Mlang: an Open-Source Toolchain for the Income Tax Computation

Denis Merigoux & Raphaël Monat

Prosecco, Inria & LIP6, Sorbonne Université

JFLA, April 8th 2021

Income Tax in France

Each year

- Tax computation by DGFiP

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

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

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`

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.

```
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

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

Income Tax Code

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

How are we supposed to execute the published code?

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, ...)

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

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
- But the interpreter yields wrong values compared to the official simulator

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
- But the interpreter yields wrong values compared to the official simulator

March 2019: we started our open-source compiler for the income tax computation.

M, briefly

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

M, briefly

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

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)

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

Example

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'etranger (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)

[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 μM 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)

$\Rightarrow$ a μM 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: $f + \texttt{undef} = f$, $f \div 0 = 0$, $x[|x| + 1] = \texttt{undef}$, $x[-1] = 0$...

[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...

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?

Test failures

August 2019: only 20% of DGFiP tests passed...

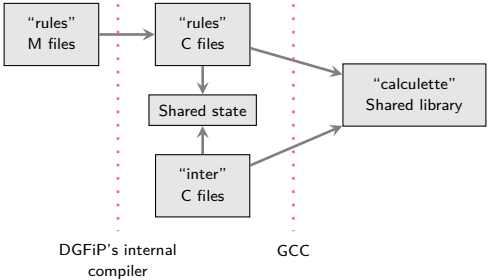
After investigation, we knew that some code was missing.

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!

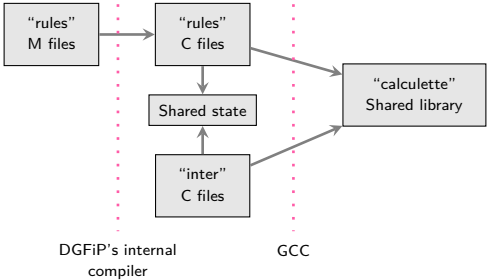
DGFiP's legacy architecture

Discovered in June 2020:



DGFiP's legacy architecture

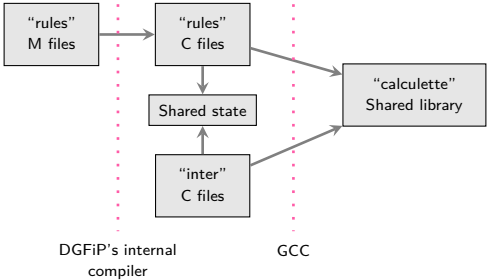
Discovered in June 2020:



"inter" files : 35k lines of C code written to compensate M's lack of functions.

DGFiP's legacy architecture

Discovered in June 2020:



"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++

```
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++

```
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 $\Rightarrow$ 100 lines of M++

Mlang's architecture

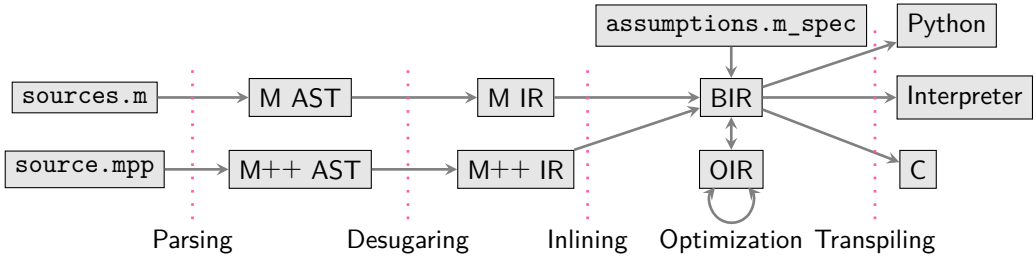
MLANG: written in OCaml, 10k lines of code

`https://github.com/MLanguage/mlang`

Mlang's architecture

MLANG: written in OCaml, 10k lines of code

`https://github.com/MLanguage/mlang`



Demo!

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

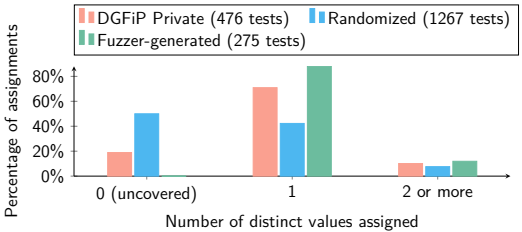
Measuring tests quality

Instrument the interpreter to measure coverage

Mlang's correctness

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

Code optimization

Compiler optimizations

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

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

# 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

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

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

Conclusion

Component	Published	Replacement
M code	yes	–
“inter” code	no	Hand-written M++ file
M compiler	no	MLANG
Test files	no	Fuzzer-based tests

⇒ Income tax computation now reproducible outside DGFiP!

Conclusion

Component	Published	Replacement
M code	yes	–
“inter” code	no	Hand-written M++ file
M compiler	no	MLANG
Test files	no	Fuzzer-based tests

⇒ Income tax computation now reproducible outside DGFiP!

From research to production

DGFiP is moving to MLANG. 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, . . . rmonat.fr/jfla21

Technical paper at Compiler Construction 2021.

Thanks to DGFiP and in particular bureau SI-1E for the collaboration!