CUTECat

Generating Testcases for Computational Laws through Concolic Execution

Pierre Goutagny[1] Aymeric Fromherz[2] Raphaël Monat[1]

Catala Seminar, March 10, 2025

[1]Inria Lille, [2]Inria Paris

Introduction

In this talk

- Program that encode laws
- What bugs these programs can have
- How Catala prevents some of them
- How I can detect them before they even happen

Computational law

- **Computational laws** encode algorithms: taxes, social benefits, etc.
- Administrations implement them as programs
- Critical: *e.g.* French military payroll system Louvois: 120k military personnel over- or under-paid, overpayments totalling 545M € to pay back

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

Transforming this law into a program

- **Input**: household description
  - income
  - number of children
- **Output**: computed income tax

Transforming this law into a program

- **Input**: household description
  - income
  - number of children
- **Output**: computed income tax
- Compute according to default logic

Transforming this law into a program

- **Input**: household description
  - income
  - number of children
- **Output**: computed income tax
- Compute according to default logic
- Stay close to the text of the law

A simple Catala program

~~~
# 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%
```
~~~

- Literate programming

~~~
# 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%
```
~~~

A simple Catala program

~~~
# 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%
```
~~~

- Literate programming

~~~
# 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%
```
~~~

A simple Catala program

~~~
# 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%
```
~~~

- Literate programming
- Follows the exception/default structure of the law

Kinds of error

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

Kinds of error

~~~~
# 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 the code
  - interpretation conflicts, *e.g.* `income = $9,000` and `children = 4`
  - unhandled cases
  - in Catala: ambiguity = crash
  - resolved by lawyers/administration if implementation is correct

Kinds of error

~~~
# 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 the code
  - interpretation conflicts, *e.g.* `income = $9,000` and `children = 4`
  - unhandled cases
  - in Catala: ambiguity = crash
  - resolved by lawyers/administration if implementation is correct
- Other errors: division by zero, assertion error, etc.

Two levels of assurance

- Crashing in ambiguous situations

Two levels of assurance

- Crashing in ambiguous situations
    - Catala doesn’t silently favor one interpretation

Two levels of assurance

- Crashing in ambiguous situations
  - Catala doesn’t silently favor one interpretation
  - Happens when the code is executed

Two levels of assurance

- Crashing in ambiguous situations
  - Catala doesn’t silently favor one interpretation
  - Happens when the code is executed
  - Risky when used in real life

Two levels of assurance

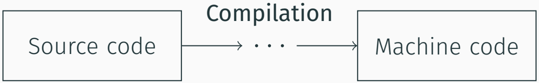
- Crashing in ambiguous situations
    - Catala doesn’t silently favor one interpretation
    - Happens when the code is executed
    - Risky when used in real life
- Anticipate those bugs when the code is written: we want to find bugs *a priori*

Background

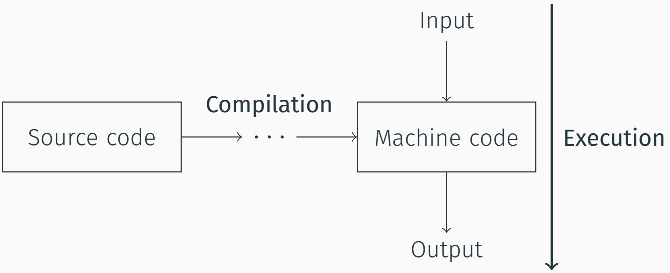
Program compilation & execution

Source code

Program compilation & execution



Program compilation & execution



Bugs

Execution can yield two outcomes

- Expected output

Bugs

Execution can yield two outcomes

- Expected output
- Unexpected behavior

Bugs

Execution can yield two outcomes

- Expected output
- Unexpected behavior
    - Freeze

Bugs

Execution can yield two outcomes

- Expected output
- Unexpected behavior
  - Freeze
  - Crash

Bugs

Execution can yield two outcomes

- Expected output
- Unexpected behavior
  - Freeze
  - Crash
  - Wrong result

Bugs

Execution can yield two outcomes

- Expected output
- Unexpected behavior
  - Freeze
  - Crash
  - Wrong result

Bugs are relative to the expected behavior: specification

How to find bugs

- Testing by hand: time consuming, tedious, error prone

How to find bugs

- Testing by hand: time consuming, tedious, error prone
- Automate random inputs

How to find bugs

- Testing by hand: time consuming, tedious, error prone
- Automate random inputs
- Code review

How to find bugs

- Testing by hand: time consuming, tedious, error prone
- Automate random inputs
- Code review
- … no guarantee that we can find every bug

How to find bugs

- Testing by hand: time consuming, tedious, error prone
- Automate random inputs
- Code review
- ... no guarantee that we can find every bug
- → Formal methods

How to find bugs

- Testing by hand: time consuming, tedious, error prone
- Automate random inputs
- Code review
- … no guarantee that we can find every bug
- $\rightarrow$ Formal methods
    - rigorous reasoning about the program

How to find bugs

- Testing by hand: time consuming, tedious, error prone
- Automate random inputs
- Code review
- … no guarantee that we can find every bug
- → Formal methods
    - rigorous reasoning about the program
    - find bugs systematically

How to find bugs

- Testing by hand: time consuming, tedious, error prone
- Automate random inputs
- Code review
- … no guarantee that we can find every bug
- → Formal methods
    - rigorous reasoning about the program
    - find bugs systematically
    - guarantee formally that program has bugs/no bugs

Formal methods: reasoning on programs

- More generally, we want to reason on programs

Formal methods: reasoning on programs

- More generally, we want to reason on programs
- Identify and prove properties

Formal methods: reasoning on programs

- More generally, we want to reason on programs
- Identify and prove properties
    - bugs: “The program always terminates without crashing”

Formal methods: reasoning on programs

- More generally, we want to reason on programs
- Identify and prove properties
  - bugs: “The program always terminates without crashing”
  - statistics: “The program performs less than 10 additions”

Formal methods: reasoning on programs

- More generally, we want to reason on programs
- Identify and prove properties
  - bugs: “The program always terminates without crashing”
  - statistics: “The program performs less than 10 additions”
  - “Income tax increases with income”

Formal methods: reasoning on programs

- More generally, we want to reason on programs
- Identify and prove properties
    - bugs: “The program always terminates without crashing”
    - statistics: “The program performs less than 10 additions”
    - “Income tax increases with income”
    - “The marginal tax rate is bounded”

Formal methods: reasoning on programs

- More generally, we want to reason on programs
- Identify and prove properties
  - bugs: “The program always terminates without crashing”
  - statistics: “The program performs less than 10 additions”
  - “Income tax increases with income”
  - “The marginal tax rate is bounded”
- Different properties are proven with different methods

For Catala

- Focus on finding bugs

For Catala

- Focus on finding bugs
- We expect well written Catala programs not to crash

For Catala

- Focus on finding bugs
- We expect well written Catala programs not to crash
- Property: “No inputs can lead the Catala program to an ambiguous situation”

For Catala

- Focus on finding bugs
- We expect well written Catala programs not to crash
- Property: “No inputs can lead the Catala program to an ambiguous situation”
- Our method must:
    - handle default logic
    - generate (counter-)examples for non-programmers

For Catala

- Focus on finding bugs
- We expect well written Catala programs not to crash
- Property: “No inputs can lead the Catala program to an ambiguous situation”
- Our method must:
  - handle default logic
  - generate (counter-)examples for non-programmers

→ Concolic execution

Outline

Background

Concolic execution of default terms

Performance and usability improvements

Experimental evaluation

Concolic execution of default terms

Concolic execution: first example

Concolic = *conc*rete + symb*olic*

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

Concolic execution: first example

Concolic = *conc*rete + symb*olic*

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

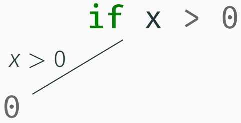
```
if x > 0
```

Step	x	y	Output	Constraints after evaluation	Next path to try

Concolic execution: first example

Concolic = *concrete* + symb*olic*

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

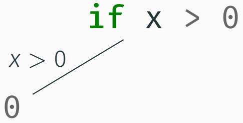


Step	x	y	Output	Constraints after evaluation	Next path to try
1	1	20	0	$x > 0$	

Concolic execution: first example

Concolic = *concrete* + symb*olic*

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



if x > 0

$x > 0$

0

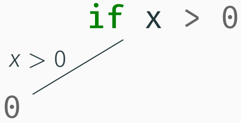
Step	x	y	Output	Constraints after evaluation	Next path to try
1	1	20	0	$x > 0$	$\neg(x > 0)$

Solver

Concolic execution: first example

Concolic = *concrete* + symb*olic*

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



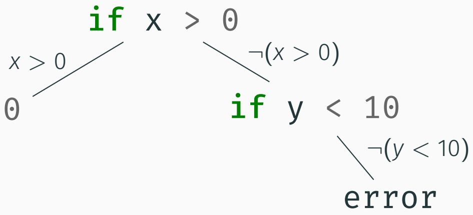
Step	x	y	Output	Constraints after evaluation	Next path to try
1	1	20	0	$x > 0$	$\neg(x > 0)$
2	0	20			

⤸ Solver

Concolic execution: first example

Concolic = *conc*rete + symb*olic*

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



if x > 0

$x > 0$

0

$\neg(x > 0)$

if y < 10

$\neg(y < 10)$

error

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

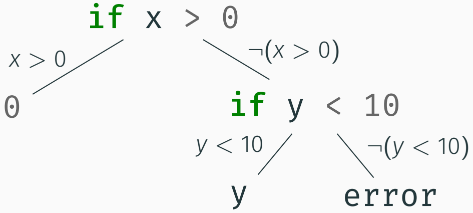
⤸ Solver

⤸ Solver

Concolic execution: first example

Concolic = *concrete* + symb*olic*

```
if x > 0
  then 0
  else if y < 10
    then y
    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	`error`	$\neg(x > 0) \land \neg(y < 10)$	$\neg(x > 0) \land y < 10$
3	0	9	9	$\neg(x > 0) \land y < 10$	-

Solver

Solver

Default terms: syntax

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

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

Default terms: syntax

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

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

Default terms: syntax

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

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

Default terms: syntax

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

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

Default terms: syntax

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

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

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

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%
```

Default terms: semantics

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

```
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

$$\left\langle \begin{array}{l} \langle \mid \texttt{income} \leq \$10,000 \;\texttt{:-}\; 10\% \rangle, \\ \langle \mid \texttt{nb\_children} \geq 3 \;\texttt{:-}\; 15\% \rangle \\ \mid \texttt{true} \;\texttt{:-}\; 20\% \end{array} \right\rangle$$

```
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 $b_{default} =$ `true`, then **evaluate** $e_{default}$
    - Else if $b_{default} =$ `false`, then **return** $\varnothing$

Default terms: semantics

$$\begin{aligned}
&\langle \\
&\quad \langle \mid \texttt{income} \leq \$10,000 \;\texttt{:-}\; 10\% \rangle, \\
&\quad \langle \mid \texttt{nb\_children} \geq 3 \;\texttt{:-}\; 15\% \rangle \\
&\quad \mid \texttt{true} \;\texttt{:-}\; 20\% \\
&\rangle
\end{aligned}$$

```
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

$$\left\langle \begin{array}{l} \langle \;|\; \texttt{income} \leq \$10,000 \;\texttt{:-}\; 10\% \rangle, \\ \langle \;|\; \texttt{nb\_children} \geq 3 \;\texttt{:-}\; 15\% \rangle \\ |\; \texttt{true} \;\texttt{:-}\; 20\% \end{array} \right\rangle$$

```
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 $b_{default} =$ `true`, then **evaluate** $e_{default}$
   - Else if $b_{default} =$ `false`, then **return** $\varnothing$

Default terms: semantics

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

```
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** $\varnothing$

Concolic execution of default terms

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

Concolic execution of default terms

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

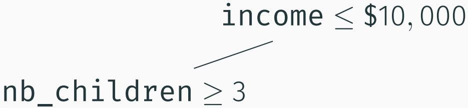
$\texttt{income} = \$9{,}000; \quad \texttt{nb\_children} = 4$

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

Concolic execution of default terms

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

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



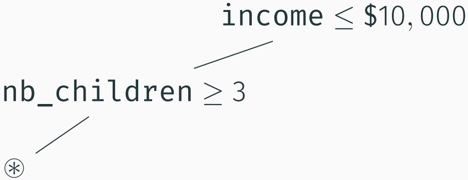
$\texttt{income} \leq \$10,000$

$\texttt{nb\_children} \geq 3$

Concolic execution of default terms

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

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



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

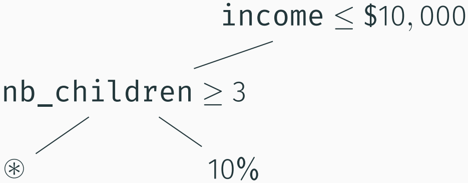
$\texttt{nb\_children} \geq 3$

$\circledast$

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; $\texttt{nb\_children}$ = 2



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

$\texttt{nb\_children} \geq 3$

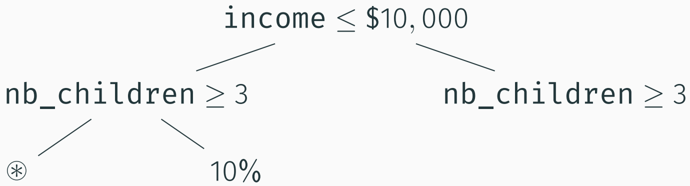
$\circledast$

10%

Concolic execution of default terms

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

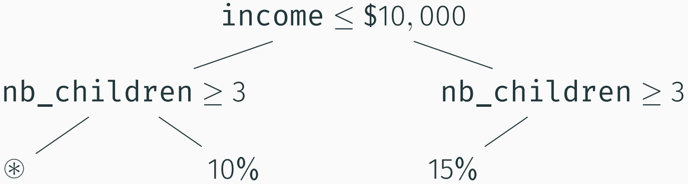
`income` = \$11,000; `nb_children` = 4



Concolic execution of default terms

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

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



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

$\texttt{nb\_children} \geq 3$

$\texttt{nb\_children} \geq 3$

$\circledast$

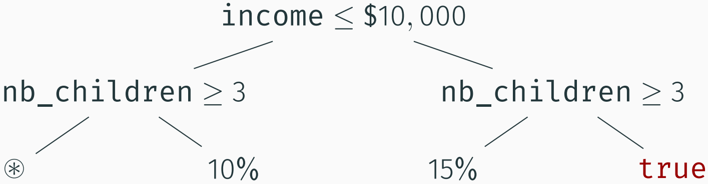
10%

15%

Concolic execution of default terms

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

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



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

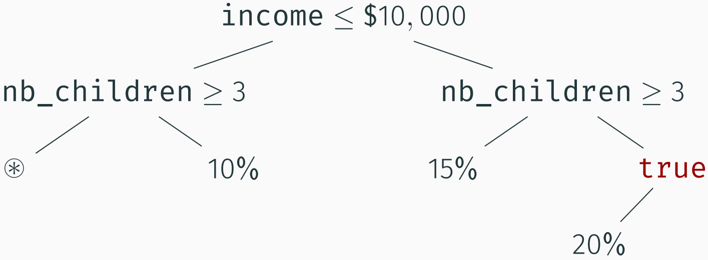
$\texttt{nb\_children} \geq 3$ $\qquad$ $\texttt{nb\_children} \geq 3$

$\circledast$ $\qquad$ 10% $\qquad$ 15% $\qquad$ $\texttt{true}$

Concolic execution of default terms

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

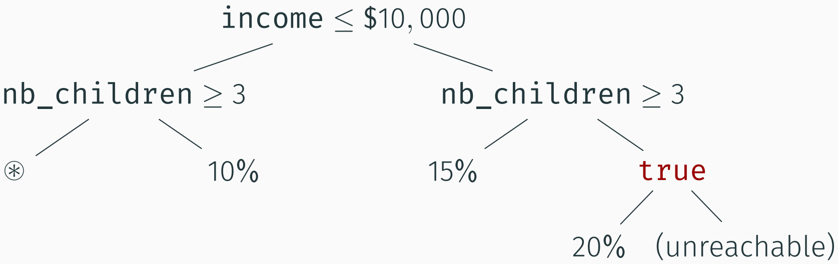
$\texttt{income} = \$11{,}000; \quad \texttt{nb\_children} = 2$



Concolic execution of default terms

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

$\texttt{income} = ???; \qquad \texttt{nb\_children} = ???$



Fixing the interpretation conflict

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

~~~
# 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%
```
```
```

Fixing the interpretation conflict

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

~~~
# Article 3
If the income is less than $10,000, the percentage mentioned at article 1 is 10%.
```catala
scope IncomeTaxComputation:
 exception children 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:
 label children
 exception definition tax_rate
  under condition house.nb_children >= 3
  consequence equals 15%
~~~

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.
→ 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%
⟩
```

Performance and usability improvements

Performance optimizations using reordering

Independence of exception evaluation order

Performance optimizations using reordering

Theorem (Independence of exception evaluation order)

If there is a default value v such that

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

then

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

Performance optimizations using reordering

Theorem (Independence of exception evaluation order)

If there is a default value v such that

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

then

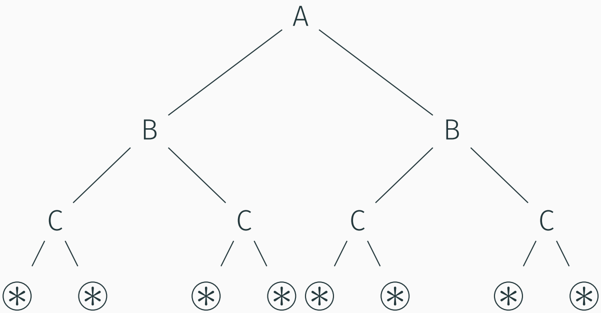
$$\langle \ldots, e_j, \ldots, e_i, \ldots \mid b_{default} :- e_{default} \rangle \longrightarrow^* v.$$

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 ? → 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 ? → Answer: $10,000.01

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

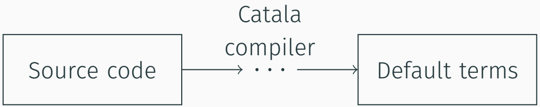
Usability improvement

~~~~
# 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%
```
~~~~

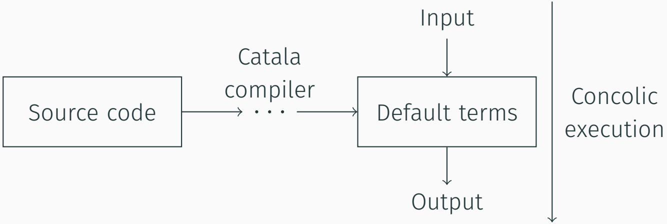
Query: income > $10,000 ? → Answer: ~~$10,000.01~~ **$11,000**

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

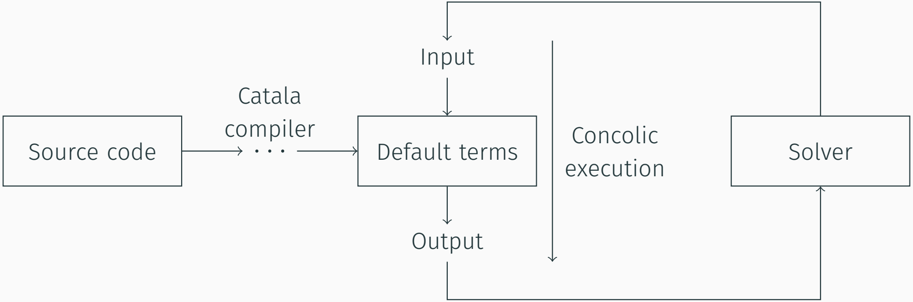
Implementation of CUTECat



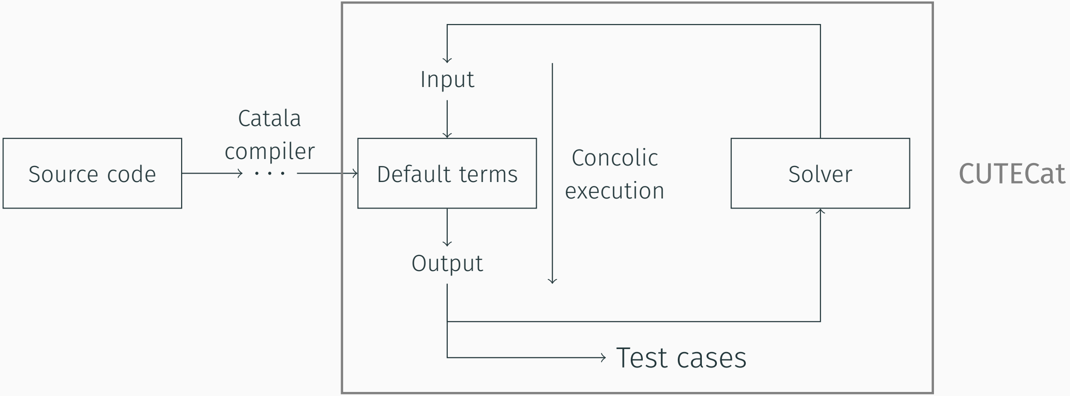
Implementation of CUTECat



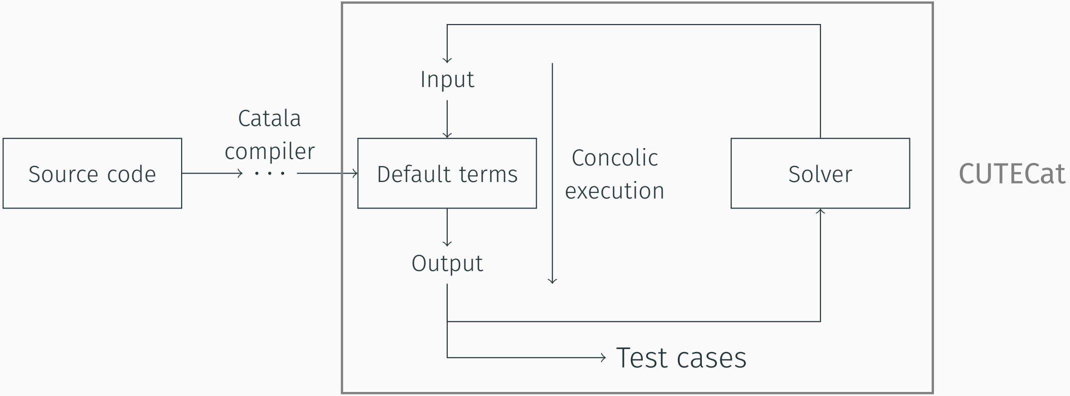
Implementation of CUTECat



Implementation of CUTECat



Implementation of CUTECat



- 3.4k lines of OCaml code
- Z3 SMT Solver

Experimental evaluation

Code base

Law	Lines of law in Markdown	Lines of Catala	Total
French housing benefits	5736	8615	14351

Code base

Law	Lines of law in Markdown	Lines of Catala	Total
French housing benefits	**5736**	**8615**	**14351**
US Tax code § 132	35	56	91
Minimum wage	74	161	235
Family quotient	36	165	201
Handwritten unit tests	139	699	838

Performance on small programs

	Time (s)			
Law	No optimizations	Incremental	All opt.	Generated tests
US Tax code	0.27	0.02	0.02	10
Minimum wage	1.01	0.08	0.08	17
Family quotient	82.61	5.21	4.34	381

Case study: housing benefits

Key results

- 186,390 test cases generated in **7h of CPU time**
- 99.83% of tests satisfy soft constraints
- Able to find a conflict
- 4.5x overhead w.r.t. concrete execution
- 366s spent in solver, the rest in evaluation

Conclusion

Conclusion

- CUTECat: a concolic testing engine for computational law
- Optimizations improve efficiency and usability for lawyers
- $\sim$**200k test cases** in less than **7h** on real-world example

Conclusion

- CUTECat: a concolic testing engine for computational law
- Optimizations improve efficiency and usability for lawyers
- $\sim$**200k test cases** in less than **7h** on real-world example

Future work:

- Complex cases *e.g.* lists and dates
- Conformance testing *e.g.* for simulator
- Improve user-friendliness for non-programmers

Conclusion

- CUTECat: a concolic testing engine for computational law
- Optimizations improve efficiency and usability for lawyers
- $\sim$**200k test cases** in less than **7h** on real-world example

Future work:

- Complex cases *e.g.* lists and dates
- Conformance testing *e.g.* for simulator
- Improve user-friendliness for non-programmers

Questions:

- What properties to prove?
- How to integrate analysis steps in practice?

Conclusion

- CUTECat: a concolic testing engine for computational law
- Optimizations improve efficiency and usability for lawyers
- $\sim$**200k test cases** in less than **7h** on real-world example

Future work:

- Complex cases *e.g.* lists and dates
- Conformance testing *e.g.* for simulator
- Improve user-friendliness for non-programmers

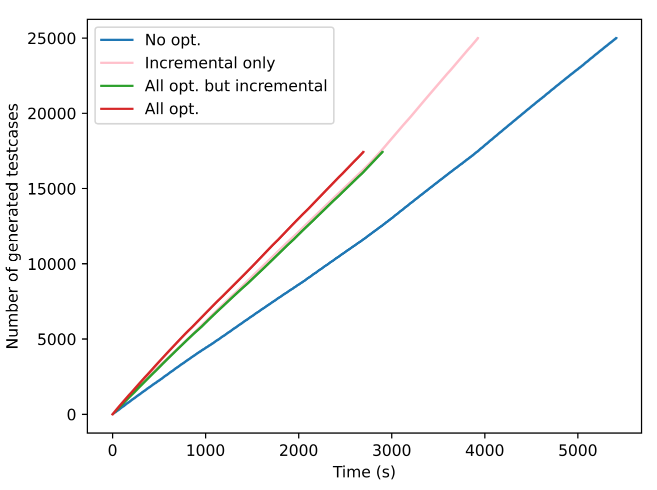
Questions:

- What properties to prove?
- How to integrate analysis steps in practice?

Contact, ESOP'25 preprint, slides: pierregoutagny.fr

References i

Ablation study



Generated tests vs time