou 4 000 €, pour [...].

When does the law specify an algorithm?

Law and algorithms

Code Général des Impôts, Article 197, I, 3, b, 3°

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, et ;
— au dénominateur, 2 000 €, pour [...], ou 4 000 €, pour [...].

When does the law specify an algorithm?

- Decision without human intervention
- No ambiguity
- Quantitative data (income, number of children, etc.)

An example of algorithmic translation

```
RATE = if (not [...1...]) then
  20 %
```

“Le taux de la réduction prévue au premier alinéa du présent b est de 20 %. Toutefois, pour [...(1)...],...”

An example of algorithmic translation

```
RATE = if (not [...1...]) then
  20 %
else
  20 % * (
    (


    )
    /
    (                                                  )
  )
```

“ il est égal à 20 % multiplié par le rapport entre :
— au numérateur, ...;
— au dénominateur, ...”

An example of algorithmic translation

```
RATE = if (not [...1...]) then
  20 %
else
  20 % * (
    (
      (if [...2,3...] then 20 500 € else 41000 €)
       - INCOME
    )
    /
    (                                          )
  )
```

“au numérateur, la différence entre 20 500 €, pour [...(2)...], ou 41 000 €, pour [...(3)...], et le montant des revenus mentionnés au troisième alinéa du présent b”

An example of algorithmic translation

```
RATE = if (not [...1...]) then
  20 %
else
  20 % * (
    (
      (if [...2,3...] then 20 500 € else 41000 €)
       - INCOME
    )
    /
    (if [...4,5...] then 2000 € else 4000 €)
  )
```

“au dénominateur, 2 000 €, pour [...(4)...], ou 4 000 €, pour [...(5)...].”

Formal study of the tax computation

What specification for the tax computation?

Formal study of the tax computation

What specification for the tax computation?

- inputs : income, number of children, etc
- output $= f(\text{inputs})$: amount of tax

f should be studied formally!

Formal study of the tax computation

What specification for the tax computation?

- inputs : income, number of children, etc
- output $= f(\text{inputs})$: amount of tax

f should be studied formally!

f's properties:

- Is f *increasing* with income?
- Is f *decreasing* with the number of children?

Formal study of the tax computation

What specification for the tax computation?

- inputs : income, number of children, etc
- output $= f(\text{inputs})$: amount of tax

f should be studied formally!

f's properties:

- Is f *increasing* with income?
- Is f *decreasing* with the number of children?
- What is f's *derivative*? $\Rightarrow$ marginal tax rate

Case study: marginal tax rate

Model

Characteristic	Household before	Household after
Yearly income	R_0	$R_0 + \Delta_R$

Case study: marginal tax rate

Model

Characteristic	Household before	Household after
Yearly income	R_0	$R_0 + \Delta_R$
Income tax	T_0	$T_0 + \Delta_T$
Benefits	B_0	$B_0 - \Delta_B$

Case study: marginal tax rate

Model

Characteristic	Household before	Household after
Yearly income	R_0	$R_0 + \Delta_R$
Income tax	T_0	$T_0 + \Delta_T$
Benefits	B_0	$B_0 - \Delta_B$
Take-home income	$I_0 = R_0 - T_0 + B_0$	$I_0 + \Delta_R - \Delta_T - \Delta_B$

Case study: marginal tax rate

Model

Characteristic	Household before	Household after
Yearly income	R_0	$R_0 + \Delta_R$
Income tax	T_0	$T_0 + \Delta_T$
Benefits	B_0	$B_0 - \Delta_B$
Take-home income	$I_0 = R_0 - T_0 + B_0$	$I_0 + \Delta_R - \Delta_T - \Delta_B$

Effective withholding marginal rate

$$R_{\text{eff}} = \frac{\Delta_T + \Delta_B}{\Delta_R}$$

Case study: marginal tax rate

Model

Characteristic	Household before	Household after
Yearly income	R_0	$R_0 + \Delta_R$
Income tax	T_0	$T_0 + \Delta_T$
Benefits	B_0	$B_0 - \Delta_B$
Take-home income	$I_0 = R_0 - T_0 + B_0$	$I_0 + \Delta_R - \Delta_T - \Delta_B$

Effective withholding marginal rate

$$R_{\text{eff}} = \frac{\Delta_T + \Delta_B}{\Delta_R}$$

Is there a household such that $R_{\text{eff}} \geqslant 70\%$?

Our hammer : SMT solving

How to answer that?

Our hammer : SMT solving

How to answer that?

- Enumerate all possible households?
  $\Rightarrow$ inefficient...

Our hammer : SMT solving

How to answer that?

- Enumerate all possible households?
  $\Rightarrow$ inefficient...
- Test on real data?
  $\Rightarrow$ fiscal secret, corner cases

Our hammer : SMT solving

How to answer that?

- Enumerate all possible households?
  $\Rightarrow$ inefficient...
- Test on real data?
  $\Rightarrow$ fiscal secret, corner cases
- Optimization / *machine learning*?
  $\Rightarrow$ f too complex?

Our hammer : SMT solving

How to answer that?

- Enumerate all possible households?
  $\Rightarrow$ inefficient...
- Test on real data?
  $\Rightarrow$ fiscal secret, corner cases
- Optimization / *machine learning*?
  $\Rightarrow$ f too complex?

Satisfiability modulo theories

SMT solver able to solve complex numerical constraints problems.

$\Rightarrow$ Implemented in a prototype considering a simplified household model

Counter-example

The SMT solver finds something!

Counter-example

The SMT solver finds something!

Cohabiting couple with two high-schoolers (15 and 17-years-old), depending on second parent (jobless). Zone II monthly rent €897,75. Monthly raise of €250/€2 760,76.

Counter-example

The SMT solver finds something!

Cohabiting couple with two high-schoolers (15 and 17-years-old), depending on second parent (jobless). Zone II monthly rent €897,75. Monthly raise of €250/€2 760,76.

Characteristic	Value before	Value after	Variation
Yearly income R	€33 129,12	€36 129,12	+ **€3 000,00**
IR	€3 147,00	€3 957,00	+ €810,00
PA	€1 320,00	€0,00	− €1 320,00
AF	€1 584,00	€1 584,00	€0,00
ARS	€806,00	€0,00	− €806,00
BC	€0,00	€0,00	€0,00
BL	€0,00	€0,00	€0,00
APL	€0,00	€0,00	€0,00
Take-home income N	€33 692,12	€33 756,12	+ **€64,00**
Marginal rate			**97,9 %**

SMT-verified fiscal legislation?

Avantages

- Systematic analysis
- Very expressive model
- No data needed

SMT-verified fiscal legislation?

Avantages

- Systematic analysis
- Very expressive model
- No data needed

Scalability challenge

- Complexity $\Rightarrow$ resources and compute time
- SMT queries need optimizing
- Going beyond prototyping

From prototype to production

The DGFiP (French IRS) released a part of its tax computation system in 2016:

- `https://framagit.org/dgfip/ir-calcul`
- Written in M, custom language
- Computes income taxes for private individuals
- 2017 version: 48 files, 92,000 lines of code

From prototype to production

The DGFiP (French IRS) released a part of its tax computation system in 2016:

- `https://framagit.org/dgfip/ir-calcul`
- Written in M, custom language
- Computes income taxes for private individuals
- 2017 version: 48 files, 92,000 lines of code

Leveraging public code

- M language formalized [2]
- Pending on missing information from DGFiP

Are the algorithms faithful to the law?

Current code validation

Best case scenario: a few test cases written by lawyers

- must update tests when law changes
- code coverage?

Are the algorithms faithful to the law?

Current code validation

Best case scenario: a few test cases written by lawyers

- must update tests when law changes
- code coverage?

Currently, no organization can claim a correct-by-construction law implementation:

- Governement-issued simulators (social benefits)
- DGFiP (income tax)
- Private companies: PayFit, ADP (payroll taxes)

A locally verifiable formal specification

Correct-by-construction lax implementation?

A locally verifiable formal specification

Correct-by-construction lax implementation? $\Rightarrow$ litterate programming paradigm

A locally verifiable formal specification

Correct-by-construction lax implementation? $\Rightarrow$ litterate programming paradigm

Example

The tax reduction is equal to 20% of the income...

```
/* tax.redution := income * 20 / 100 */
```

... except when the income is below €20 000, where it is 30%.

```
/* tax.redution when income < 20000 := income * 30 / 100 */
```

A locally verifiable formal specification

Correct-by-construction lax implementation? $\Rightarrow$ litterate programming paradigm

Example

The tax reduction is equal to 20% of the income...

```
/* tax.redution := income * 20 / 100 */
```

... except when the income is below €20 000, where it is 30%.

```
/* tax.redution when income < 20000 := income * 30 / 100 */
```

The programming language should follow the structure of the law!

A default logic-based DSL

Default logic

Proposed by Reiter in 1980 [3]. Basic idea is to give a precedence order on logic predicates.

$$\text{default case} < \text{exceptions}$$

A default logic-based DSL

Default logic

Proposed by Reiter in 1980 [3]. Basic idea is to give a precedence order on logic predicates.

$$\text{default case} < \text{exceptions}$$

The law specification DSL should use default logic [1]. DSL programs:

- validated by law professionals
- extracted to conventional programming languages

A default logic-based DSL

Default logic

Proposed by Reiter in 1980 [3]. Basic idea is to give a precedence order on logic predicates.

$$\text{default case} < \text{exceptions}$$

The law specification DSL should use default logic [1]. DSL programs:

- validated by law professionals
- extracted to conventional programming languages

$\Rightarrow$ Topic of a Master internship at Prosecco in Spring 2020.

Conclusion

Current work and beyond:

- SMT-based tax impact study
- Formalization of the official income tax computation codebase
- Tooling for correct-by-construction law implementations

Open-source code: `https://gitlab.inria.fr/verifisc`

Thanks to DGFiP and in particular Christophe Gaie and his team for collaborating!

References I

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

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

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