Formal methods meet legal implementations

Raphaël Monat

rmonat.fr

GT LVP, ENS Paris-Saclay
23 November 2026



Legal implementations?

Software implementations running laws precisely describing computations

Legal implementations?

Software implementations running laws precisely describing computations

- Income tax

Legal implementations?

Software implementations running laws precisely describing computations

- Income tax
- Social benefits

Legal implementations?

Software implementations running laws precisely describing computations

- Income tax
- Social benefits
- Payroll systems

Legal implementations?

Software implementations running laws precisely describing computations

- Income tax
- Social benefits
- Payroll systems
- ~~AI judges~~

Legal implementations are critical software

Critical software with tremendous impact on citizens

Legal implementations are critical software

Critical software with tremendous impact on citizens

- Phoenix, Canadian payroll system for civil servants [Moc18]

Legal implementations are critical software

Critical software with tremendous impact on citizens

- Phoenix, Canadian payroll system for civil servants [Moc18]
  - Goal: “$70 million in annual savings”

Legal implementations are critical software

Critical software with tremendous impact on citizens

- Phoenix, Canadian payroll system for civil servants [Moc18]
  - Goal: “$70 million in annual savings”
  - Majority of civil servants suffered pay issues

Legal implementations are critical software

Critical software with tremendous impact on citizens

- Phoenix, Canadian payroll system for civil servants [Moc18]
  - Goal: “$70 million in annual savings”
  - Majority of civil servants suffered pay issues
  - “$2.2 billion in unplanned expenditures”

Legal implementations are critical software

Critical software with tremendous impact on citizens

- Phoenix, Canadian payroll system for civil servants [Moc18]
  - Goal: “$70 million in annual savings”
  - Majority of civil servants suffered pay issues
  - “$2.2 billion in unplanned expenditures”
- Dutch childcare benefits scandal

Legal implementations are critical software

Critical software with tremendous impact on citizens

- Phoenix, Canadian payroll system for civil servants [Moc18]
  - Goal: “$70 million in annual savings”
  - Majority of civil servants suffered pay issues
  - “$2.2 billion in unplanned expenditures”
- Dutch childcare benefits scandal
  - Wrong accusations of frauds on 10,000s households, reimbursements asked

Legal implementations are critical software

Critical software with tremendous impact on citizens

- Phoenix, Canadian payroll system for civil servants [Moc18]
  - Goal: “$70 million in annual savings”
  - Majority of civil servants suffered pay issues
  - “$2.2 billion in unplanned expenditures”
- Dutch childcare benefits scandal
  - Wrong accusations of frauds on 10,000s households, reimbursements asked
  - “unlawful, discriminatory and improper” – Data Protection Authority [Dut20]

Legal implementations are critical software

Critical software with tremendous impact on citizens

- Phoenix, Canadian payroll system for civil servants [Moc18]
  - Goal: “$70 million in annual savings”
  - Majority of civil servants suffered pay issues
  - “$2.2 billion in unplanned expenditures”
- Dutch childcare benefits scandal
  - Wrong accusations of frauds on 10,000s households, reimbursements asked
  - “unlawful, discriminatory and improper” – Data Protection Authority [Dut20]
  - Culminated with gov. resignation in 2021.

Legal implementations are critical software

Critical software with tremendous impact on citizens

- Phoenix, Canadian payroll system for civil servants [Moc18]
  - Goal: “$70 million in annual savings”
  - Majority of civil servants suffered pay issues
  - “$2.2 billion in unplanned expenditures”
- Dutch childcare benefits scandal
  - Wrong accusations of frauds on 10,000s households, reimbursements asked
  - “unlawful, discriminatory and improper” – Data Protection Authority [Dut20]
  - Culminated with gov. resignation in 2021.

Need more?

See “Automating public services: learning from cancelled systems” [Red+22].

Legal implementations require interdisciplinary collaboration

Law != Computer Science: interdisciplinary collaboration is required.

Legal implementations require interdisciplinary collaboration

Law != Computer Science: interdisciplinary collaboration is required.

- "epistemic trespassing" risk Ballantyne [Bal19] and Lawsky [Law24]
  "Epistemic trespassers are thinkers who have [...] expertise to make good judgments in one field, but move into another field where they lack competence – and pass judgment nevertheless"

Legal implementations require interdisciplinary collaboration

Law != Computer Science: interdisciplinary collaboration is required.

- "epistemic trespassing" risk Ballantyne [Bal19] and Lawsky [Law24]
  "Epistemic trespassers are thinkers who have [...] expertise to make good judgments in one field, but move into another field where they lack competence – and pass judgment nevertheless"
- Experiment where teams implemented a bankruptcy law Escher et al. [Esc+24]

Legal implementations require interdisciplinary collaboration

Law != Computer Science: interdisciplinary collaboration is required.

- "epistemic trespassing" risk Ballantyne [Bal19] and Lawsky [Law24]
  "Epistemic trespassers are thinkers who have [...] expertise to make good judgments in one field, but move into another field where they lack competence – and pass judgment nevertheless"
- Experiment where teams implemented a bankruptcy law Escher et al. [Esc+24]
  - CS or CS+Law teams

Legal implementations require interdisciplinary collaboration

Law != Computer Science: interdisciplinary collaboration is required.

- "epistemic trespassing" risk Ballantyne [Bal19] and Lawsky [Law24]
  "Epistemic trespassers are thinkers who have [...] expertise to make good judgments in one field, but move into another field where they lack competence – and pass judgment nevertheless"
- Experiment where teams implemented a bankruptcy law Escher et al. [Esc+24]
  - CS or CS+Law teams
  - "need for greater skepticism about the [...] reliability of legal software"

Legal implementations require interdisciplinary collaboration

Law != Computer Science: interdisciplinary collaboration is required.

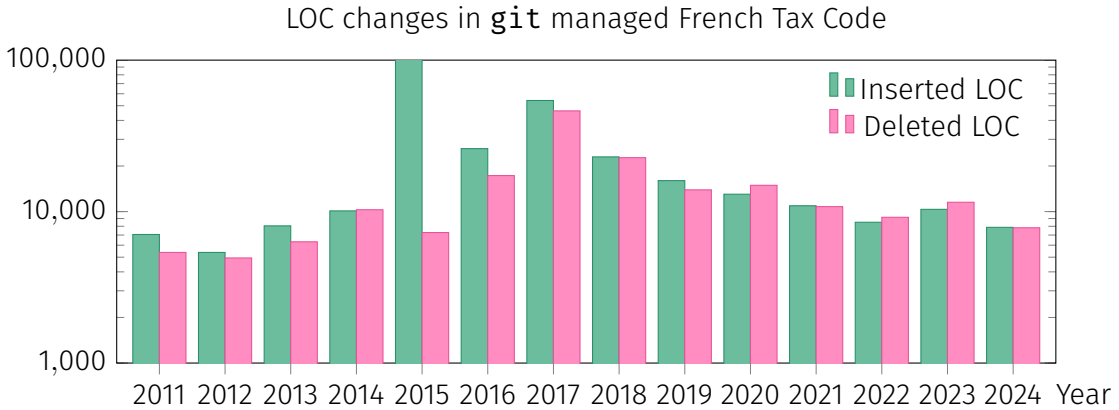
- "epistemic trespassing" risk Ballantyne [Bal19] and Lawsky [Law24]
  "Epistemic trespassers are thinkers who have [...] expertise to make good judgments in one field, but move into another field where they lack competence – and pass judgment nevertheless"
- Experiment where teams implemented a bankruptcy law Escher et al. [Esc+24]
  - CS or CS+Law teams
  - "need for greater skepticism about the [...] reliability of legal software"
  - "participants (esp. CS) expressed unwarranted confidence in [tool utility]"

Legal implementations are updated regularly and incrementally

$\simeq$ yearly legislative changes needing implementations

Legal implementations are updated regularly and incrementally

≃ yearly legislative changes needing implementations



Legal implementations have a long lifespan

⚠ long history not covered here

Legal implementations have a long lifespan

⚠ long history not covered here

French Income Tax

- 1972 First COBOL implementations
- 1987 Transition to codebase still in use

Legal implementations have a long lifespan

⚠ long history not covered here

French Income Tax

- 1972 First COBOL implementations
- 1987 Transition to codebase still in use

French Benefits

- 1996 Centralization of multiple benefits
- Modernization efforts, types of benefits has tripled

Towards high-assurance legal implementations

Trusting the process?

Towards high-assurance legal implementations

Trusting the process?

- Reproducibility of the computation?

Towards high-assurance legal implementations

Trusting the process?

- Reproducibility of the computation?
- Compliance with the law, acting as specification?

Towards high-assurance legal implementations

Trusting the process?

- Reproducibility of the computation?
- Compliance with the law, acting as specification?
- Usability by public administrations, citizens, NGOs, researchers?

Towards high-assurance legal implementations

Trusting the process?

- Reproducibility of the computation?
- Compliance with the law, acting as specification?
- Usability by public administrations, citizens, NGOs, researchers?
- Accurate simulations?

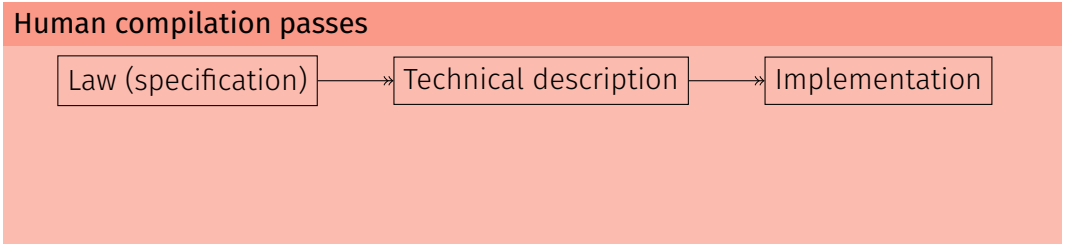
Outline

1. Introduction
2. Catala, a DSL for the Law
3. Automated Verification of Catala Programs
   - Date Arithmetic [MFM24]
   - CuteCat Concolic Execution [GFM25]
4. Conclusion

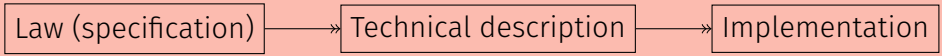
Joint work

- Marie Alauzen
- Justine Banuls
- Vincent Botbol
- Nicolas Chataing
- Caroline Flori
- Aymeric Fromherz
- Louis Gesbert
- Pierre Goutagny (slides ack)
- Estelle Hary
- Liane Huttner
- Sarah Lawsky
- Denis Merigoux (now transfer lead)
- Romain Primet
- Jonathan Protzenko
- Émile Rolley
- Lilya Slimani

Catala, a DSL for the Law

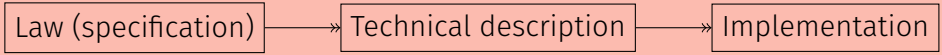

Human compilation passes
Law (specification)
Technical description
Implementation

Human compilation passes



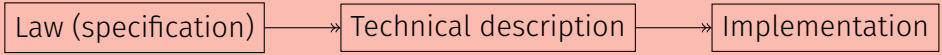
- Structural correspondence is lost in the intermediate representations

Human compilation passes



- Structural correspondence is lost in the intermediate representations
- Hidden knowledge, bus factor, difficult to audit, ...

Human compilation passes



- Structural correspondence is lost in the intermediate representations
- Hidden knowledge, bus factor, difficult to audit, ...

Catala, a new DSL

Initiated through interdisciplinary work, and requiring peer programming

- Merigoux, Chataing, and Protzenko [MCP21]
- Huttner and Merigoux [HM22]
- Merigoux, Alauzen, and Slimani [MAS23]

Structure of computational law: income tax example

Article 1

The income tax is a fixed percentage of the income.

Structure of computational law: income tax example

Article 1

The income tax is a fixed percentage of the income.

Article 2

The fixed percentage mentioned at article 1 is 20%.

Structure of computational law: income tax example

Article 1

The income tax is a fixed percentage of the income.

Article 2 default case

The fixed percentage mentioned at article 1 is 20%.

Structure of computational law: income tax example

Article 1

The income tax is a fixed percentage of the income.

Article 2

default case

The fixed percentage mentioned at article 1 is 20%.

Article 3

If the income is less than $10,000, the percentage mentioned at article 1 is 10%.

Article 4

For people in charge of 3 or more children, the percentage mentioned at article 1 is 15%.

Structure of computational law: income tax example

Article 1

The income tax is a fixed percentage of the income.

Article 2 default case

The fixed percentage mentioned at article 1 is 20%.

Article 3 exception

If the income is less than $10,000, the percentage mentioned at article 1 is 10%.

Article 4 exception

For people in charge of 3 or more children, the percentage mentioned at article 1 is 15%.

Default logic

Running Example on Income Tax

```
# Article 1
The income tax is a fixed percentage of
the income.
```catala
scope IncomeTaxComputation:
 definition income_tax equals
   house.income * tax_rate
```

Article 2
The fixed percentage mentioned at
article 1 is 20%.
```catala
scope IncomeTaxComputation:
  definition tax_rate equals 20%
```
```

```
# Article 3
If the income is less than $10,000, the
percentage mentioned at article 1 is 10%.
```catala
scope IncomeTaxComputation:
 exception definition tax_rate
  under condition house.income <= $10,000
  consequence equals 10%
```

Article 4
For people in charge of 3 or more
children, the percentage mentioned at
article 1 is 15%.
```catala
scope IncomeTaxComputation:
 exception definition tax_rate
  under condition house.nb_children >= 3
  consequence equals 15%
```
```

https://book.catala-lang.org/playground/learn.html?chapter=2-1-basic-blocks

Running Example on Income Tax

```
# Article 1
The income tax is a fixed percentage of
the income.
```catala
scope IncomeTaxComputation:
 definition income_tax equals
   house.income * tax_rate
```

Article 2
The fixed percentage mentioned at
article 1 is 20%.
```catala
scope IncomeTaxComputation:
  definition tax_rate equals 20%
```
```

```
# Article 3
If the income is less than $10,000, the
percentage mentioned at article 1 is 10%.
```catala
scope IncomeTaxComputation:
 exception definition tax_rate
  under condition house.income <= $10,000
  consequence equals 10%
```

Article 4
For people in charge of 3 or more
children, the percentage mentioned at
article 1 is 15%.
```catala
scope IncomeTaxComputation:
 exception definition tax_rate
  under condition house.nb_children >= 3
  consequence equals 15%
```
```

https://book.catala-lang.org/playground/learn.html?chapter=2-1-basic-blocks

Running Example on Income Tax

```
# Article 1
The income tax is a fixed percentage of
the income.
```catala
scope IncomeTaxComputation:
 definition income_tax equals
   house.income * tax_rate
```

Article 2
The fixed percentage mentioned at
article 1 is 20%.
```catala
scope IncomeTaxComputation:
  definition tax_rate equals 20%
```

Article 3
If the income is less than $10,000, the
percentage mentioned at article 1 is 10%.
```catala
scope IncomeTaxComputation:
 exception definition tax_rate
  under condition house.income <= $10,000
  consequence equals 10%
```

Article 4
For people in charge of 3 or more
children, the percentage mentioned at
article 1 is 15%.
```catala
scope IncomeTaxComputation:
 exception definition tax_rate
  under condition house.nb_children >= 3
  consequence equals 15%
```
```

https://book.catala-lang.org/playground/learn.html?chapter=2-1-basic-blocks

Compilation into Default Terms

Source code $\xrightarrow{\text{compiler}}$ **default terms**

$$e ::= \langle\, e_1, \ldots, e_n \;\mid\; b_{default} \;:-\; e_{default} \,\rangle$$

Compilation into Default Terms

Source code $\xrightarrow{\text{compiler}}$ **default terms**

$$e ::= \langle \underbrace{e_1, \ldots, e_n}_{\text{exceptions}} \mid b_{default} \ :- \ e_{default} \rangle$$

Compilation into Default Terms

Source code $\xrightarrow{\text{compiler}}$ **default terms**

$$e ::= \langle \underbrace{e_1, \ldots, e_n}_{\text{exceptions}} \mid \underbrace{b_{default}}_{\text{guard}} :- e_{default} \rangle$$

Compilation into Default Terms

Source code $\xrightarrow{\text{compiler}}$ **default terms**

$$e ::= \langle \underbrace{e_1, \ldots, e_n}_{\text{exceptions}} \mid \underbrace{b_{default}}_{\text{guard}} :- \underbrace{e_{default}}_{\text{base case}} \rangle$$

Compilation into Default Terms

Source code $\xrightarrow{\text{compiler}}$ **default terms**

$$e ::= \langle \underbrace{e_1, \ldots, e_n}_{\text{exceptions}} \mid \underbrace{b_{default}}_{\text{guard}} :- \underbrace{e_{default}}_{\text{base case}} \rangle$$

$$\begin{array}{rcl} v & ::= & \texttt{true} \mid \texttt{false} \mid n \mid \ldots \\ & \mid & \varnothing \\ & \mid & \circledast \end{array}$$

Default Terms: Semantics

```
⟨
  ⟨ | income ≤ $10,000 :- 10%⟩,
  ⟨ | nb_children ≥ 3 :- 15%⟩
  | true :- 20%
⟩
```

```
exception definition tax_rate
 under condition house.income <= $10,000
 consequence equals 10%
exception definition tax_rate
 under condition house.nb_children >= 3
 consequence equals 15%
definition tax_rate equals 20%
```

Default Terms: Semantics

$$
\langle \\
\quad \langle \mid \texttt{income} \leq \$10,000 \texttt{ :- } 10\% \rangle, \\
\quad \langle \mid \texttt{nb_children} \geq 3 \texttt{ :- } 15\% \rangle \\
\quad \mid \texttt{true :- } 20\% \\
\rangle
$$

```
exception definition tax_rate
 under condition house.income <= $10,000
 consequence equals 10%
exception definition tax_rate
 under condition house.nb_children >= 3
 consequence equals 15%
definition tax_rate equals 20%
```

1. Evaluate all exceptions
2. If **exactly 1 exception** is raised, then **return its value**
3. Else if **at least 2 exceptions** are raised, then **return** $\circledast$
4. Else if **0 exceptions** are raised, evaluate $b_{default}$ and
   - If $b_{default} = \texttt{true}$, then **evaluate** $e_{default}$
   - Else if $b_{default} = \texttt{false}$, then **return** $\varnothing$

Default Terms: Semantics

$$
\begin{array}{l}
\langle \\
\quad \langle \mid \texttt{income} \leq \$10,000 \texttt{ :- } \mathbf{10\%} \rangle, \\
\quad \langle \mid \texttt{nb_children} \geq 3 \texttt{ :- } 15\% \rangle \\
\quad \mid \texttt{true} \texttt{ :- } 20\% \\
\rangle
\end{array}
$$

```
exception definition tax_rate
 under condition house.income <= $10,000
 consequence equals 10%
exception definition tax_rate
 under condition house.nb_children >= 3
 consequence equals 15%
definition tax_rate equals 20%
```

1. Evaluate all exceptions
2. If **exactly 1 exception** is raised, then **return its value**
3. Else if **at least 2 exceptions** are raised, then **return** $\circledast$
4. Else if **0 exceptions** are raised, evaluate $b_{default}$ and
   - If $b_{default} = \texttt{true}$, then **evaluate** $e_{default}$
   - Else if $b_{default} = \texttt{false}$, then **return** $\varnothing$

Default Terms: Semantics

⟨
⟨ | income ≤ $10,000 :- 10%⟩,
⟨ | nb_children ≥ 3 :- **15%**⟩
| true :- 20%
⟩

```
exception definition tax_rate
 under condition house.income <= $10,000
 consequence equals 10%
exception definition tax_rate
 under condition house.nb_children >= 3
 consequence equals 15%
definition tax_rate equals 20%
```

1. Evaluate all exceptions
2. If **exactly 1 exception** is raised, then **return its value**
3. Else if **at least 2 exceptions** are raised, then **return** ⊛
4. Else if **0 exceptions** are raised, evaluate $b_{default}$ and
   - If $b_{default} = \texttt{true}$, then **evaluate** $e_{default}$
   - Else if $b_{default} = \texttt{false}$, then **return** ∅

Default Terms: Semantics

$$
\begin{array}{l}
\langle \\
\quad \langle \mid \texttt{income} \leq \$10,000 \texttt{ :- } 10\% \rangle, \\
\quad \langle \mid \texttt{nb_children} \geq 3 \texttt{ :- } 15\% \rangle \\
\quad \mid \texttt{true} \texttt{ :- } 20\% \\
\rangle
\end{array}
$$

```
exception definition tax_rate
 under condition house.income <= $10,000
 consequence equals 10%
exception definition tax_rate
 under condition house.nb_children >= 3
 consequence equals 15%
definition tax_rate equals 20%
```

1. Evaluate all exceptions
2. If **exactly 1 exception** is raised, then **return its value**
3. Else if **at least 2 exceptions** are raised, then **return** $\circledast$
4. Else if **0 exceptions** are raised, evaluate $b_{default}$ and
   - If $b_{default} = \texttt{true}$, then **evaluate** $e_{default}$
   - Else if $b_{default} = \texttt{false}$, then **return** $\varnothing$

Default Terms: Semantics

⟨
⟨ | income ≤ $10,000 :- 10%⟩,
⟨ | nb_children ≥ 3 :- 15%⟩
| true :- 20%
⟩

```
exception definition tax_rate
 under condition house.income <= $10,000
 consequence equals 10%
exception definition tax_rate
 under condition house.nb_children >= 3
 consequence equals 15%
definition tax_rate equals 20%
```

1. Evaluate all exceptions
2. If **exactly 1 exception** is raised, then **return its value**
3. Else if **at least 2 exceptions** are raised, then **return** ⊛
4. Else if **0 exceptions** are raised, evaluate $b_{default}$ and
   - If $b_{default} =$ `true`, then **evaluate** $e_{default}$
   - Else if $b_{default} =$ `false`, then **return** ∅

Outlook: Public Administration Adoption (Merigoux, Gesbert, Primet, Botbol)

Public administration trials

- DGFIP Income Tax Experiment, Jul. 2023-Oct. 2024, 8.9kLOC
- CNAF Family Benefits, Oct. 2024-Oct. 2025, 8.4kLOC

Inria Press Release About Trials

Outlook: Public Administration Adoption (Merigoux, Gesbert, Primet, Botbol)

Public administration trials

- DGFIP Income Tax Experiment, Jul. 2023-Oct. 2024, 8.9kLOC
- CNAF Family Benefits, Oct. 2024-Oct. 2025, 8.4kLOC

Inria Press Release About Trials

Towards adoption at CNAF

CNAF CIO interview

- Emphasis on sovereignty (compared to proprietary solutions), open-source
- Housing benefits, solidarity income, in-work benefit will be migrated

Automated Verification of Catala Programs

Catala is safe

- Designed by PL specialists
- Statically typed
- Decimals for money
- Not Turing complete

Catala is safe

- Designed by PL specialists
- Statically typed
- Decimals for money
- Not Turing complete

But runtime errors can still abound

- New datatypes: dates

Catala is safe

- Designed by PL specialists
- Statically typed
- Decimals for money
- Not Turing complete

But runtime errors can still abound

- New datatypes: dates
- Division by zero, assertion error, etc.

Catala is safe

- Designed by PL specialists
- Statically typed
- Decimals for money
- Not Turing complete

But runtime errors can still abound

- New datatypes: dates
- Division by zero, assertion error, etc.
- **Ambiguities** in the code

Motivation

Catala is safe

- Designed by PL specialists
- Statically typed
- Decimals for money
- Not Turing complete

But runtime errors can still abound

- New datatypes: dates
- Division by zero, assertion error, etc.
- **Ambiguities** in the code

⇝ RTEs unwanted for large batch computations

Ambiguities in Catala code

```
# Article 3
If the income is less than $10,000, the
percentage mentioned at article 1 is 10%.
```catala
scope IncomeTaxComputation:
 exception definition tax_rate
  under condition house.income <= $10,000
  consequence equals 10%
```
```

```
# Article 4
For people in charge of 3 or more children,
the percentage mentioned at article 1 is 15%.
```catala
scope IncomeTaxComputation:
 exception definition tax_rate
  under condition house.nb_children >= 3
  consequence equals 15%
```
```

Ambiguities in Catala code

- interpretation conflicts, *e.g.* `income = $9,000` and `children = 4`

Ambiguities in Catala code

```
# Article 3
If the income is less than $10,000, the
percentage mentioned at article 1 is 10%.
```catala
scope IncomeTaxComputation:
 exception definition tax_rate
  under condition house.income <= $10,000
  consequence equals 10%
```
```

```
# Article 4
For people in charge of 3 or more children,
the percentage mentioned at article 1 is 15%.
```catala
scope IncomeTaxComputation:
 exception definition tax_rate
  under condition house.nb_children >= 3
  consequence equals 15%
```
```

Ambiguities in Catala code

- interpretation conflicts, *e.g.* `income = $9,000` and `children = 4`
- unhandled cases

Ambiguities in Catala code

```
# Article 3
If the income is less than $10,000, the
percentage mentioned at article 1 is 10%.
```catala
scope IncomeTaxComputation:
 exception definition tax_rate
  under condition house.income <= $10,000
  consequence equals 10%
```
```

```
# Article 4
For people in charge of 3 or more children,
the percentage mentioned at article 1 is 15%.
```catala
scope IncomeTaxComputation:
 exception definition tax_rate
  under condition house.nb_children >= 3
  consequence equals 15%
```
```

Ambiguities in Catala code

- interpretation conflicts, *e.g.* `income = $9,000` and `children = 4`
- unhandled cases
- resolved by lawyers/administration if implementation is correct

Fixing the interpretation conflict

Suppose the lawyer says the `income` condition has priority.

```
⟨
  ⟨ | income ≤ $10,000 :- 10%⟩,
  ⟨ | nb_children ≥ 3 :- 15%⟩
  | true :- 20%
⟩
```

Fixing the interpretation conflict

Suppose the lawyer says the `income` condition has priority.
$\rightarrow$ it becomes an exception to the exception.

```
⟨
  ⟨ | income ≤ $10,000 :- 10%⟩,
  ⟨ | nb_children ≥ 3 :- 15%⟩
  | true :- 20%
⟩
```

```
⟨
  ⟨
    ⟨ | income ≤ $10,000 :- 10%⟩
    | nb_children ≥ 3 :- 15%
  ⟩
  | true :- 20%
⟩
```

Automated Verification of Catala Programs

Date Arithmetic [MFM24]

```
$ date -d "2024-01-31 + 1 month" +%F
```

Motivation

```
$ date -d "2024-01-31 + 1 month" +%F
2024-03-02
```

Motivation

```
$ date -d "2024-01-31 + 1 month" +%F
2024-03-02
$ date -d "2024-02-01 + 1 month" +%F
```

```
$ date -d "2024-01-31 + 1 month" +%F
2024-03-02
$ date -d "2024-02-01 + 1 month" +%F
2024-03-01
```

Motivation

```
$ date -d "2024-01-31 + 1 month" +%F
2024-03-02
$ date -d "2024-02-01 + 1 month" +%F
2024-03-01
```

Non-monotonic behavior?!

A wide variety of choices

Different legal bodies and choices

- 1 month = 30 days (Council of European Communities)

A wide variety of choices

Different legal bodies and choices

- 1 month = 30 days (Council of European Communities)
- When do leapers become adults?

A wide variety of choices

Different legal bodies and choices

- 1 month = 30 days (Council of European Communities)
- When do leapers become adults?
  - 28 February in New Zealand, Taiwan

A wide variety of choices

Different legal bodies and choices

- 1 month = 30 days (Council of European Communities)
- When do leapers become adults?
  - 28 February in New Zealand, Taiwan
  - 1 March in France, Germany, Hong-Kong

A wide variety of choices

Different legal bodies and choices

- 1 month = 30 days (Council of European Communities)
- When do leapers become adults?
  - 28 February in New Zealand, Taiwan
  - 1 March in France, Germany, Hong-Kong

$\Longrightarrow$ Formal, flexible semantics required!

A wide variety of choices

Different legal bodies and choices

- 1 month = 30 days (Council of European Communities)
- When do leapers become adults?
  - 28 February in New Zealand, Taiwan
  - 1 March in France, Germany, Hong-Kong

$\Longrightarrow$ Formal, flexible semantics required! Focus on Gregorian calendar.

Semantics – Rounding

$$(2024, 01, 31) +_m 1 \to (2024, 02, 31)$$

Semantics – Rounding

$$(2024, 01, 31) +_m 1 \to (2024, 02, 31)$$

Rounding to valid dates required!

Semantics – Rounding

$$(2024, 01, 31) +_m 1 \to (2024, 02, 31)$$

Rounding to valid dates required!

$$\begin{array}{lrcl} \text{rounding mode} & r & ::= & \uparrow \mid \downarrow \mid \bot \\ \text{expressions} & e & ::= & v \mid e +_\delta n \mid \mathrm{rnd}_r\, e \end{array}$$

$$(2024, 01, 31) +_m 1 \to (2024, 02, 31)$$

Rounding to valid dates required!

$$\begin{array}{lrcl} \text{rounding mode} & r & ::= & \uparrow \mid \downarrow \mid \bot \\ \text{expressions} & e & ::= & v \mid e +_\delta n \mid \mathrm{rnd}_r\, e \end{array}$$

$$\mathrm{rnd}_\uparrow(2024, 02, 31) = (2024, 03, 01)$$
$$\mathrm{rnd}_\downarrow(2024, 02, 31) = (2024, 02, 29)$$
$$\mathrm{rnd}_\bot(2024, 02, 31) = \bot$$

$$(2024, 01, 31) +_m 1 \to (2024, 02, 31)$$

Rounding to valid dates required!

$$\begin{array}{lrcl} \text{rounding mode} & r & ::= & \uparrow \mid \downarrow \mid \bot \\ \text{expressions} & e & ::= & v \mid e +_\delta n \mid \mathsf{rnd}_r\, e \end{array}$$

$$\mathsf{rnd}_\uparrow(2024, 02, 31) = (2024, 03, 01)$$
$$\mathsf{rnd}_\downarrow(2024, 02, 31) = (2024, 02, 29)$$
$$\mathsf{rnd}_\bot(2024, 02, 31) = \bot$$

Coreutils-like rounding not defined here

Non-properties

Commutativity of addition

$$(2024, 03, 31) +_{\uparrow} 1\mathrm{m} +_{\uparrow} 1\mathrm{d} = (2024, 05, 01) +_{\uparrow} 1\mathrm{d} = (2024, 05, 02)$$

Non-properties

Commutativity of addition

$$(2024, 03, 31) +_{\uparrow} 1\text{m} +_{\uparrow} 1\text{d} = (2024, 05, 01) +_{\uparrow} 1\text{d} = (2024, 05, 02)$$
$$(2024, 03, 31) +_{\uparrow} 1\text{d} +_{\uparrow} 1\text{m} = (2024, 04, 01) +_{\uparrow} 1\text{m} = (2024, 05, 01)$$

Non-properties

Commutativity of addition

$$(2024, 03, 31) +_{\uparrow} 1\text{m} +_{\uparrow} 1\text{d} = (2024, 05, 01) +_{\uparrow} 1\text{d} = (2024, 05, 02)$$

$$(2024, 03, 31) +_{\uparrow} 1\text{d} +_{\uparrow} 1\text{m} = (2024, 04, 01) +_{\uparrow} 1\text{m} = (2024, 05, 01)$$

“Associativity” of addition

$$(2024, 03, 31) +_{\uparrow} 1\text{m} +_{\uparrow} 1\text{m} = (2024, 05, 01) +_{\uparrow} 1\text{m} = (2024, 06, 01)$$

Non-properties

Commutativity of addition

$$(2024, 03, 31) +_{\uparrow} 1\text{m} +_{\uparrow} 1\text{d} = (2024, 05, 01) +_{\uparrow} 1\text{d} = (2024, 05, 02)$$
$$(2024, 03, 31) +_{\uparrow} 1\text{d} +_{\uparrow} 1\text{m} = (2024, 04, 01) +_{\uparrow} 1\text{m} = (2024, 05, 01)$$

“Associativity” of addition

$$(2024, 03, 31) +_{\uparrow} 1\text{m} +_{\uparrow} 1\text{m} = (2024, 05, 01) +_{\uparrow} 1\text{m} = (2024, 06, 01)$$
$$(2024, 03, 31) +_{r} 2\text{m} = (2024, 05, 31)$$

Meaningful ambiguities

Rounding choice can change comparisons

```
d + 1 month <= April 30 2024
```

Meaningful ambiguities

Rounding choice can change comparisons

```
d + 1 month <= April 30 2024
```

- Rounding-sensitive comparison `d = March 31 2024`

Meaningful ambiguities

Rounding choice can change comparisons

```
d + 1 month <= April 30 2024
```

- Rounding-sensitive comparison `d = March 31 2024`

When rounding up or down doesn't change a computation

```
d + 1 month <= April 15 2024
```

Meaningful ambiguities

Rounding choice can change comparisons

```
d + 1 month <= April 30 2024
```

- Rounding-sensitive comparison `d = March 31 2024`

When rounding up or down doesn't change a computation

```
d + 1 month <= April 15 2024
```

- No rounding? Safe

Meaningful ambiguities

Rounding choice can change comparisons

```
d + 1 month <= April 30 2024
```

- Rounding-sensitive comparison `d = March 31 2024`

When rounding up or down doesn't change a computation

```
d + 1 month <= April 15 2024
```

- No rounding? Safe
- Otherwise, the rounding of `d + 1 month` will not change the comparison.

Meaningful ambiguities

Rounding choice can change comparisons

```
d + 1 month <= April 30 2024
```

- Rounding-sensitive comparison `d = March 31 2024`

When rounding up or down doesn't change a computation

```
d + 1 month <= April 15 2024
```

- No rounding? Safe
- Otherwise, the rounding of `d + 1 month` will not change the comparison.

$\Longrightarrow$ Prove rounding-insensitivity of an expression *e*

Meaningful ambiguities

Rounding choice can change comparisons

```
d + 1 month <= April 30 2024
```

- Rounding-sensitive comparison `d = March 31 2024`

When rounding up or down doesn't change a computation

```
d + 1 month <= April 15 2024
```

- No rounding? Safe
- Otherwise, the rounding of `d + 1 month` will not change the comparison.

$\Longrightarrow$ Prove rounding-insensitivity of an expression e

- $\mathbb{E}_{\uparrow}[\![e]\!] = \mathbb{E}_{\downarrow}[\![e]\!]$

Meaningful ambiguities

Rounding choice can change comparisons

```
d + 1 month <= April 30 2024
```

- Rounding-sensitive comparison `d = March 31 2024`

When rounding up or down doesn't change a computation

```
d + 1 month <= April 15 2024
```

- No rounding? Safe
- Otherwise, the rounding of `d + 1 month` will not change the comparison.

$\Longrightarrow$ Prove rounding-insensitivity of an expression e

- $\mathbb{E}_{\uparrow}[\![e]\!] = \mathbb{E}_{\downarrow}[\![e]\!]$ encoded as $\mathtt{sync}(e)$

Meaningful ambiguities

Rounding choice can change comparisons

```
d + 1 month <= April 30 2024
```

- Rounding-sensitive comparison `d = March 31 2024`

When rounding up or down doesn't change a computation

```
d + 1 month <= April 15 2024
```

- No rounding? Safe
- Otherwise, the rounding of `d + 1 month` will not change the comparison.

$\Longrightarrow$ Prove rounding-insensitivity of an expression e

- $\mathbb{E}_{\uparrow}[\![e]\!] = \mathbb{E}_{\downarrow}[\![e]\!]$ encoded as $\mathtt{sync}(e)$
- Considering product programs with both rounding modes [DOM21]

Meaningful ambiguities

Rounding choice can change comparisons

```
d + 1 month <= April 30 2024
```

- Rounding-sensitive comparison `d = March 31 2024`

When rounding up or down doesn't change a computation

```
d + 1 month <= April 15 2024
```

- No rounding? Safe
- Otherwise, the rounding of `d + 1 month` will not change the comparison.

$\Longrightarrow$ Prove rounding-insensitivity of an expression e

- $\mathbb{E}_{\uparrow}[\![e]\!] = \mathbb{E}_{\downarrow}[\![e]\!]$ encoded as $\mathtt{sync}(e)$
- Considering product programs with both rounding modes [DOM21]
- Will reduce the need for costly legal interpretations

- Defines addition, accessors, projection, lexicographic comparison

- Defines addition, accessors, projection, lexicographic comparison
- Translates constraints on dates into numerical constraints

- Defines addition, accessors, projection, lexicographic comparison
- Translates constraints on dates into numerical constraints
  date $d_1 \rightsquigarrow$ ghost numerical variables $\mathsf{d}(d_1), \mathsf{m}(d_1), \mathsf{y}(d_1)$

- Defines addition, accessors, projection, lexicographic comparison
- Translates constraints on dates into numerical constraints
  date $d_1 \rightsquigarrow$ ghost numerical variables $\mathsf{d}(d_1), \mathsf{m}(d_1), \mathsf{y}(d_1)$
- Acts as a functor lifting a numerical abstract domain

- Defines addition, accessors, projection, lexicographic comparison
- Translates constraints on dates into numerical constraints
  date $d_1 \rightsquigarrow$ ghost numerical variables $\mathsf{d}(d_1), \mathsf{m}(d_1), \mathsf{y}(d_1)$
- Acts as a functor lifting a numerical abstract domain

$\mathsf{d}(d_1) \in [1, 31] \wedge \mathsf{m}(d_1) \in [1, 12] \wedge \mathsf{y}(d_1) = 2024$: all valid dates of 2024

`date d1 = rand_date(); date d2 = d1 + 1 month;` rounding down.

`date d1 = rand_date(); date d2 = d1 + 1 month;` rounding down.

- No concrete constraints on `d1`
- Intervals would be imprecise

$\Longrightarrow$ relational abstract domains needed!

`date d1 = rand_date(); date d2 = d1 + 1 month;` rounding down.

- No concrete constraints on `d1`
- Intervals would be imprecise

$\Longrightarrow$ relational abstract domains needed!

4 cases apply, including:

`date d1 = rand_date(); date d2 = d1 + 1 month;` rounding down.

- No concrete constraints on `d1`
- Intervals would be imprecise

$\Longrightarrow$ relational abstract domains needed!

4 cases apply, including:

- 30-day month

$$\mathtt{d}(d1) = 31,\ \mathtt{m}(d1) \in \{\mathtt{Mar}, \mathtt{May}, \mathtt{Aug}, \mathtt{Oct}\}, \mathtt{d}(d2) = 30,\ \mathtt{m}(d2) = \mathtt{m}(d1) + 1, \mathtt{y}(d2) = \mathtt{y}(d1)$$

`date d1 = rand_date(); date d2 = d1 + 1 month;` rounding down.

- No concrete constraints on `d1`
- Intervals would be imprecise

$\Longrightarrow$ relational abstract domains needed!

4 cases apply, including:

- 30-day month

$$\mathtt{d}(d1) = 31,\ \underbrace{\mathtt{m}(d1) \in \{\mathtt{Mar}, \mathtt{May}, \mathtt{Aug}, \mathtt{Oct}\}}_{\text{Bounded set of ints}}, \mathtt{d}(d2) = 30,\ \mathtt{m}(d2) = \mathtt{m}(d1) + 1, \mathtt{y}(d2) = \mathtt{y}(d1)$$

`date d1 = rand_date(); date d2 = d1 + 1 month;` rounding down.

- No concrete constraints on `d1`
- Intervals would be imprecise

$\Longrightarrow$ relational abstract domains needed!

4 cases apply, including:

- 30-day month

$$\mathtt{d}(d1) = 31,\ \underbrace{\mathtt{m}(d1) \in \{\mathtt{Mar}, \mathtt{May}, \mathtt{Aug}, \mathtt{Oct}\}}_{\text{Bounded set of ints}}, \mathtt{d}(d2) = 30,\ \underbrace{\mathtt{m}(d2) = \mathtt{m}(d1) + 1, \mathtt{y}(d2) = \mathtt{y}(d1)}_{\text{Polyhedra}}$$

`date d1 = rand_date(); date d2 = d1 + 1 month;` rounding down.

- No concrete constraints on `d1`
- Intervals would be imprecise

$\Longrightarrow$ relational abstract domains needed!

4 cases apply, including:

- 30-day month

$$\mathtt{d}(d1) = 31,\ \underbrace{\mathtt{m}(d1) \in \{\mathtt{Mar}, \mathtt{May}, \mathtt{Aug}, \mathtt{Oct}\}}_{\text{Bounded set of ints}}, \mathtt{d}(d2) = 30,\ \underbrace{\mathtt{m}(d2) = \mathtt{m}(d1) + 1, \mathtt{y}(d2) = \mathtt{y}(d1)}_{\text{Polyhedra}}$$

- No rounding $\quad \mathtt{d}(d1) = \mathtt{d}(d2),\ \mathtt{m(d2)} \equiv_{12} \mathtt{m(d1)} + 1, \mathtt{y}(d1) \leq \mathtt{y}(d2) \leq \mathtt{y}(d1) + 1$

`date d1 = rand_date(); date d2 = d1 + 1 month;` rounding down.

- No concrete constraints on `d1`
- Intervals would be imprecise

$\Longrightarrow$ relational abstract domains needed!

4 cases apply, including:

- 30-day month

$$\mathtt{d}(d1) = 31,\ \underbrace{\mathtt{m}(d1) \in \{\mathtt{Mar}, \mathtt{May}, \mathtt{Aug}, \mathtt{Oct}\}}_{\text{Bounded set of ints}}, \mathtt{d}(d2) = 30,\ \underbrace{\mathtt{m}(d2) = \mathtt{m}(d1) + 1, \mathtt{y}(d2) = \mathtt{y}(d1)}_{\text{Polyhedra}}$$

- No rounding $\mathtt{d}(d1) = \mathtt{d}(d2),\ \underbrace{\mathtt{m(d2)} \equiv_{12} \mathtt{m(d1)} + 1}_{\text{Linear congruence domain}}, \mathtt{y}(d1) \leq \mathtt{y}(d2) \leq \mathtt{y}(d1) + 1$

Abstract double semantics

Moving to double programs

- Analyze the program in both rounding modes

Abstract double semantics

Moving to double programs

- Analyze the program in both rounding modes
- Shallow variable duplication depending on their rounding mode

Abstract double semantics

Moving to double programs

- Analyze the program in both rounding modes
- Shallow variable duplication depending on their rounding mode

```
date d1 = rand_date(); date d2 = d1 + 1 month; double semantics
```

Abstract double semantics

Moving to double programs

- Analyze the program in both rounding modes
- Shallow variable duplication depending on their rounding mode

`date d1 = rand_date(); date d2 = d1 + 1 month;` double semantics

- No rounding

$$\mathsf{d}(d1) = \mathsf{d}(d2) \qquad \mathsf{m}(\mathtt{d2}) \equiv_{12} \mathsf{m}(\mathtt{d1}) + 1 \qquad \mathsf{y}(d1) \leq \mathsf{y}(d2) \leq \mathsf{y}(d1) + 1$$

Moving to double programs

- Analyze the program in both rounding modes
- Shallow variable duplication depending on their rounding mode

`date d1 = rand_date(); date d2 = d1 + 1 month;` double semantics

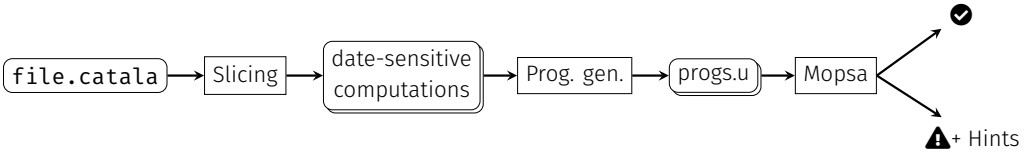
- No rounding

$$\mathsf{d}(d1) = \mathsf{d}(d2) \qquad \mathsf{m}(\mathtt{d2}) \equiv_{12} \mathsf{m}(\mathtt{d1}) + 1 \qquad \mathsf{y}(d1) \leq \mathsf{y}(d2) \leq \mathsf{y}(d1) + 1$$

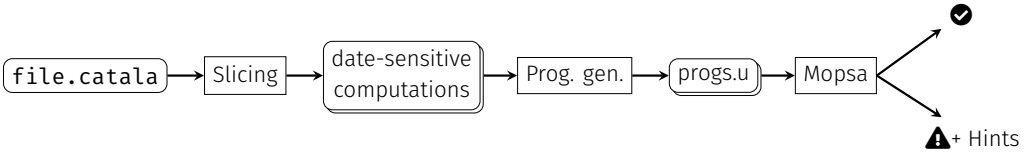
- 30-day month

$$\mathsf{d}(d1) = 31, \mathsf{m}(d1) \in \{\, \mathtt{Mar}, \mathtt{May}, \mathtt{Aug}, \mathtt{Sep} \,\}$$
$$\downarrow\mathsf{d}(d2) = 30, \downarrow\mathsf{m}(d2) \in \{\, \mathtt{Apr}, \mathtt{Jun}, \mathtt{Sep}, \mathtt{Nov} \,\}, \downarrow\mathsf{m}(d2) = \mathsf{m}(d1) + 1$$
$$\uparrow\mathsf{d}(d2) = 1, \uparrow\mathsf{m}(d2) \in \{\, \mathtt{May}, \mathtt{Jul}, \mathtt{Oct}, \mathtt{Dec} \,\}, \uparrow\mathsf{m}(d2) = \mathsf{m}(d1) + 2$$
$$\downarrow\mathsf{y}(d2) = \uparrow\mathsf{y}(d2) = \mathsf{y}(d1)$$

Date ambiguity detection pipeline

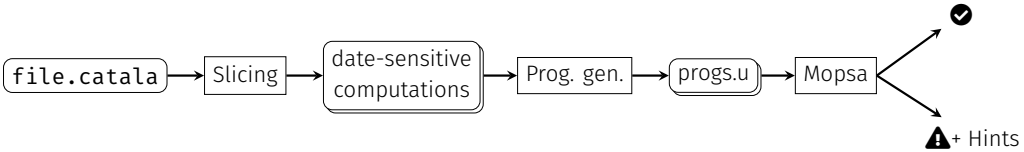


Date ambiguity detection pipeline



2 rounding-sensitive cases detected

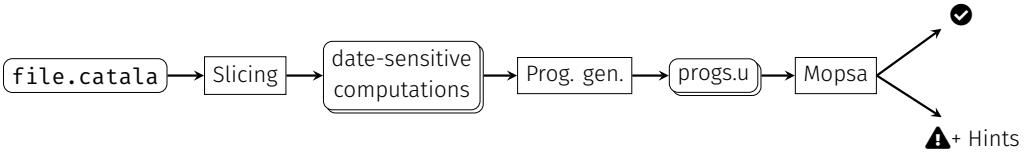
Date ambiguity detection pipeline



2 rounding-sensitive cases detected

No false alarms

Date ambiguity detection pipeline

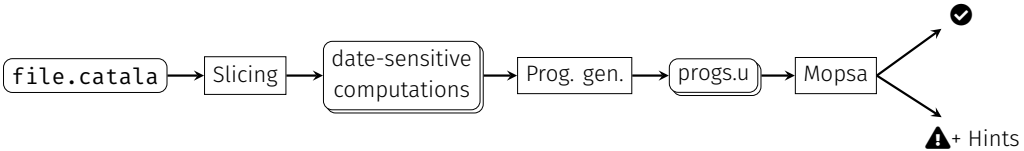


2 rounding-sensitive cases detected

No false alarms

Intra-scope extraction for now

Date ambiguity detection pipeline



2 rounding-sensitive cases detected

No false alarms

Intra-scope extraction for now

Manual inter-scope extraction

16 additional cases:

- 10 can be proved safe assuming `current_date` $\geq$ `2023`
- Other are real issues

Automated Verification of Catala Programs

CuteCat Concolic Execution [GFM25]

Concolic execution: first example

```
if x > 0
    then 0
    else if y < 10
        then 1
        else error
```

Concolic execution: first example

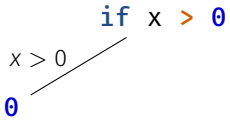
```
if x > 0
    then 0
    else if y < 10
        then 1
        else error
```

```
if x > 0
```

Step	x	y	Output	Constraints after evaluation	Next path to try

Concolic execution: first example

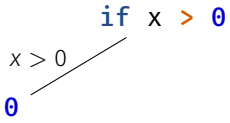
```
if x > 0
    then 0
    else if y < 10
        then 1
        else error
```



Step	x	y	Output	Constraints after evaluation	Next path to try
1	1	20	0	$x > 0$	

Concolic execution: first example

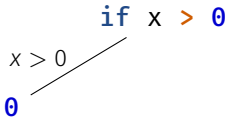
```
if x > 0
    then 0
    else if y < 10
        then 1
        else error
```



Step	x	y	Output	Constraints after evaluation	Next path to try
1	1	20	0	$x > 0$	$\neg(x > 0)$ ⟲ SMT

Concolic execution: first example

```
if x > 0
    then 0
    else if y < 10
        then 1
        else error
```

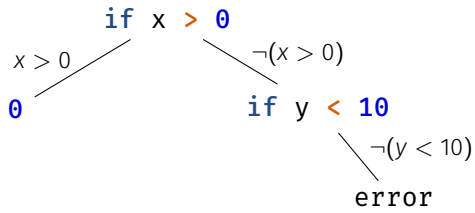


Step	x	y	Output	Constraints after evaluation	Next path to try
1	1	20	0	$x > 0$	$\neg(x > 0)$
2	0	20			

↻ SMT

Concolic execution: first example

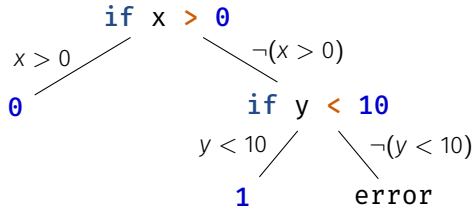
```
if x > 0
    then 0
    else if y < 10
        then 1
        else error
```



Step	x	y	Output	Constraints after evaluation	Next path to try	
1	1	20	0	$x > 0$	$\neg(x > 0)$	↻ SMT
2	0	20	`error`	$\neg(x > 0) \wedge \neg(y < 10)$	$\neg(x > 0) \wedge y < 10$	↻ SMT

Concolic execution: first example

```
if x > 0
    then 0
    else if y < 10
        then 1
        else error
```



Step	x	y	Output	Constraints after evaluation	Next path to try	
1	1	20	0	$x > 0$	$\neg(x > 0)$	↻ SMT
2	0	20	`error`	$\neg(x > 0) \wedge \neg(y < 10)$	$\neg(x > 0) \wedge y < 10$	↻ SMT
3	0	9	1	$\neg(x > 0) \wedge y < 10$	-	

Concolic execution of default terms

$$\langle\langle \mid \texttt{income} \leq \$10,000 \texttt{ :- } 10\% \rangle, \langle \mid \texttt{nb_children} \geq 3 \texttt{ :- } 15\% \rangle \mid \texttt{true} \texttt{ :- } 20\% \rangle$$

Concolic execution of default terms

$\langle\langle \mid \texttt{income} \leq \$10,000 \texttt{ :- } 10\% \rangle, \langle \mid \texttt{nb_children} \geq 3 \texttt{ :- } 15\% \rangle \mid \texttt{true} \texttt{ :- } 20\% \rangle$

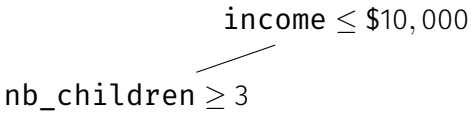
$\texttt{income} = \$9{,}000; \quad \texttt{nb_children} = 4$

$\texttt{income} \leq \$10,000$

Concolic execution of default terms

$\langle\langle \mid \texttt{income} \leq \$10,000 \texttt{ :- } 10\% \rangle, \langle \mid \texttt{nb_children} \geq 3 \texttt{ :- } 15\% \rangle \mid \texttt{true} \texttt{ :- } 20\% \rangle$

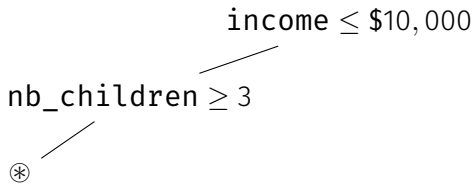
$\texttt{income} = \$9{,}000; \quad \texttt{nb_children} = 4$



Concolic execution of default terms

⟨⟨ | income ≤ $10,000 :- 10%⟩, ⟨ | nb_children ≥ 3 :- 15%⟩ | true :- 20%⟩

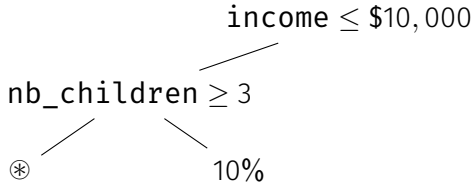
income = $9,000; nb_children = 4



Concolic execution of default terms

$\langle\langle \mid \texttt{income} \leq \$10,000 \texttt{ :- } 10\% \rangle, \langle \mid \texttt{nb_children} \geq 3 \texttt{ :- } 15\% \rangle \mid \texttt{true} \texttt{ :- } 20\% \rangle$

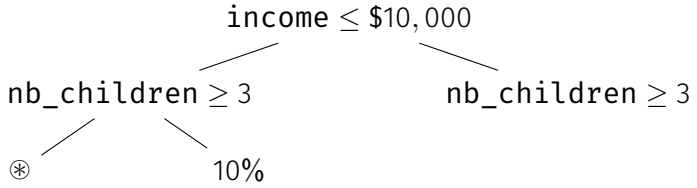
`income` = $9,000; `nb_children` = 2



Concolic execution of default terms

$\langle\langle \mid \texttt{income} \leq \$10,000 \texttt{ :- } 10\% \rangle, \langle \mid \texttt{nb_children} \geq 3 \texttt{ :- } 15\% \rangle \mid \texttt{true} \texttt{ :- } 20\% \rangle$

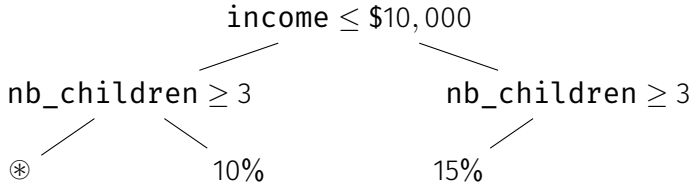
`income` = **$11,000**; `nb_children` = **4**



Concolic execution of default terms

$\langle\langle \mid \texttt{income} \leq \$10,000 \texttt{ :- } 10\% \rangle, \langle \mid \texttt{nb_children} \geq 3 \texttt{ :- } 15\% \rangle \mid \texttt{true} \texttt{ :- } 20\% \rangle$

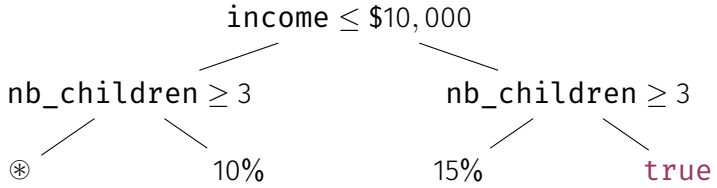
$\texttt{income} = \$11{,}000; \quad \texttt{nb_children} = 4$



Concolic execution of default terms

$\langle\langle \mid \texttt{income} \leq \$10,000 \texttt{ :- } 10\% \rangle, \langle \mid \texttt{nb_children} \geq 3 \texttt{ :- } 15\% \rangle \mid \texttt{true} \texttt{ :- } 20\% \rangle$

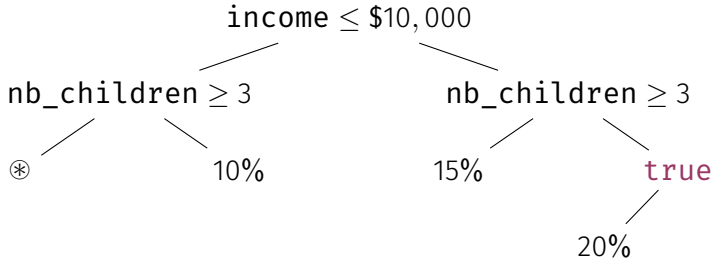
$\texttt{income} = \$11{,}000; \quad \texttt{nb_children} = 2$



Concolic execution of default terms

$\langle\langle \mid \texttt{income} \leq \$10,000 \texttt{ :- } 10\% \rangle, \langle \mid \texttt{nb_children} \geq 3 \texttt{ :- } 15\% \rangle \mid \texttt{true} \texttt{ :- } 20\% \rangle$

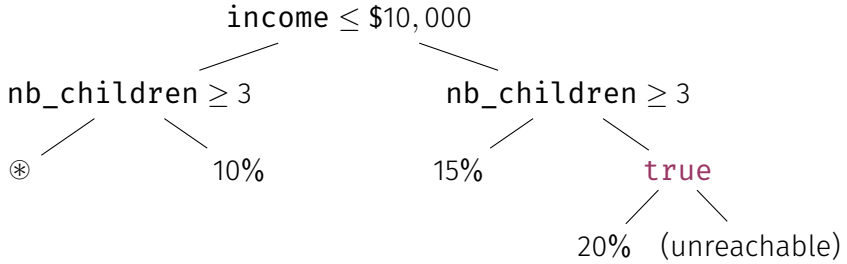
$\texttt{income} = \$11{,}000; \quad \texttt{nb_children} = 2$



Concolic execution of default terms

⟨⟨ | income ≤ $10,000 :- 10%⟩, ⟨ | nb_children ≥ 3 :- 15%⟩ | true :- 20%⟩

income = ???; nb_children = ???



Performance optimizations using reordering

Theorem (Independence of exception evaluation order)

If there is a default value v such that

$$\langle \ldots, e_i, \ldots, \mathbf{e_j}, \ldots \mid b_{default} \ :- \ e_{default} \rangle \longrightarrow^* v,$$

then

$$\langle \ldots, \mathbf{e_j}, \ldots, e_i, \ldots \mid b_{default} \ :- \ e_{default} \rangle \longrightarrow^* v.$$

Performance optimizations using reordering

Theorem (Independence of exception evaluation order)

If there is a default value v such that

$$\langle \ldots, e_i, \ldots, \boldsymbol{e_j}, \ldots \mid b_{default} \ :- \ e_{default} \rangle \longrightarrow^* v,$$

then

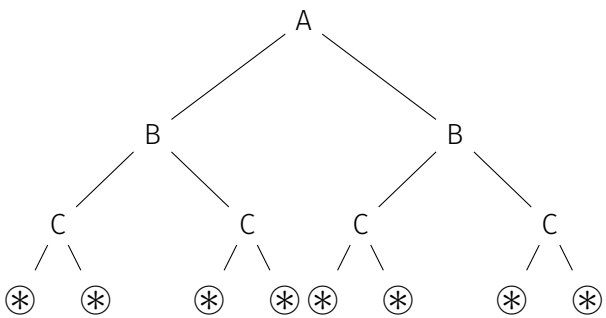
$$\langle \ldots, \boldsymbol{e_j}, \ldots, e_i, \ldots \mid b_{default} \ :- \ e_{default} \rangle \longrightarrow^* v.$$

Example:

$$\langle A, B, C, \circledast \mid b_{default} \ :- \ e_{default} \rangle \sim \langle \circledast, A, B, C \mid b_{default} \ :- \ e_{default} \rangle$$

Performance optimizations using reordering – Example

$$\langle A, B, C, \circledast \mid b_{default} :- e_{default} \rangle \sim \langle \circledast, A, B, C \mid b_{default} :- e_{default} \rangle$$





Usability improvement

```
# Article 3
If the income is less than $10,000, the percentage
mentioned at article 1 is 10%.
```catala
scope IncomeTaxComputation:
 exception definition tax_rate
  under condition house.income <= $10,000
  consequence equals 10%
```
```

Usability improvement

```
# Article 3
If the income is less than $10,000, the percentage
mentioned at article 1 is 10%.
```catala
scope IncomeTaxComputation:
 exception definition tax_rate
  under condition house.income <= $10,000
  consequence equals 10%
```
```

Query: income > $10,000 ?

Usability improvement

```
# Article 3
If the income is less than $10,000, the percentage
mentioned at article 1 is 10%.
```catala
scope IncomeTaxComputation:
 exception definition tax_rate
  under condition house.income <= $10,000
  consequence equals 10%
```
```

Query: income > $10,000 ? → Answer: $10,000.01

Usability improvement

```
# Article 3
If the income is less than $10,000, the percentage
mentioned at article 1 is 10%.
```catala
scope IncomeTaxComputation:
 exception definition tax_rate
  under condition house.income <= $10,000
  consequence equals 10%
```
```

Query: income > $10,000 ? $\rightarrow$ Answer: $10,000.01

- Difficult to compute by hand

Usability improvement

```
# Article 3
If the income is less than $10,000, the percentage
mentioned at article 1 is 10%.
```catala
scope IncomeTaxComputation:
 exception definition tax_rate
  under condition house.income <= $10,000
  consequence equals 10%
```
```

Query: income > $10,000 ? $\rightarrow$ Answer: $10,000.01

- Difficult to compute by hand
- Find more usable input values using **soft constraints**

Usability improvement

```
# Article 3
If the income is less than $10,000, the percentage
mentioned at article 1 is 10%.
```catala
scope IncomeTaxComputation:
 exception definition tax_rate
  under condition house.income <= $10,000
  consequence equals 10%
```
```

Query: income > $10,000 ? $\rightarrow$ Answer: ~~$10,000.01~~ **$11,000**

- Difficult to compute by hand
- Find more usable input values using soft constraints !
  - *e.g.* **round to $1,000**

Performance optimizations of the SMT solver

- Incremental mode

Performance optimizations of the SMT solver

- Incremental mode
  - Solver keeps a stack of constraints

Performance optimizations of the SMT solver

- Incremental mode
  - Solver keeps a stack of constraints
  - Keeps satisfiability status of save points

Performance optimizations of the SMT solver

- Incremental mode
  - Solver keeps a stack of constraints
  - Keeps satisfiability status of save points
  - Works best with depth-first exploration

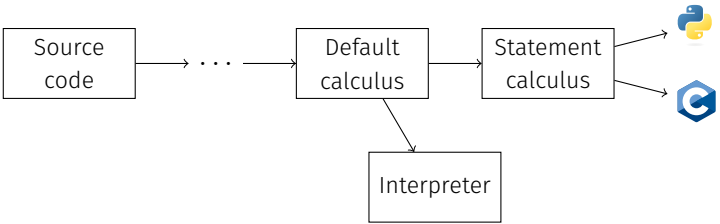
Performance optimizations of the SMT solver

- Incremental mode
  - Solver keeps a stack of constraints
  - Keeps satisfiability status of save points
  - Works best with depth-first exploration
  - Allows efficient soft constraints

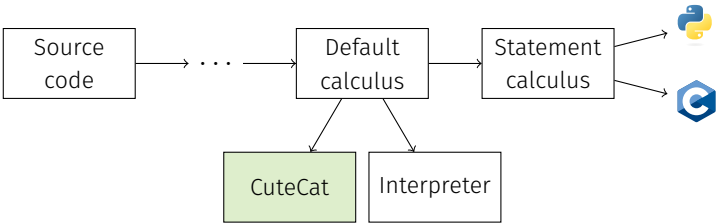
Performance optimizations of the SMT solver

- Incremental mode
  - Solver keeps a stack of constraints
  - Keeps satisfiability status of save points
  - Works best with depth-first exploration
  - Allows efficient soft constraints
- Redundant constraint elimination

Implementation of CuteCat



Implementation of CuteCat



- Integrated into Catala compiler's *default calculus* IR
- 3.4k lines of OCaml code
- **Z3** SMT Solver

Case study: housing benefits

French housing benefits: **5736** lines of law, **8615** lines of code

- 186,390 test cases generated in **7h of single-threaded CPU time**
- 366s spent in solver, the rest in evaluation
- Able to find a conflict: bedrooms vs flat sharing
- 99.83% of tests satisfy soft constraints

Overhead of the concolic execution

- 4.5x overhead w.r.t. Catala interpreter on our benchmarks
- SymCC [PF20] or SYMSAN [Che+22] reach the same order of magnitude
- KLEE sometimes reports several orders of magnitude [Yun+18]

Overhead of the concolic execution

- 4.5x overhead w.r.t. Catala interpreter on our benchmarks
- SymCC [PF20] or SYMSAN [Che+22] reach the same order of magnitude
- KLEE sometimes reports several orders of magnitude [Yun+18]
- More optimizations in future work

Conclusion

Conclusion

- Legal software vs. *rocket science*

Conclusion

- Legal software vs. *rocket science*
- Catala is an ideal testbed for automated verification research!

Conclusion

- Legal software vs. *rocket science*
- Catala is an ideal testbed for automated verification research!
- Ongoing work around “Auditing Government Benefits Screening Tools” [EB20]

Conclusion

- Legal software vs. *rocket science*
- Catala is an ideal testbed for automated verification research!
- Ongoing work around “Auditing Government Benefits Screening Tools” [EB20]
- Administrations and NGOs as a “third way” between academia and industry

References – I

[Bal19] Nathan Ballantyne. **“Epistemic trespassing”.** In: Mind 510 (2019), pp. 367–395.

[Che+22] Ju Chen et al. **“SYMSAN: Time and Space Efficient Concolic Execution via Dynamic Data-Flow Analysis”.** In: USENIX Association, 2022, pp. 2531–2548.

[DOM21] David Delmas, Abdelraouf Ouadjaout, and Antoine Miné. **“Static Analysis of Endian Portability by Abstract Interpretation”.** In: Lecture Notes in Computer Science. Springer, 2021, pp. 102–123.

References – II

[Dut20] Dutch Data Protection Authority. Werkwijze Belastingdienst in strijd met de wet en discriminerend. 2020. URL: https://autoriteitpersoonsgegevens.nl/actueel/werkwijze-belastingdienst-in-strijd-met-de-wet-en-discriminerend.

[EB20] Nel Escher and Nikola Banovic. "Exposing Error in Poverty Management Technology: A Method for Auditing Government Benefits Screening Tools". In: Proc. ACM Hum. Comput. Interact. CSCW (2020), 064:1–064:20.

[Esc+24] Nel Escher et al. "Code-ifying the Law: How Disciplinary Divides Afflict the Development of Legal Software". In: Proc. ACM Hum. Comput. Interact. CSCW2 (2024), pp. 1–37. DOI: 10.1145/3686937.

References – III

[GFM25] Pierre Goutagny, Aymeric Fromherz, and Raphaël Monat. **“CUTECat: Concolic Execution for Computational Law”.** In: ed. by Viktor Vafeiadis. Lecture Notes in Computer Science. Springer, 2025, pp. 31–61. DOI: 10.1007/978-3-031-91121-7_2.

[HM22] Liane Huttner and Denis Merigoux. **“Catala: Moving Towards the Future of Legal Expert Systems”.** In: Artificial Intelligence and Law (Aug. 2022). DOI: 10.1007/s10506-022-09328-5.

[Law24] Sarah Lawsky. **“Computational Law and Epistemic Trespassing”.** In: Journal of Cross-disciplinary Research in Computational Law 2 (May 2024).

References – IV

[MAS23] Denis Merigoux, Marie Alauzen, and Lilya Slimani. **“Rules, Computation and Politics. Scrutinizing Unnoticed Programming Choices in French Housing Benefits”.** In: Journal of Cross-disciplinary Research in Computational Law 1 (2023), p. 23.

[MCP21] Denis Merigoux, Nicolas Chataing, and Jonathan Protzenko. **“Catala: a programming language for the law”.** In: 2021.

[MFM24] Raphaël Monat, Aymeric Fromherz, and Denis Merigoux. **“Formalizing Date Arithmetic and Statically Detecting Ambiguities for the Law”.** In: ed. by Stephanie Weirich. Lecture Notes in Computer Science. Springer, 2024, pp. 421–450. DOI: 10.1007/978-3-031-57267-8_16.

References – V

[Moc18] Percy Mockler. The Phoenix Pay Problem: Working toward a Solution. Senate Canada, 2018.

[PF20] Sebastian Poeplau and Aurélien Francillon. “Symbolic Execution with SymCC: Don’t Interpret, Compile!” In: SEC’20. USA: USENIX Association, 2020, pp. 181–198.

[Red+22] J Redden et al. “Automating public services: learning from cancelled systems”. In: Cardiff, UK: Data Justice Lab, Cardiff University (2022).

[Yun+18] Insu Yun et al. “QSYM : A Practical Concolic Execution Engine Tailored for Hybrid Fuzzing”. In: USENIX Association, 2018, pp. 745–761.