A Modern Compiler for the French Tax Code

Denis Merigoux & Raphaël Monat & Jonathan Protzenko

Prosecco, Inria & LIP6, Sorbonne Université & Microsoft Research

Compiler Construction, March 3\textsuperscript{rd}, 2021
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
Income Tax in France

Each year

- 38M fiscal households
Income Tax in France

Each year

- 38M fiscal households
- Tax computation performed by the State (DGFiP)
Income Tax in France

Each year

- 38M fiscal households
- Tax computation performed by the State (DGFiP)
- €75B
Income Tax in France

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

Each year

- 38M fiscal households
- Tax computation performed by the State (DGFiP)
- €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 Code

- Made public in April 2016
Income Tax Code

- 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

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

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

\[
\text{IRNETTER} = \max(0, \\
\begin{array}{l}
\text{IRNET2} \\
+ (\text{TAXASSUR + PTAXA} - \min(\text{TAXASSUR + PTAXA} + 0, \max(0, \text{INE-IRB + AVFISCOPTER}))) \\
- \max(0, \text{TAXASSUR + PTAXA} - \min(\text{TAXASSUR + PTAXA} + 0, \max(0, \text{INE-IRB + AVFISCOPTER}))) + \min(0, \text{IRNET2}))
\end{array} \\
+ (\text{IPCAPTAXT + PPCAP} - \min(\text{IPCAPTAXT + PPCAP} + 0, \max(0, \text{INE-IRB + AVFISCOPTER} - \text{TAXASSUR - PTAXA}))) \\
- \max(0, \text{IPCAPTAXT + PPCAP} - \min(\text{IPCAPTAXT + PPCAP} + 0, \max(0, \text{INE-IRB + AVFISCOPTER} - \text{TAXASSUR - PTAXA}))) + \min(0, \text{TAXANEG})))
\end{array} \\
+ (\text{TAXLOY + PTAXLOY} - \min(\text{TAXLOY + PTAXLOY} + 0, \max(0, \text{INE-IRB + AVFISCOPTER} - \text{TAXASSUR - PTAXA}))) \\
- \max(0, \text{TAXLOY + PTAXLOY} - \min(\text{TAXLOY + PTAXLOY} + 0, \max(0, \text{INE-IRB + AVFISCOPTER} - \text{TAXASSUR - PTAXA}))) + \min(0, \text{PCAPNEG})))
\end{array} \\
+ (\text{IHAUTREVT + PHAUTREV + CHRPVIMP} - \max(0, \text{IHAUTREVT + PHAUTREV + CHRPVIMP} + \min(0, \text{LOYELEVNEG})))
\]
Income Tax Code

$M$, 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, . . .)
Income Tax 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, ...)
- 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

March 2019: we started our open-source compiler for the income tax computation.
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
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
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)
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
- Static-size arrays (size defined at declaration)
- Loops indexed by characters that are substituted in variable names; unrollable
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
- Static-size arrays (size defined at declaration)
- Loops indexed by characters that are substituted in variable names; unrollable
- undef value
Variable declaration

IRNETBIS : computed primrest = 0 : "net income tax before 8ZI hack";
8ZI: net tax after living abroad (non-residents);
Example

Variable declaration

IRNETBIS : computed primrest = 0 : "net income tax before 8ZI hack";
8ZI: net tax after living abroad (non-residents);

Computation rule

rule 221220:
application : iliad ;
IRNETBIS = max(0, IRNETTER -
    PIR * positive(THRESHOLD_12 - IRNETTER + PIR)
    * positive(THRESHOLD_12 - PIR)
    * positive_or_zero(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)

---

\(^1\)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)

\[\Rightarrow\] a $\mu$M kernel, its semantics formalized in Coq.

---

\(^1\)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 Coq.

The undefined value

- Used for: default inputs, runtime errors & missing cases in inline conditionals
- Fun facts: $f + \text{undef} = f$, $f ÷ 0 = 0$, $x[|x| + 1] = \text{undef}$, $x[-1] = 0$...
- Full semantics in the paper

\(^1\)https://www3.impots.gouv.fr/simulateur/calcul_impot/2020/index.htm
Is this the end?

Test failures

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

Test failures

**August 2019:** only 20% of DGFiP tests passed…
After investigation, we knew that some code was missing.

Patience is the key

- **Aug. 2019**  Sent official technical questions to DGFiP
- **Jan. 2020**  Meeting with DGFiP (5 levels of hierarchy involved!)
- **Apr. 2020**  Agreement signed
- **Jun. 2020**  First access to the unpublished sources!
Introduction

M, the tip of the iceberg

Below the surface: extracting M++

Mlang, a compiler for the French Tax Code

Conclusion
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

Security concerns meant no publication possible. How to extract the logic of the code, without publishing the "inter" code itself?
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?
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
Introducing a new DSL: 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_CRETEO = 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++
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
**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

sources.m → M AST → M IR → BIR
source.mpp → M++ AST → M++ IR → OIR

Assumptions: assumptions.m_spec
Python → Interpreter
C

Steps:
- Parsing
- Desugaring
- Inlining
- Optimization
- Transpiling
Mlang’s correctness

It works (precise down to the euro)!

- All backends validated on DGFiP’s tests for 2018 and 2019
- Generate more test cases via random pick or fuzzing (AFL)

![Chart showing the number of distinct values assigned and the percentage of assignments for different classes of assignments: 0 (uncovered), 1, 2 or more. The chart compares DGFiP Private (476 tests), Randomized (1267 tests), and Fuzzer-generated (275 tests).]
## Code optimization

| Compiler optimizations | | | |
|------------------------|--|---|---|---|---|
| Global value numbering | Partial evaluation |
| Dead code elimination | 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><strong>All</strong></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.
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
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!
## 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>$\text{M}_{\text{LANG}}$</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 $\text{M}_{\text{LANG}}$. Transition planned for 2021-2022.
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

Paper, source code, etc: rmonat.fr/cc21
Thanks to DGFiP and in particular bureau SI-1E for the collaboration!