A Modern Compiler for the French Tax Code

Raphaël Monat, with Denis Merigoux and Jonathan Protzenko
Introduction
Income Tax in France

Each year

- Tax computation by DGFiP

38M fiscal households
€75B = 30% of the State's income

Trusting the computation?
Correct computation with respect to the law
Reproducibility of the computation
Accurate simulation of tax reforms
Income Tax in France

Each year

- Tax computation by DGFiP
- 38M fiscal households
Income Tax in France

Each year

- Tax computation by DGFiP
- 38M fiscal households
- €75B

Correct computation with respect to the law
Reproducibility of the computation
Accurate simulation of tax reforms
Income Tax in France

Each year

- Tax computation by DGFiP
- 38M fiscal households
- €75B = 30% of the State’s income
Income Tax in France

Each year

- Tax computation by DGFiP
- 38M fiscal households
- €75B = 30% of the State’s income

Tulassing the computation?

- Correct computation with respect to the law
- Reproducibility of the computation
- Accurate simulation of tax reforms
Income Tax Code

Made public in April 2016
Made public in April 2016
Updated every year
Income Tax Code

- Made public in April 2016
- Updated every year
- https://gitlab.adullact.net/dgfip/ir-calcul

```plaintext
IRNETTER = max(0, IRNET2 + (TAXASSUR + PTAXA - min(TAXASSUR + PTAXA + 0, max(0, INE - IRB + AVFISCOPTER)) - max(0, TAXASSUR + PTAXA - min(TAXASSUR + PTAXA + 0, max(0, INE - IRB + AVFISCOPTER))) + min(0, IRNET2)) + (IPCAPTAXT + PPCAP - min(IPCAPTAXT + PPCAP, max(0, INE - IRB + AVFISCOPTER - TAXASSUR - PTAXA)) - max(0, IPCAPTAXT + PPCAP - min(IPCAPTAXT + PPCAP, max(0, INE - IRB + AVFISCOPTER - TAXASSUR - PTAXA))) + min(0, TAXANEG)) + (TAXLOY + PTAXLOY - min(TAXLOY + PTAXLOY, max(0, INE - IRB + AVFISCOPTER - TAXASSUR - PTAXA - IPCAPTAXT - PPCAP)) - max(0, TAXLOY + PTAXLOY - min(TAXLOY + PTAXLOY, max(0, INE - IRB + AVFISCOPTER - TAXASSUR - PTAXA - IPCAPTAXT - PPCAP)) + min(0, PCAPNEG))) + (IHAUTREVT + PHAUTREV + CHRPVIMP - max(0, IHAUTREVT + PHAUTREV + CHRPVIMP + min(0, LOYELEVNEG)));
```
Income Tax Code

- Made public in April 2016
- Updated every year
- [https://gitlab.adullact.net/dgfip/ir-calcul](https://gitlab.adullact.net/dgfip/ir-calcul)

48 files, 92,000 lines of code written in a custom language, M.
Income Tax Code

- Made public in April 2016
- Updated every year
- [https://gitlab.adullact.net/dgfip/ir-calcul](https://gitlab.adullact.net/dgfip/ir-calcul)

48 files, 92,000 lines of code written in a custom language, M.

Sample code

```plaintext
IRNETTER = max(0, IRNET2
+ (TAXASSUR + PTAXA - min(TAXASSUR+PTAXA+0,max(0,INE-IRB+AVFISCOPTER))
- max(0,TAXASSUR + PTAXA - min(TAXASSUR + PTAXA + 0,max(0,INE-IRB+AVFISCOPTER))+ min(0,IRNET2))
+ (IPCAPTAXT + PPCAP - min(IPCAPTAXT + PPCAP,max(0,INE-IRB+AVFISCOPTER-TAXASSUR-PTAXA))
- max(0,IPCAPTAXT+PPCAP -min(IPCAPTAXT+PPCAP,max(0,INE-IRB+AVFISCOPTER-TAXASSUR-PTAXA))
+ min(0,TAXANEG)))
+ (TAXLOY+PTAXLOY-min(TAXLOY+PTAXLOY,max(0,INE-IRB+AVFISCOPTER-TAXASSUR-PTAXA-IPCAPTAXT-PPCAP))
- max(0, TAXLOY+PTAXLOY-min(TAXLOY+PTAXLOY,max(0,INE-IRB+AVFISCOPTER-TAXASSUR-PTAXA-IPCAPTAXT
- PPCAP))+min(0,PCAPNEG)))
+ (IHAUTREV+PHAUTREV +CHRVPIMP- max(0,IHAUTREV+PHAUTREV +CHRVPIMP+ min(0,LOYELEVNEG))));
```
M, the tax computation domain-specific language created by DGFiP.
M, the tax computation domain-specific language created by DGFiP.

How are we supposed to execute the published code?
M, the tax computation domain-specific language created by DGFiP.

How are we supposed to execute the published code?

- No official, publicly available documentation (no grammar, no semantics, ...)

M, the tax computation domain-specific language created by DGFiP.

How are we supposed to execute the published code?

- No official, publicly available documentation (no grammar, no semantics, ...)
- Python-based primitive interpreter written in 2016 by another State agency
M, the tax computation domain-specific language created by DGFiP.

How are we supposed to execute the published code?

- No official, publicly available documentation (no grammar, no semantics, ...)
- Python-based primitive interpreter written in 2016 by another State agency
- But the interpreter yields wrong values compared to the official simulator
M, the tax computation domain-specific language created by DGFiP.

How are we supposed to execute the published code?

- No official, publicly available documentation (no grammar, no semantics, …)
- Python-based primitive interpreter written in 2016 by another State agency
- But the interpreter yields wrong values compared to the official simulator

We started our open-source compiler in March 2019.
Outline

1. Introduction
2. M, the tip of the iceberg
3. Below the surface: extracting M++
4. MLANG, a compiler for the French Tax Code
5. Conclusion
M, the tip of the iceberg
The core of M: arithmetic expressions assigned to variables.
The core of M: *arithmetic expressions* assigned to variables.

**M quirks**

- Assignments order determined at compilation time using a topological sort

<table>
<thead>
<tr>
<th>M quirks</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Assignments order determined at compilation time using a topological sort</td>
</tr>
</tbody>
</table>
M, briefly

The core of M: **arithmetic expressions** assigned to variables.

**M quirks**

- Assignments order determined at compilation time using a topological sort
- Only floating-point numbers, booleans are zero and one
The core of M: **arithmetic expressions** assigned to variables.

### M quirks

- Assignments order determined at compilation time using a topological sort
- Only floating-point numbers, booleans are zero and one
- Static-size arrays (size defined at declaration)
The core of M: **arithmetic expressions** assigned to variables.

**M quirks**

- Assignments order determined at compilation time using a topological sort
- Only floating-point numbers, booleans are zero and one
- Static-size arrays (size defined at declaration)
- Loops indexed by characters that are substituted in variable names; unrollable
The core of M: arithmetic expressions assigned to variables.

M quirks

➤ Assignments order determined at compilation time using a topological sort
➤ Only floating-point numbers, booleans are zero and one
➤ Static-size arrays (size defined at declaration)
➤ Loops indexed by characters that are substituted in variable names; unrollable
➤ `undef` value
Variable declaration

IRNETBIS : calculee primrest = 0 : "IRNET avant bidouille du 8ZI" ;
8ZI : "Impot net apres depart a l'etranger (non residents)" ;
Example

**Variable declaration**

IRNETBIS : calculee primrest = 0 : "IRNET avant bidouille du 8ZI" ;
8ZI : "Impot net apres depart a l'tranger (non residents)" ;

**Computation rule**

rule 221220:
application : iliad ;
IRNETBIS = max(0, IRNETTER -
    PIR * positif(SEUIL_12 - IRNETTER + PIR)
    * positif(SEUIL_12 - PIR)
    * positif_ou_nul(IRNETTER - SEUIL_12));
A formal semantics for $M$

We reverse-engineered the semantics:

- At first, using the online simulator$^1$
- Later, using the private tests DGFiP sent us (August 7, 2019)

Fun facts:

- $f + \text{undef} = f$
- $f ÷ 0 = 0$
- $x[|x| + 1] = \text{undef}$
- $x[−1] = 0$

A formal semantics for M

We reverse-engineered the semantics:

- At first, using the online simulator\(^1\)
- Later, using the private tests DGFiP sent us (August 7, 2019)

\(\Rightarrow\) a \(\mu\)M kernel, its semantics formalized in the Coq proof assistant.

\(^1\)https://www3.impots.gouv.fr/simulateur/calcul_impot/2020/index.htm
A formal semantics for M

We reverse-engineered the semantics:

- At first, using the online simulator\(^1\)
- Later, using the private tests DGFiP sent us (August 7, 2019)

⇒ a \(\mu\)M kernel, its semantics formalized in the Coq proof assistant.

**The undefined value**

- Used for: default inputs, runtime errors & missing cases in inline conditionals
- Fun facts: \(f + \text{undef} = f, f \div 0 = 0, x[|x| + 1] = \text{undef}, x[-1] = 0...\)

\(^1\)https://www3.impots.gouv.fr/simulateur/calcul_impot/2020/index.htm
Test failures

**August 2019:** only 20% of DGFiP tests passed...
Is this the end?

Test failures

**August 2019**: only 20% of DGFiP tests passed...
After investigation, we knew that some code was missing.
Is this the end?

<table>
<thead>
<tr>
<th>Test failures</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>August 2019</strong>: only 20% of DGFiP tests passed...</td>
</tr>
<tr>
<td>After investigation, we knew that some code was missing.</td>
</tr>
</tbody>
</table>

Patience is the key

<table>
<thead>
<tr>
<th>Date</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aug. 2019</td>
<td>Sent official technical questions to DGFiP</td>
</tr>
<tr>
<td>Jan. 2020</td>
<td>Meeting with DGFiP (5 levels of hierarchy involved!)</td>
</tr>
<tr>
<td>Apr. 2020</td>
<td>Agreement signed</td>
</tr>
<tr>
<td>Jun. 2020</td>
<td>First access to the unpublished sources!</td>
</tr>
</tbody>
</table>
Below the surface: extracting M++
DGFiP’s legacy architecture

Discovered in June 2020:

DGFiP’s internal compiler

GCC
DGFiP’s legacy architecture

Discovered in June 2020:

- “rules” M files
- “rules” C files
- Shared state
- “inter” C files
- “calculette” Shared library

DGFiP’s internal compiler

GCC

“inter” files: 35k lines of C code written to compensate M’s lack of functions.
DGFiP’s legacy architecture

Discovered in June 2020:

“rules” M files → “rules” C files

Shared state

“inter” C files

“calculette” Shared library

DGFiP’s internal compiler

GCC

“inter” files: 35k lines of C code written to compensate M’s lack of functions.

Security concerns meant no publication possible. How to extract the logic of the code, without publishing the “inter” code itself?
Introducing a new DSL: M++

Tax benefits computation in M++

```
compute_benefits():
    if exists_deposit_defined_variables() or exists_taxbenefit_ceiled_variables():
        partition with var_is_taxbenefit:
            V_INDTEO = 1
            V_CALCUL_NAPS = 1
            NAPSANSPENA, IAD11, INE, IRE, PREM8_11 <- call_m()
            V_CALCUL_NAPS = 0
            iad11 = cast(IAD11)
            ire = cast(IRE)
            ine = cast(INE)
            prem = cast(PREM8_11)
            PREM8_11 = prem
            V_IAD11TEO = iad11
            V_IRETETO = ire
            V_INETETO = ine
```
Introducing a new DSL: M++

Tax benefits computation in M++

```plaintext
compute_benefits():
    if exists_deposit_defined_variables() or exists_taxbenefit_ceiled_variables():
        partition with var_is_taxbenefit:
            V_INDTEO = 1
            V_CALCUL_NAPS = 1
            NAPSANSPENA, IAD11, INE, IRE, PREM8_11 <- call_m()
            V_CALCUL_NAPS = 0
            iad11 = cast(IAD11)
            ire = cast(IRE)
            ine = cast(INE)
            prem = cast(PREM8_11)
            PREM8_11 = prem
            V_IAD11TEO = iad11
            V_IRETEO = ire
            V_INETEO = ine
```

- High-level, no mutable state under the hood
- Tailored for the needs of the “inter” files and DGFiP devs
- 6,000 lines of “inter” C code ⇒ 100 lines of M++
MLANG, a compiler for the French Tax Code
MLANG’s architecture

MLANG: written in OCaml, 10k lines of code
https://github.com/MLanguage/mlang
MLANG’s architecture

MLANG: written in OCaml, 10k lines of code
https://github.com/MLanguage/mlang

resources.m

source.mpp

M AST

M++ AST

M IR

M++ IR

BIR

OIR

assumptions.m_spec

Python

Interpreter

C

Parsing

Desugaring

Inlining

Optimization

Transpiling
MLANG’s correctness

How to check that MLANG is correct?

476 tests from DGFiP

- private
- quality?

Let’s generate our own tests

- Randomized tests
- Fuzzer-based tests

Measuring tests quality

Instrument the interpreter to measure coverage
It works (precise down to the euro)!

- All backends validated
- On DGFiP’s tests for 2018 and 2019 and our tests
Code optimization

Compiler optimizations

- Global value numbering
- Dead code elimination
- Partial evaluation
- Dataflow defined-ness analysis

<table>
<thead>
<tr>
<th>Spec. name</th>
<th># inputs</th>
<th># outputs</th>
<th># instructions</th>
<th>% reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>2,459</td>
<td>10,411</td>
<td>129,683</td>
<td>80.2%</td>
</tr>
<tr>
<td>Selected outs</td>
<td>2,459</td>
<td>11</td>
<td>99,922</td>
<td>84.8%</td>
</tr>
<tr>
<td>Tests</td>
<td>1,635</td>
<td>646</td>
<td>111,839</td>
<td>83.0%</td>
</tr>
<tr>
<td>Simplified</td>
<td>228</td>
<td>11</td>
<td>4,172</td>
<td>99.4%</td>
</tr>
<tr>
<td>Basic</td>
<td>3</td>
<td>1</td>
<td>553</td>
<td>99.9%</td>
</tr>
</tbody>
</table>

# of instructions with optimizations disabled (2018 code): 656,020.
# Code optimization

## Compiler optimizations

- Global value numbering
- Dead code elimination
- Partial evaluation
- Dataflow defined-ness analysis

<table>
<thead>
<tr>
<th>Spec. name</th>
<th># inputs</th>
<th># outputs</th>
<th># instructions</th>
<th>% reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>2,459</td>
<td>10,411</td>
<td>129,683</td>
<td>80.2%</td>
</tr>
<tr>
<td>Selected outs</td>
<td>2,459</td>
<td>11</td>
<td>99,922</td>
<td>84.8%</td>
</tr>
<tr>
<td>Tests</td>
<td>1,635</td>
<td>646</td>
<td>111,839</td>
<td>83.0%</td>
</tr>
<tr>
<td>Simplified</td>
<td>228</td>
<td>11</td>
<td>4,172</td>
<td>99.4%</td>
</tr>
<tr>
<td>Basic</td>
<td>3</td>
<td>1</td>
<td>553</td>
<td>99.9%</td>
</tr>
</tbody>
</table>

# of instructions with optimizations disabled (2018 code): 656,020.
## Code optimization

### Compiler optimizations

- Global value numbering
- Dead code elimination
- Partial evaluation
- Dataflow defined-ness analysis

<table>
<thead>
<tr>
<th>Spec. name</th>
<th># inputs</th>
<th># outputs</th>
<th># instructions</th>
<th>% reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>2,459</td>
<td>10,411</td>
<td>129,683</td>
<td>80.2%</td>
</tr>
<tr>
<td>Selected outs</td>
<td>2,459</td>
<td>11</td>
<td>99,922</td>
<td>84.8%</td>
</tr>
<tr>
<td>Tests</td>
<td>1,635</td>
<td>646</td>
<td>111,839</td>
<td>83.0%</td>
</tr>
<tr>
<td>Simplified</td>
<td>228</td>
<td>11</td>
<td>4,172</td>
<td>99.4%</td>
</tr>
<tr>
<td>Basic</td>
<td>3</td>
<td>1</td>
<td>553</td>
<td>99.9%</td>
</tr>
</tbody>
</table>

# of instructions with optimizations disabled (2018 code): 656,020.
Conclusion
### Conclusion

<table>
<thead>
<tr>
<th>Component</th>
<th>Published</th>
<th>Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>M code</td>
<td>yes</td>
<td>—</td>
</tr>
<tr>
<td>&quot;inter&quot; code</td>
<td>no</td>
<td>Hand-written M++ file</td>
</tr>
<tr>
<td>M compiler</td>
<td>no</td>
<td>MLANG</td>
</tr>
<tr>
<td>Test files</td>
<td>no</td>
<td>Fuzzer-based tests</td>
</tr>
</tbody>
</table>

⇒ Income tax computation now reproducible outside DGFiP!
## Conclusion

<table>
<thead>
<tr>
<th>Component</th>
<th>Published</th>
<th>Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>M code</td>
<td>yes</td>
<td>—</td>
</tr>
<tr>
<td>“inter” code</td>
<td>no</td>
<td>Hand-written M++ file</td>
</tr>
<tr>
<td>M compiler</td>
<td>no</td>
<td>MLANG</td>
</tr>
<tr>
<td>Test files</td>
<td>no</td>
<td>Fuzzer-based tests</td>
</tr>
</tbody>
</table>

⇒ Income tax computation now reproducible outside DGFiP!

### From research to production

DGFiP is moving to MLANG. Transition started in June 2021.
Looking ahead

- Income tax studies based on the official code now possible
- Semantic analyses of the income tax code
- A success story, encouraging further opening of critical “state software”
- Deriving correct-by-construction implementations from the law
Looking ahead

- Income tax studies based on the official code now possible
- Semantic analyses of the income tax code
- A success story, encouraging further opening of critical “state software”
- Deriving correct-by-construction implementations from the law

Thanks to DGFiP and in particular bureau SI-1E for the collaboration!
How to maintain the implementation of the law?

- The structure of the law is lost during the translation.
- Law texts are complicated, and have a specific structure.
- High number of changes in some cases (tax code).

**CATALA**\(^2\) to the rescue

- Literate programming
- Relies on default logic
- Fosters cooperation between lawyers and developers
- Formal semantics

A Modern Compiler for the French Tax Code

Questions

Raphaël Monat, with Denis Merigoux and Jonathan Protzenko