Formal Logic for Law, Language and Artificial Intelligence

Joost J. Joosten

Universitat de Barcelona

Facultad de Ciencias, Universidad de Salamanca
Seminar related to *Nuevas Tendencias en Sistemas Inteligentes*, Friday, February 27, 2026

The business model of our research lab

Business focussed: from concrete to abstract

Work presented in collaboration with various co-authors: Moritz Müller, Juli Ponce Solé, David Fernández-Duque, Bjørn Jespersen, Ana de Almeida Borges, Eduardo Hermo Reyes, Sofia Santiago Fernández, Petia Guintech, Mireia González Bedmar, Juan Conejero Rodríguez, Marina López Chamoza, Eric Sancho Adamson, Aleix Solé Sanchez, Quim Casals Buñuel, Marta Soria Heredia, Guillermo Errezil Alberdi, Daniel Soussa E Ribeiro, etc.

Law and Code

- Law essentially discretional powers when applied

Law and Code

- Law essentially discretional powers when applied
- Hence, open texture is needed

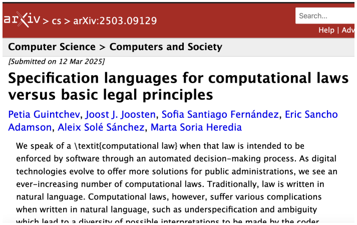
Law and Code

- Law essentially discretional powers when applied
- Hence, open texture is needed
- Any automated process and in particular, any automated process in the legal sector need unambiguity

Law and Code

- Law essentially discretional powers when applied
- Hence, open texture is needed
- Any automated process and in particular, any automated process in the legal sector need unambiguity
- The programmer needs to disambiguate?

An Example from a recent paper



arXiv > cs > arXiv:2503.09129

Search...

Help | Adv

Computer Science > Computers and Society

[Submitted on 12 Mar 2025]

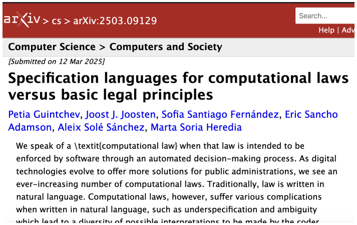
Specification languages for computational laws versus basic legal principles

Petia Guintchev, Joost J. Joosten, Sofia Santiago Fernández, Eric Sancho Adamson, Aleix Solé Sánchez, Marta Soria Heredia

We speak of a \textit{computational law} when that law is intended to be enforced by software through an automated decision-making process. As digital technologies evolve to offer more solutions for public administrations, we see an ever-increasing number of computational laws. Traditionally, law is written in natural language. Computational laws, however, suffer various complications when written in natural language, such as underspecification and ambiguity which lead to a diversity of possible interpretations to be made by the coder

- What can possibly be wrong with the following:

An Example from a recent paper



arXiv > cs > arXiv:2503.09129

Search...

Help | Adv

Computer Science > Computers and Society

[Submitted on 12 Mar 2025]

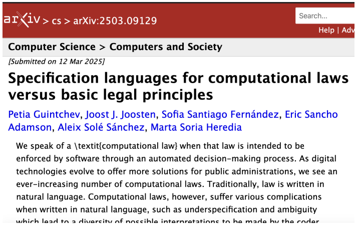
Specification languages for computational laws versus basic legal principles

Petia Guintchev, Joost J. Joosten, Sofia Santiago Fernández, Eric Sancho Adamson, Aleix Solé Sánchez, Marta Soria Heredia

We speak of a \textit{computational law} when that law is intended to be enforced by software through an automated decision-making process. As digital technologies evolve to offer more solutions for public administrations, we see an ever-increasing number of computational laws. Traditionally, law is written in natural language. Computational laws, however, suffer various complications when written in natural language, such as underspecification and ambiguity which lead to a diversity of possible interpretations to be made by the coder

- What can possibly be wrong with the following:
- Article 6.1.: The daily driving time shall not exceed nine hours. However, the daily driving time may be extended to at most 10 hours not more than twice during the week.

An Example from a recent paper



arXiv > cs > arXiv:2503.09129

Search...

Help | Adv

Computer Science > Computers and Society

[Submitted on 12 Mar 2025]

Specification languages for computational laws versus basic legal principles

Petia Guintchev, Joost J. Joosten, Sofia Santiago Fernández, Eric Sancho Adamson, Aleix Solé Sánchez, Marta Soria Heredia

We speak of a \textit{computational law} when that law is intended to be enforced by software through an automated decision-making process. As digital technologies evolve to offer more solutions for public administrations, we see an ever-increasing number of computational laws. Traditionally, law is written in natural language. Computational laws, however, suffer various complications when written in natural language, such as underspecification and ambiguity which lead to a diversity of possible interpretations to be made by the coder

- What can possibly be wrong with the following:
- Article 6.1.: The daily driving time shall not exceed nine hours. However, the daily driving time may be extended to at most 10 hours not more than twice during the week.
- EU Regulation 561/2006 on road transport

Small leap in a year, giant step for a truck-driver

- Art. 4. (i) '**a week**' *means the period of time between 00.00 on Monday and 24.00 on Sunday*;

Small leap in a year, giant step for a truck-driver

- Art. 4. (i) *'**a week**' means the period of time between 00.00 on Monday and 24.00 on Sunday*;
- Art. 4. (k): *'**daily driving time**' means the total accumulated driving time between the end of one daily rest period and the beginning of the following daily rest period or between a daily rest period and a weekly rest period*;.

Small leap in a year, giant step for a truck-driver

- Art. 4. (i) '**a week**' *means the period of time between 00.00 on Monday and 24.00 on Sunday*;
- Art. 4. (k): '**daily driving time**' *means the total accumulated driving time between the end of one daily rest period and the beginning of the following daily rest period or between a daily rest period and a weekly rest period;*.
- *Daily* in *daily drivingtime* is not a subsective modifier, rather *daily drivingtime* is a privative phrase

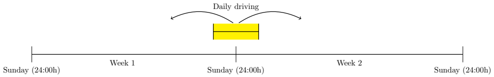
Small leap in a year, giant step for a truck-driver

- Art. 4. (i) '**a week**' *means the period of time between 00.00 on Monday and 24.00 on Sunday*;
- Art. 4. (k): '**daily driving time**' *means the total accumulated driving time between the end of one daily rest period and the beginning of the following daily rest period or between a daily rest period and a weekly rest period;*.
- *Daily* in *daily drivingtime* is not a subsective modifier, rather *daily drivingtime* is a privative phrase
- *Weekly* in *weekly rest* is a subsective modifier, but of the informal notion rather than of the formal/technical one

Small leap in a year, giant step for a truck-driver

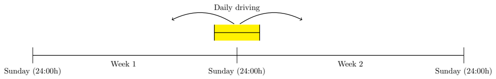
- Art. 4. (i) '**a week**' *means the period of time between 00.00 on Monday and 24.00 on Sunday*;
- Art. 4. (k): '**daily driving time**' *means the total accumulated driving time between the end of one daily rest period and the beginning of the following daily rest period or between a daily rest period and a weekly rest period;*.
- *Daily* in *daily drivingtime* is not a subsective modifier, rather *daily drivingtime* is a privative phrase
- *Weekly* in *weekly rest* is a subsective modifier, but of the informal notion rather than of the formal/technical one
- UTC and 27 leap seconds

Small leap in a year, giant step for a truck-driver



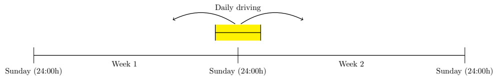
- Underspecifaction

Small leap in a year, giant step for a truck-driver



- Underspecifaction
- Are we free to optimise? Some software does. With or without leap seconds?

Small leap in a year, giant step for a truck-driver



- Underspecifaction
- Are we free to optimise? Some software does. With or without leap seconds?
- Non-locality!

Non-locality in weekly rest periods

Regulation (EC) No 561/2006

§8.6. In any two consecutive weeks, a driver shall take at least:

- two regular weekly rest periods [of at least 45 hours], or
- one regular weekly rest period and one reduced weekly rest period of at least 24 hours. However, the reduction shall be compensated by an equivalent period of rest taken en bloc before the end of the third week following the week in question.

A weekly rest period shall start no later than at the end of six 24-hour periods from the end of the previous weekly rest period.

§8.9. A weekly rest period that falls in two weeks may be counted in either week, but not in both.

Let's break it down...

- Regular weekly rest: ≥ 45 hours

Let's break it down...

- Regular weekly rest: $\geq$ 45 hours
- Reduced weekly rest: $\geq$ 24 hours

Let's break it down...

- Regular weekly rest: ≥ 45 hours
- Reduced weekly rest: ≥ 24 hours
- Each rest period is assigned to only one week it intersects

Let's break it down...

- Regular weekly rest: $\geq$ 45 hours
- Reduced weekly rest: $\geq$ 24 hours
- Each rest period is assigned to only one week it intersects
- Every week must have a regular or reduced weekly rest

Let's break it down...

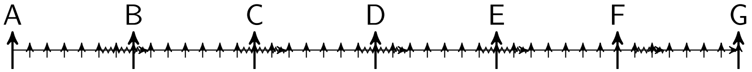
- Regular weekly rest: $\geq$ 45 hours
- Reduced weekly rest: $\geq$ 24 hours
- Each rest period is assigned to only one week it intersects
- Every week must have a regular or reduced weekly rest
- Every other week must have a full weekly rest

Let's break it down...

- Regular weekly rest: ≥ 45 hours
- Reduced weekly rest: ≥ 24 hours
- Each rest period is assigned to only one week it intersects
- Every week must have a regular or reduced weekly rest
- Every other week must have a full weekly rest
- Any reduced rest must be compensated by a continuous block in the following three weeks

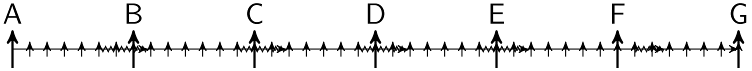
Combinatorics of rest assignments

Can we assign a week to each rest period so that each week is assigned to at least one rest period?



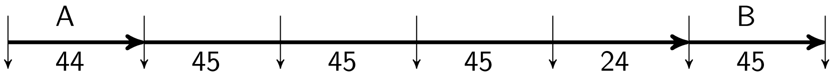
Combinatorics of rest assignments

Can we assign a week to each rest period so that each week is assigned to at least one rest period?



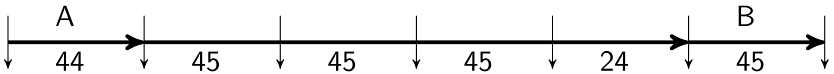
In principle this is an NP problem (assign 0 or 1 to each rest period according to whether it should belong to the earlier week or the later week).

Non-locality of compensations

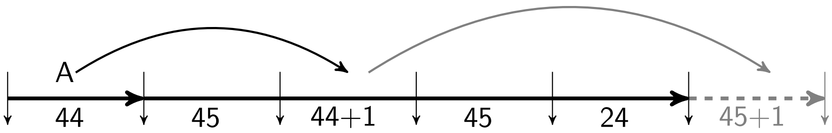


Illegal

Non-locality of compensations

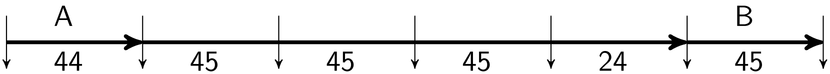


Illegal

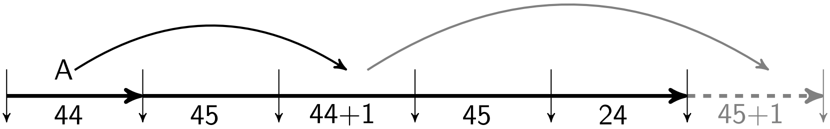


Legal

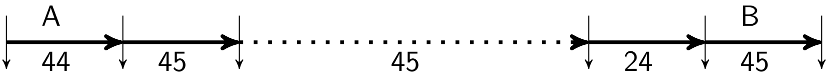
Non-locality of compensations



Illegal

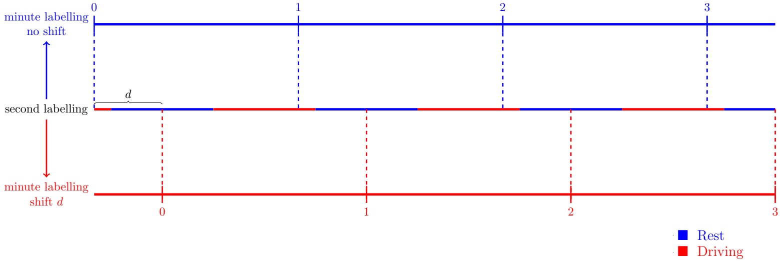


Legal



This can be iterated indefinitely: non-locality

Small leap in a year, giant step for a truck-driver



(51) Given a calendar minute, if DRIVING is registered as the activity of both the immediately preceding and the immediately succeeding minute, the whole minute shall be regarded as DRIVING.

(52) Given a calendar minute that is not regarded as DRIVING according to requirement 051, the whole minute shall be regarded to be of the same type of activity as the longest continuous activity within the minute (or the latest of the equally long activities).

Regulation (EU) 2016/799

The central computational problem of algorithmic law

Need to formalize activity sequences and laws

- formalize activity sequences are words $w \in \Sigma^*$ over a finite alphabet Σ
  e.g, dddrrw formalizes 6 minutes of activities in $\Sigma = \{d, r, w\}$.
- formalize a law by a sentence in a suitable logic L.

Need algorithm that decides the computational problem

$MC(\Sigma^*, L)$
Input: a word $w \in \Sigma^*$ and a sentence $\varphi \in L$
Problem: is w legal according to φ, i.e. $w \models \varphi$?

$\mathrm{MC}(\Sigma^*, L)$ is a formal model for algorithmic law (on activity sequences).
Question For which L is it good?

Which MC(Σ^*, L) are good models for algorithmic law?

Tractability

sufficiently fast model-checkers

fine-grained complexity analysis: parameterized complexity theory

important parameter: large time constants in law

Expressivity

test case: formalize Regulation 561

Naturality

readable sentences

sufficiently succinct

Candidate: monadic second order logic MSO

Starting point

Borges, Conejero, Fernández-Duque, González, Joosten.
To drive or not to drive: A logical and computational analysis of European transport regulations. Information and Computation 280, 2021.

- naturally formalizes Regulation 561.
- model-checking in time $f(|\varphi|) \cdot |w|$, Parameterized Complexity
  where $f : \mathbb{N} \to \mathbb{N}$ is some computable function.
- but f grows very fast:

  Theorem *(Frick, Grohe 04)*
  Assume $P \neq NP$. Then $MC(\Sigma^*, MSO)$ is not decidable in time

$$f(|\varphi|) \cdot |w|^{O(1)}$$

  for elementary $f : \mathbb{N} \to \mathbb{N}$.

Hence MSO is not sufficiently tractable.

Candidate: linear time temporal logic LTL

Model-checking in time $O(|\varphi| \cdot |w|)$, but not sufficiently expressive and not sufficiently succinct

Example `Article 6.2: The weekly driving time shall not exceed 56 hours`

Straightforwardly formalized over words of length $1w$: disjunction of

$$\bigwedge_{d \leq D} \left(\bigwedge_{r_d \leq i < \ell_{d+1}} \bigcirc^i \neg d \wedge \bigwedge_{\ell_d \leq i < r_d} \bigcirc^i d \right)$$

for all $D \leq 1w$ and

all $r_0 := 0 \leq \ell_1 < r_1 < \cdots < \ell_D < r_D < \ell_{D+1} := 1w$ with

$$\sum_{1 \leq j \leq D} (r_j - \ell_j) \leq 56h$$

This has $> \binom{7 \cdot 24 \cdot 60}{56 \cdot 60} > 10^{2784}$ many disjuncts.

Warning

Algorithmic laws could use large constants for time constraints.
Model-checking complexity should scale well with them.

FrameTitle

Using bisimulation techniques one can prove:

Theorem

There is no $\mathcal{L}_{\circ,\Box}$ formula equivalent to $\psi_{8.6}$ over the class of eventually resting models.

Theorem

All $\mathcal{L}_{\circ\mathsf{U}}$ formulas equivalent to $\psi_{8.6}$ have U*-depth at least* 1140*.*

The central computational problem of algorithmic law

Need to formalize activity sequences and laws

- formalize activity sequences are words $w \in \Sigma^*$ over a finite alphabet Σ
  e.g, dddrrw formalizes 6 minutes of activities in $\Sigma = \{d, r, w\}$.
- formalize a law by a sentence in a suitable logic L.

Need algorithm that decides the computational problem

$MC(\Sigma^*, L)$
Input: a word $w \in \Sigma^*$ and a sentence $\varphi \in L$
Problem: is w legal according to φ, i.e. $w \models \varphi$?

MC(Σ^*, L) is a formal model for algorithmic law (on activity sequences).
Question For which L is it good?

Which MC(Σ^*, L) are good models for algorithmic law?

Tractability

sufficiently fast model-checkers

fine-grained complexity analysis: parameterized complexity theory

important parameter: large time constants in law

Expressivity

test case: formalize Regulation 561

Naturality

readable sentences

sufficiently succinct

Stopwatch automata SWA: syntax

Stopwatch automaton $\mathbb{A}$

Q finite set of states including start, accept

X finite set of stopwatches

λ maps $q \in Q$ to $\lambda(q) \in \Sigma$

β maps $x \in X$ to bound $\beta(x) \in \mathbb{N}$

ζ is the set of $(x, q) \in X \times Q$ such that x is active in q

Δ is the set of transitions (q, g, α, q')

where $q, q' \in Q$, g is a guard, α is an action.

Assignment ξ maps $x \in X$ to $\xi(x) \leq \beta(x)$

Guard g is a set of assignments

Action α maps assignments to assignments

Stopwatch automata SWA: semantics

Transition system of $\mathbb{A}$

configurations (q, ξ)

switch edges $(q, \xi) \xrightarrow{0} (q', \xi')$

whenever $(q, g, \alpha, q') \in \Delta$, $\xi \in g$, $\xi' = \alpha(\xi)$

stay edges $(q, \xi) \xrightarrow{t} (q, \xi')$

where ξ' *increases* $\xi(x)$ *for* x *active in* q *to* $\min\{\xi(x)+t, \beta(x)\}$

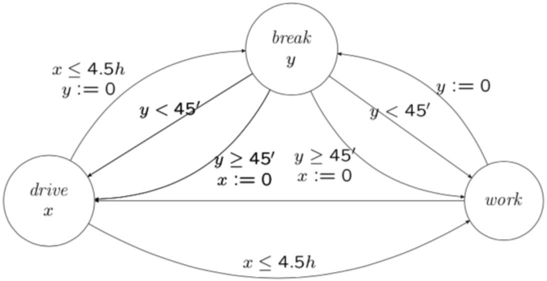
Computation $(q_0, \xi_0) \xrightarrow{t_0} (q_1, \xi_1) \xrightarrow{t_1} (q_2, \xi_2) \xrightarrow{t_2} \cdots \xrightarrow{t_{\ell-1}} (q_\ell, \xi_\ell)$

reads $w := \lambda(q_0)^{t_0}\ \lambda(q_1)^{t_1} \cdots \lambda(q_{\ell-1})^{t_{\ell-1}}$

accepts if $q_0 = \mathit{start}$, $\xi_0 \equiv 0$, $q_\ell = \mathit{accept}$, $q_i \neq \mathit{accept}$ for $i < \ell$.

Example: continuous driving

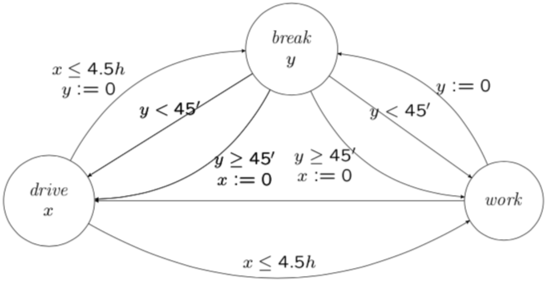
`Article 7 (1st part): After a driving period of four and a half hours a driver shall take an uninterrupted break of not less than 45 minutes,...`



$(drive, 00) \xrightarrow{3} (drive, 30) \xrightarrow{0} (break, 30) \xrightarrow{2} (break, 32) \xrightarrow{0} (work, 32) \xrightarrow{0} (break, 30)$
reads *dddrr*

Example: continuous driving

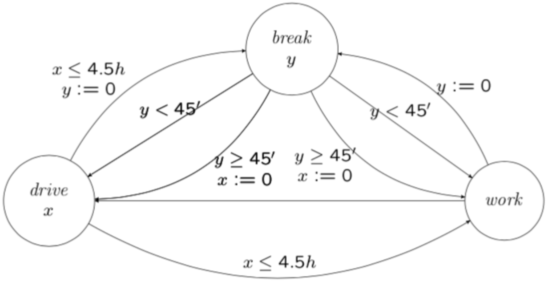
`Article 7 (1st part): After a driving period of four and a half hours a driver shall take an uninterrupted break of not less than 45 minutes,...`



$(\mathit{drive}, 00) \xrightarrow{3} (\mathit{drive}, 30) \xrightarrow{0} (\mathit{break}, 30) \xrightarrow{2} (\mathit{break}, 32) \xrightarrow{0} (\mathit{work}, 32) \xrightarrow{0}$
$(\mathit{break}, 30)$
reads *dddrr*

Example: continuous driving

`Article 7 (1st part): After a driving period of four and a half hours a driver shall take an uninterrupted break of not less than 45 minutes,...`

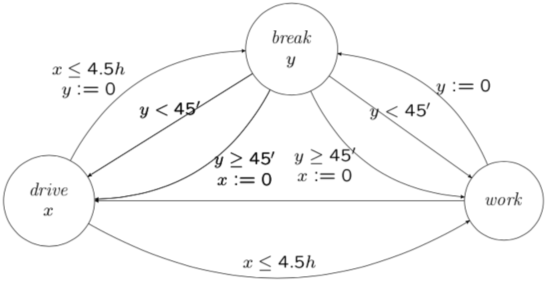


$(drive, 00) \xrightarrow{3}$ $(drive, 30)$ $\xrightarrow{0} (break, 30) \xrightarrow{2} (break, 32) \xrightarrow{0} (work, 32) \xrightarrow{0}$ $(break, 30)$

reads *dddrr*

Example: continuous driving

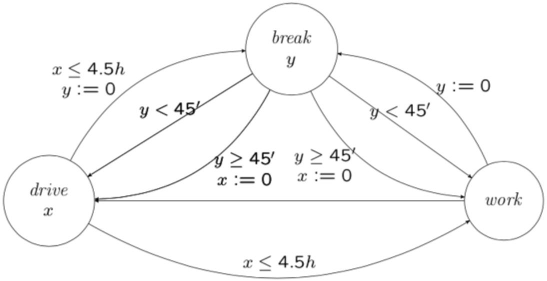
`Article 7 (1st part): After a driving period of four and a half hours a driver shall take an uninterrupted break of not less than 45 minutes,...`



$(drive, 00) \xrightarrow{3} (drive, 30) \xrightarrow{0} (break, 30) \xrightarrow{2} (break, 32) \xrightarrow{0} (work, 32) \xrightarrow{0} (break, 30)$
reads *dddrr*

Example: continuous driving

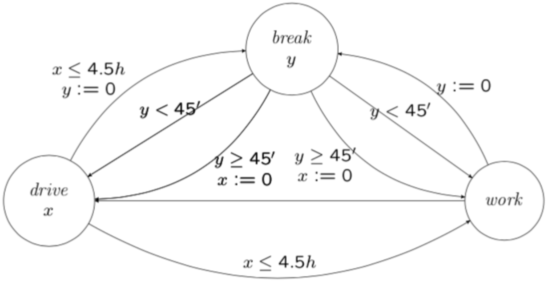
`Article 7 (1st part): After a driving period of four and a half hours a driver shall take an uninterrupted break of not less than 45 minutes,...`



$(\textit{drive}, 00) \xrightarrow{3} (\textit{drive}, 30) \xrightarrow{0} (\textit{break}, 30) \xrightarrow{2} (\textit{break}, 32) \xrightarrow{0} (\textit{work}, 32) \xrightarrow{0} (\textit{break}, 30)$
reads *dddrr*

Example: continuous driving

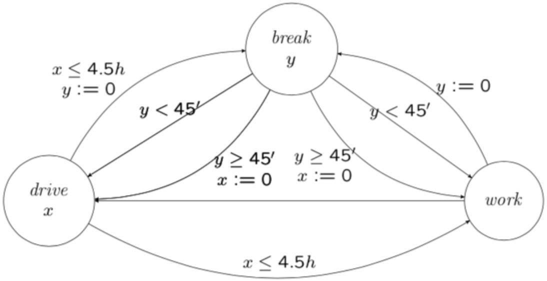
`Article 7 (1st part): After a driving period of four and a half hours a driver shall take an uninterrupted break of not less than 45 minutes,...`



$(drive, 00) \xrightarrow{3} (drive, 30) \xrightarrow{0} (break, 30) \xrightarrow{2} (break, 32) \xrightarrow{0} (work, 32) \xrightarrow{0} (break, 30)$
reads *dddrr*

Example: continuous driving

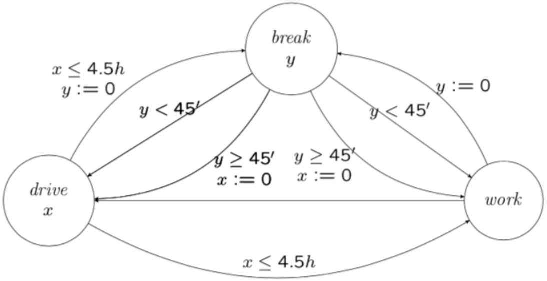
`Article 7 (1st part): After a driving period of four and a half hours a driver shall take an uninterrupted break of not less than 45 minutes,...`



$(drive, 00) \xrightarrow{3} (drive, 30) \xrightarrow{0} (break, 30) \xrightarrow{2} (break, 32) \xrightarrow{0} (work, 32) \xrightarrow{0}$
$(break, 30)$
reads *dddrr*

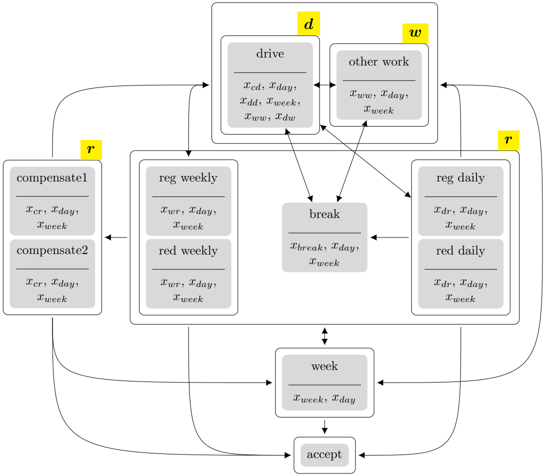
Example: continuous driving

`Article 7 (1st part): After a driving period of four and a half hours a driver shall take an uninterrupted break of not less than 45 minutes,...`



$(drive, 00) \xrightarrow{3} (drive, 30) \xrightarrow{0} (break, 30) \xrightarrow{2} (break, 32) \xrightarrow{0} (work, 32) \xrightarrow{0}$
$(break, 30)$
reads *ddd* *rr*

Automaton that accepts exactly the legal words according to Reg. 561


d
w
drive
$x_{cd}, x_{day}, x_{dd}, x_{week}, x_{ww}, x_{dw}$
other work
$x_{ww}, x_{day}, x_{week}$
r
compensate1
$x_{cr}, x_{day}, x_{week}$
compensate2
$x_{cr}, x_{day}, x_{week}$
reg weekly
$x_{wr}, x_{day}, x_{week}$
red weekly
$x_{wr}, x_{day}, x_{week}$
break
$x_{break}, x_{day}, x_{week}$
r
reg daily
$x_{dr}, x_{day}, x_{week}$
red daily
$x_{dr}, x_{day}, x_{week}$
week
x_{week}, x_{day}
accept

12 states

$>$ 100 transitions

34 stopwatches

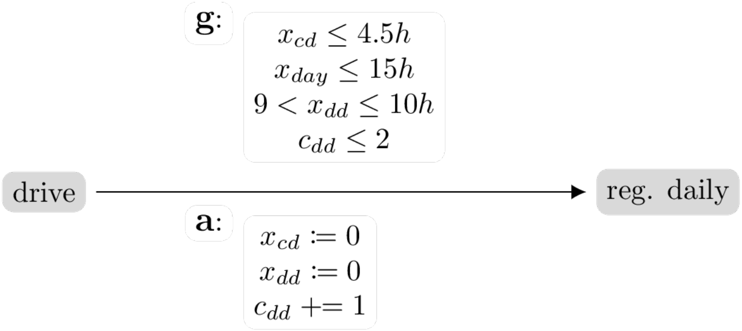
23 are nowhere active:

bits

counters

registers

$>$ 100 transitions


$$\begin{tikzcd}
\boxed{\text{drive}} \arrow[r, "{\mathbf{g}: \begin{array}{c} x_{cd} \leq 4.5h \\ x_{day} \leq 15h \\ 9 < x_{dd} \leq 10h \\ c_{dd} \leq 2 \end{array}}", "{\mathbf{a}: \begin{array}{c} x_{cd} := 0 \\ x_{dd} := 0 \\ c_{dd} += 1 \end{array}}"'] & \boxed{\text{reg. daily}}
\end{tikzcd}$$

Expressivity and model-checking

Theorem

A set of words is accepted by an SWA iff it is definable in MSO.

Theorem

There is an algorithm that decides

Input: stopwatch automaton $\mathbb{A}$ and a word w over Σ

Problem: does $\mathbb{A}$ accepts w ?

in time

$$O\big(|\mathbb{A}|^2 \cdot t^x \cdot |w|\big)$$

where

$t :=$ *largest stopwatch bound of* $\mathbb{A}$

$x :=$ *number of stopwatches of* $\mathbb{A}$

Consistency-checking

Theorem
There is an algorithm that decides

Input: SWAs $\mathbb{A}, \mathbb{B}$

Problem: is there a word accepted by both $\mathbb{A}$ *and* $\mathbb{B}$ *?*

in time

$O(|\mathbb{A}|^3 \cdot |\mathbb{B}|^3 \cdot t^x \cdot s^y)$

where

$t :=$ *largest stopwatch bound of* $\mathbb{A}$

$x :=$ *number of stopwatches of* $\mathbb{A}$

$s :=$ *largest stopwatch bound of* $\mathbb{B}$

$y :=$ *number of stopwatches of* $\mathbb{B}$

Scheduling

Theorem

There is an algorithm that decides

Input: SWA $\mathbb{A}$, *letter* $a \in \Sigma$, *word* w *over* Σ, $n \in \mathbb{N}$

Problem: compute length n *word* v *over* Σ *such that*

$\mathbb{A}$ *accepts* wv

v *maximizes* $\#_a(v)$

in time

$$O\big(|\mathbb{A}|^2 \cdot t^x \cdot (|w| + n)\big)$$

where

$t :=$ *largest stopwatch bound of* $\mathbb{A}$

$x :=$ *number of stopwatches of* $\mathbb{A}$

Lower bound

Know: MC(Σ^*, SWA) decidable in time $O\big(|\mathbb{A}|^2 \cdot t^x \cdot |w|\big)$

Doubt: Is t^x tolerable? Can it be improved?

Interesting instances have large t and small x.

Question: replace t^x by $100^{100\cdot x} \cdot t^{100}$?

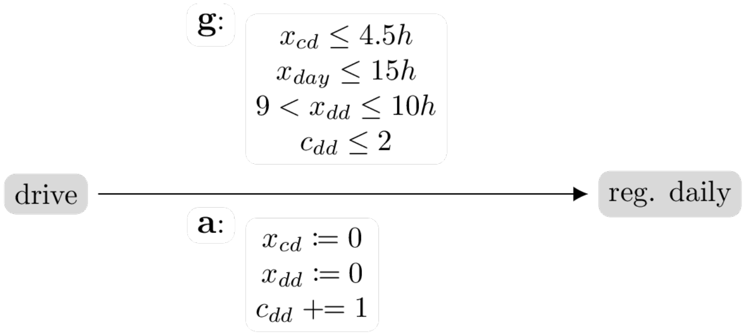
Theorem

Assume FPT $\neq$ W[1]. Let $f : \mathbb{N} \to \mathbb{N}$ be a computable function.
Then MC(Σ^*, SWA) cannot be decided in time

$$\big(|\mathbb{A}| \cdot f(x) \cdot t \cdot |w|\big)^{O(1)}.$$

Question: Can we hardwire large constants in the data structure using Hybrid Modal Logic?

$>$ 100 transitions



How to be sure that I do not make an error implementing all these transitions?

Tons of programming errors

- On June 4, 1996, the maiden flight of the European Ariane 5 launcher crashed about 40 seconds after takeoff

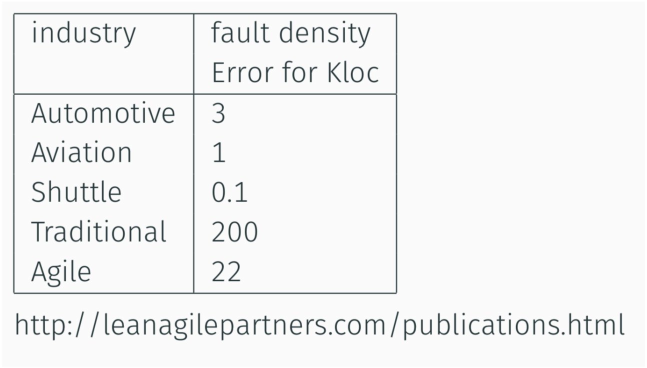
Tons of programming errors

- On June 4, 1996, the maiden flight of the European Ariane 5 launcher crashed about 40 seconds after takeoff
- Media reports indicated that the amount lost was half a billion dollars – uninsured

Tons of programming errors

- On June 4, 1996, the maiden flight of the European Ariane 5 launcher crashed about 40 seconds after takeoff
- Media reports indicated that the amount lost was half a billion dollars – uninsured
- A costly software error!

Errors are a matter of fact



industry	fault density Error for Kloc
Automotive	3
Aviation	1
Shuttle	0.1
Traditional	200
Agile	22

http://leanagilepartners.com/publications.html

Can we do any better?

Is there no way to **know** that a program is correct, rather than having much circumstantial evidence?

What is certification?

- Is it just a matter of trust? (combined with some sanity checks and experience)

What is certification?

- Is it just a matter of trust? (combined with some sanity checks and experience)
- Certificate $\Longrightarrow$ something is certain

What is certification?

- Is it just a matter of trust? (combined with some sanity checks and experience)
- Certificate $\Longrightarrow$ something is certain
- Verify $\Longrightarrow$ something is veridical

The impossibility of unrestricted certification

Alan Turing

- A mathematical theorem:

The impossibility of unrestricted certification

Alan Turing

- A mathematical theorem:
- Unrestricted certification is impossible.

Restricted certification is possible

We call a program P a *universal certifier* (wrt its language) when P takes two inputs

1. another program Q in a language compatible with P and,
2. a specification S in a language compatible with P that describes the behaviour of the program Q;

and, given two inputs Q and S, the program P outputs:

- **"YES"** if the program Q does what is said by S and,
- **"NO"** if the program Q does something different as that what is claimed by S.

Theorem

There does not exist a universal certifier.
This holds for any reasonable class of languages.

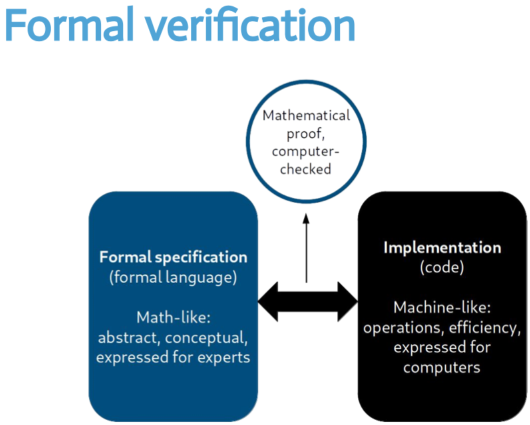
Formally verified software

Components of formally verified/certified software

Σ	A Specification: a non-ambiguous mathematical description of the input-output behaviour of the software
Π	Implementation: the code, the software, implementing the algorithm that does the work.
Δ	Proof: a mathematical proof that the program Π functions as claimed by Σ

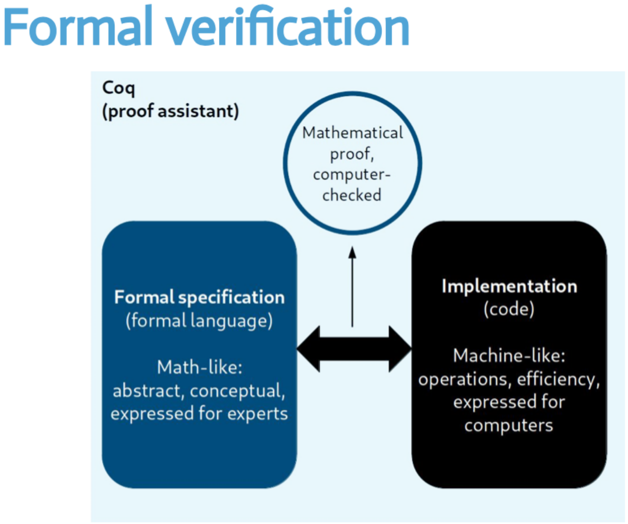
The specification Σ is written in a formal language (in our case, the language of dependent types of the Coq proof assistant).
This begs the question: **How to make the specification more accessible to the general/judicial public?**

What is verification?


Formal verification
Mathematical proof, computer-checked
Formal specification
(formal language)
Math-like:
abstract, conceptual, expressed for experts
Implementation
(code)
Machine-like:
operations, efficiency, expressed for computers

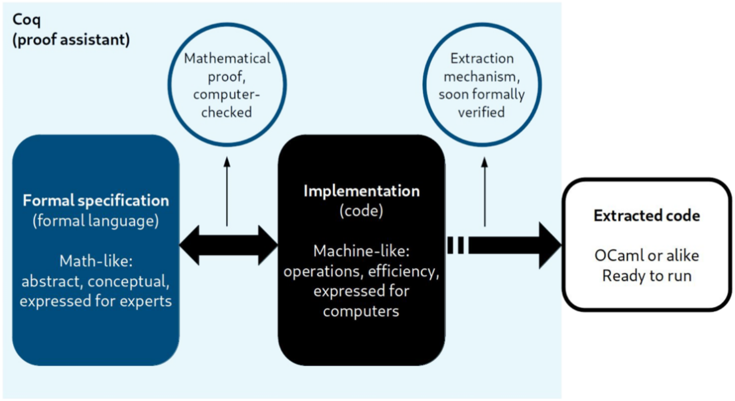
Slides FV: González Bedmar

What is verification?

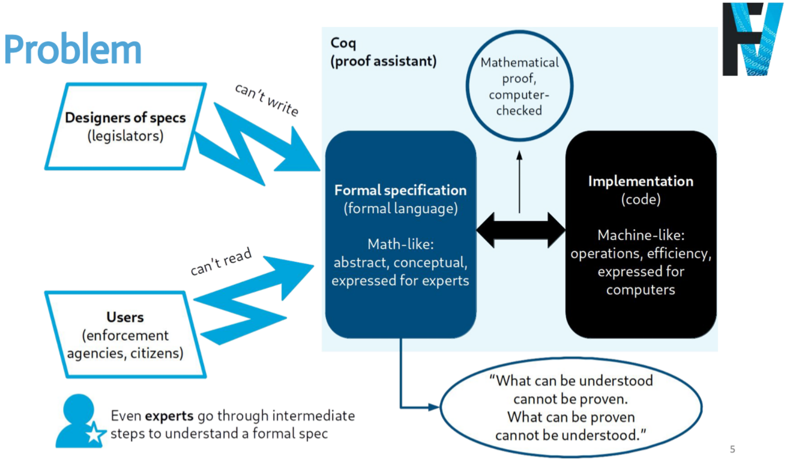

Formal verification
Coq
(proof assistant)
Mathematical proof, computer-checked
Formal specification
(formal language)
Math-like:
abstract, conceptual, expressed for experts
Implementation
(code)
Machine-like:
operations, efficiency, expressed for computers

What is verification?

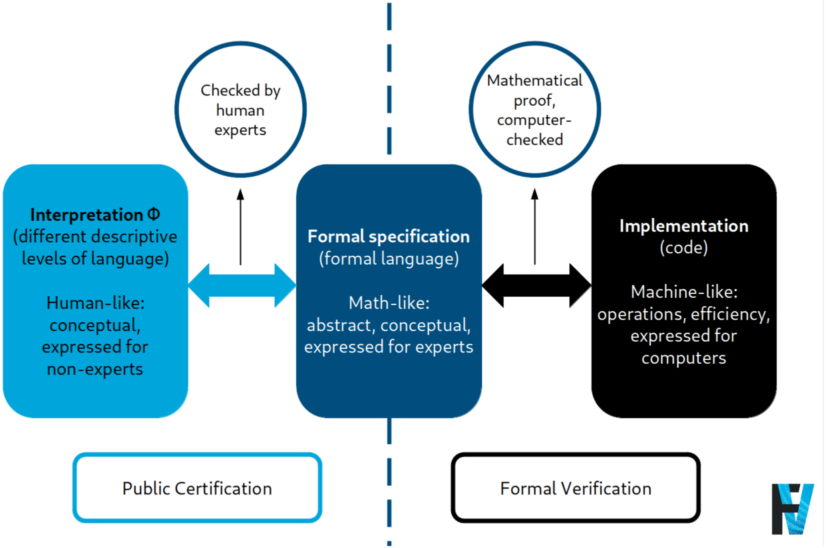
Formal verification


Coq
(proof assistant)
Mathematical proof, computer-checked
Extraction mechanism, soon formally verified
Formal specification
(formal language)
Math-like:
abstract, conceptual, expressed for experts
Implementation
(code)
Machine-like:
operations, efficiency, expressed for computers
Extracted code
OCaml or alike
Ready to run

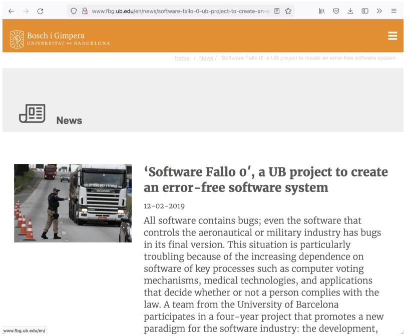
What is verification?


Problem
Designers of specs (legislators)
can't write
Users (enforcement agencies, citizens)
can't read
Even experts go through intermediate steps to understand a formal spec
Coq (proof assistant)
Mathematical proof, computer-checked
Formal specification (formal language)
Math-like: abstract, conceptual, expressed for experts
Implementation (code)
Machine-like: operations, efficiency, expressed for computers
"What can be understood cannot be proven. What can be proven cannot be understood."
5

What is verification?


Checked by human experts
Mathematical proof, computer-checked
Interpretation Φ (different descriptive levels of language)
Human-like: conceptual, expressed for non-experts
Formal specification (formal language)
Math-like: abstract, conceptual, expressed for experts
Implementation (code)
Machine-like: operations, efficiency, expressed for computers
Public Certification
Formal Verification

Over ten years of research in Barcelona



www.fbg.ub.edu/en/news/software-fallo-0-ub-project-to-create-an-e

Bosch i Gimpera
UNIVERSITAT DE BARCELONA

Home / News / 'Software Fallo 0', a UB project to create an error-free software system

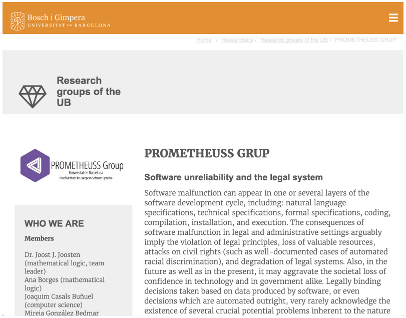
News

'Software Fallo 0', a UB project to create an error-free software system

12-02-2019

All software contains bugs; even the software that controls the aeronautical or military industry has bugs in its final version. This situation is particularly troubling because of the increasing dependence on software of key processes such as computer voting mechanisms, medical technologies, and applications that decide whether or not a person complies with the law. A team from the University of Barcelona participates in a four-year project that promotes a new paradigm for the software industry: the development,

www.fbg.ub.edu/en/



Bosch i Gimpera
UNIVERSITAT DE BARCELONA

Home / Researchers / Research groups of the UB / PROMETHEUSS GRUP

Research groups of the UB

PROMETHEUSS Group
Universitat de Barcelona
-Proof Methods for European Software Systems-

PROMETHEUSS GRUP

Software unreliability and the legal system

Software malfunction can appear in one or several layers of the software development cycle, including: natural language specifications, technical specifications, formal specifications, coding, compilation, installation, and execution. The consequences of software malfunction in legal and administrative settings arguably imply the violation of legal principles, loss of valuable resources, attacks on civil rights (such as well-documented cases of automated racial discrimination), and degradation of legal systems. Also, in the future as well as in the present, it may aggravate the societal loss of confidence in technology and in government alike. Legally binding decisions taken based on data produced by software, or even decisions which are automated outright, very rarely acknowledge the existence of several crucial potential problems inherent to the nature

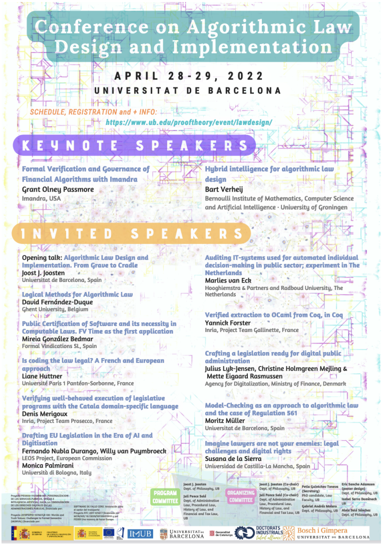
WHO WE ARE

Members

Dr. Joost J. Joosten (mathematical logic, team leader)
Ana Borges (mathematical logic)
Joaquim Casals Buñuel (computer science)
Mireia González Bedmar

Covenant between the University of Barcelona (FBG), Formal Vindications S.L. & Guretruck S.L.

Closing conference



Conference on Algorithmic Law Design and Implementation

APRIL 28-29, 2022
UNIVERSITAT DE BARCELONA

SCHEDULE, REGISTRATION and + INFO:
https://www.ub.edu/prooftheory/event/lawdesign/

KEYNOTE SPEAKERS

Formal Verification and Governance of Financial Algorithms with Imandra
Grant Olney Passmore
Imandra, USA

Hybrid intelligence for algorithmic law design
Bart Verheij
Bernoulli Institute of Mathematics, Computer Science and Artificial Intelligence · University of Groningen

INVITED SPEAKERS

Opening talk: Algorithmic Law Design and Implementation. From Grave to Cradle
Joost J. Joosten
Universitat de Barcelona, Spain

Logical Methods for Algorithmic Law
David Fernández-Duque
Ghent University, Belgium

Public Certification of Software and its necessity in Computable Laws. FV Time as the first application
Mireia González Bedmar
Formal Vindications SL, Spain

Is coding the law legal? A French and European approach
Liane Huttner
Université Paris 1 Panthéon-Sorbonne, France

Verifying well-behaved execution of legislative programs with the Catala domain-specific language
Denis Merigoux
Inria, Project Team Prosecco, France

Drafting EU Legislation in the Era of AI and Digitisation
Fernando Nubla Durango, Willy van Puymbroeck
LEOS Project, European Commission
Monica Palmirani
Università di Bologna, Italy

Auditing IT-systems used for automated individual decision-making in public sector; experiment in The Netherlands
Marlies van Eck
Hooghiemstra & Partners and Radboud University, The Netherlands

Verified extraction to OCaml from Coq, in Coq
Yannick Forster
Inria, Project Team Gallinette, France

Crafting a legislation ready for digital public administration
Julius Lyk-Jensen, Christine Holmgreen Mejling & Mette Eigaard Rasmussen
Agency for Digitalization, Ministry of Finance, Denmark

Model-Checking as an approach to algorithmic law and the case of Regulation 561
Moritz Müller
Universitat de Barcelona, Spain

Imagine lawyers are not your enemies: legal challenges and digital rights
Susana de la Sierra
Universidad de Castilla-La Mancha, Spain

PROGRAM COMMITTEE
Joost J. Joosten
Dept. of Philosophy, UB
Joli Ponce Solé
Dept. of Administrative Law, Procedural Law, History of Law, and Financial and Tax Law, UB

ORGANIZING COMMITTEE
Joost J. Joosten (Co-chair)
Dept. of Philosophy, UB
Juli Ponce Solé (Co-chair)
Dept. of Administrative Law, Procedural Law, History of Law, and Financial and Tax Law, UB
Petia Guintchev Toneva (Secretary)
PhD candidate, Law Faculty, UB
Gabriel Andrés Molero
Dept. of Philosophy, UB
Eric Sancho Adamson (poster design)
Dept. of Philosophy, UB
Isabel Serra Domènech
UB
Aleix Solé Sánchez
Dept. of Philosophy, UB

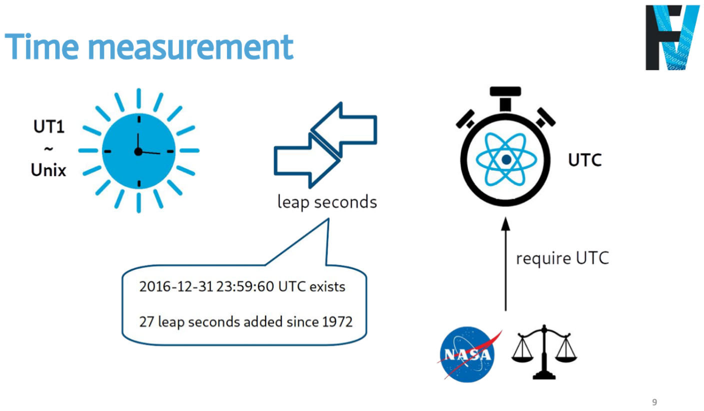
Proyecto PID2020-115774RB-I00: PERSONALIZACIÓN DE LOS SERVICIOS PÚBLICOS, SESGOS E INTELIGENCIA ARTIFICIAL: HACIA LA CONSOLIDACIÓN DE LOS DERECHOS DIGITALES EN LAS ADMINISTRACIONES PÚBLICAS, financiado por:

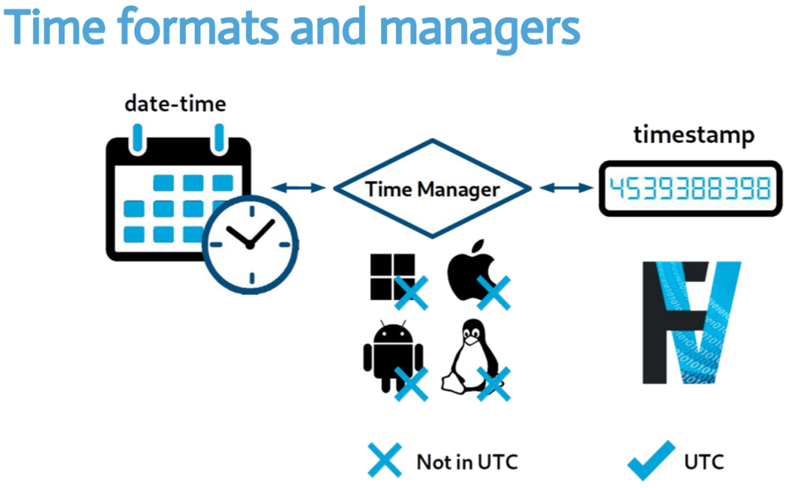
Proyecto 2019PID93-107667GB-I00: Worlds and Truth Values: Challenges to Formal Semantics (WORVAL) financiado por:

SOFTWARE DE FALLO CERO. Innovación para el sector del transporte
Proyecto RTC-2017-6740-7 financiado por MCIN/AEI/ 10.13039/501100011033 y por FEDER Una manera de hacer Europa

IMUB · UNIVERSITAT DE BARCELONA · Generalitat de Catalunya · DOCTORATS INDUSTRIALS talent alliance! · Bosch i Gimpera UNIVERSITAT DE BARCELONA

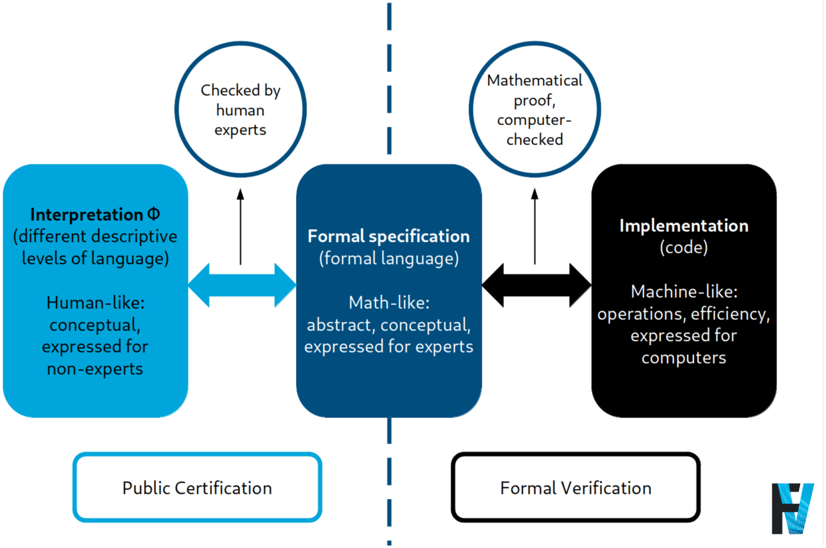
Time library

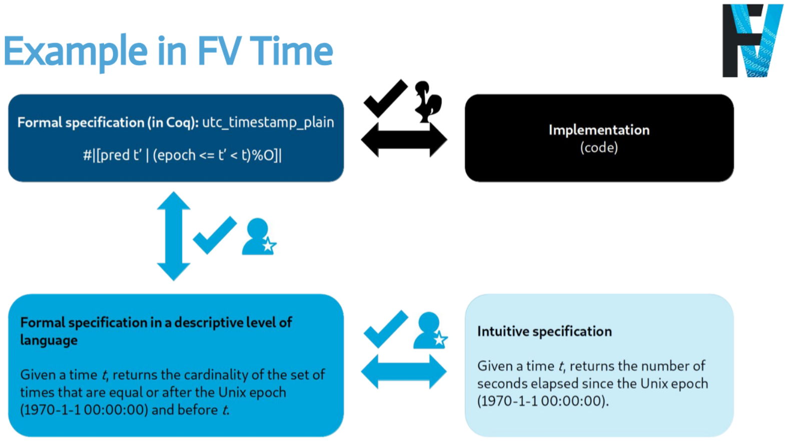

Time measurement
UT1
~
Unix
leap seconds
2016-12-31 23:59:60 UTC exists
27 leap seconds added since 1972
UTC
require UTC
NASA
9



Most important feature: formally verified!

What is certification?


Checked by human experts
Mathematical proof, computer-checked
Interpretation Φ (different descriptive levels of language)
Human-like: conceptual, expressed for non-experts
Formal specification (formal language)
Math-like: abstract, conceptual, expressed for experts
Implementation (code)
Machine-like: operations, efficiency, expressed for computers
Public Certification
Formal Verification



Example in FV Time

Formal specification (in Coq): utc_timestamp_plain

#|[pred t' | (epoch <= t' < t)%O]|

Implementation
(code)

Formal specification in a descriptive level of language

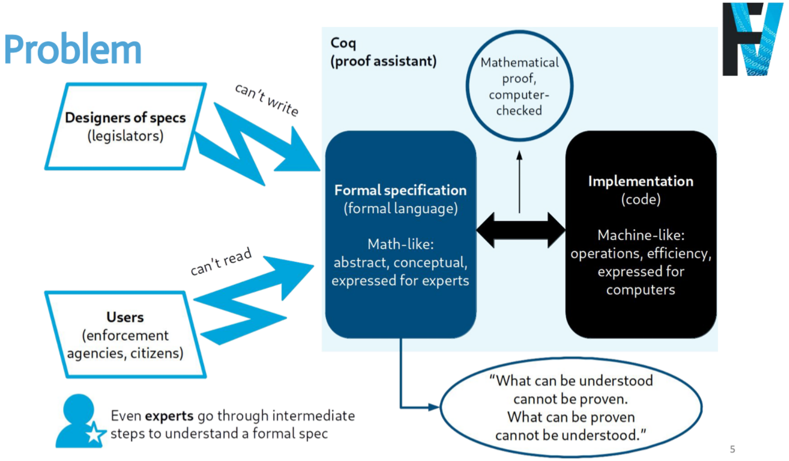
Given a time *t*, returns the cardinality of the set of times that are equal or after the Unix epoch (1970-1-1 00:00:00) and before *t*.

Intuitive specification

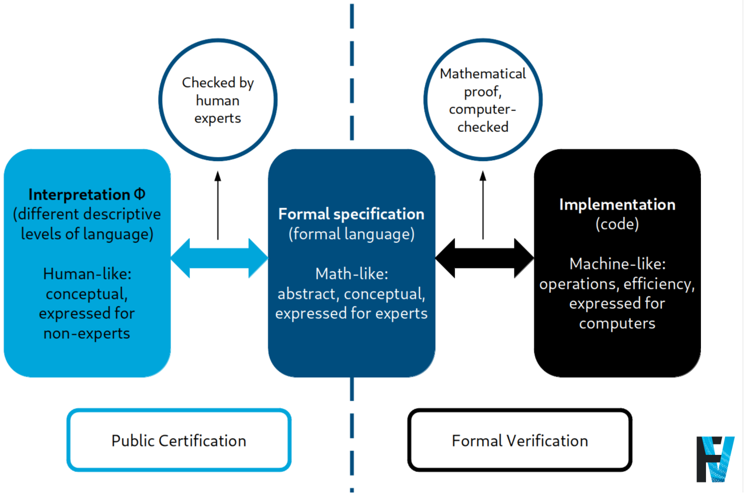
Given a time *t*, returns the number of seconds elapsed since the Unix epoch (1970-1-1 00:00:00).

Around one-thousand times more expensive!

A central problem



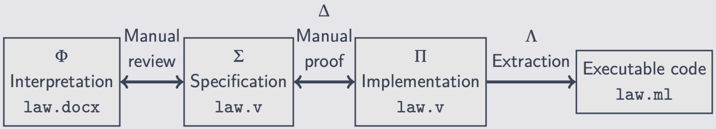
Public certification versus formal verification


Checked by human experts
Mathematical proof, computer-checked
Interpretation Φ (different descriptive levels of language)
Human-like: conceptual, expressed for non-experts
Formal specification (formal language)
Math-like: abstract, conceptual, expressed for experts
Implementation (code)
Machine-like: operations, efficiency, expressed for computers
Public Certification
Formal Verification

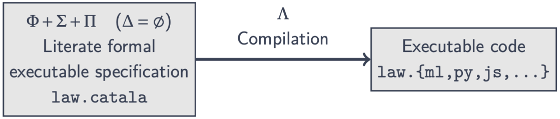
Catala: A Shortcut For Legal Expert System Certification

The Usual Way to Produce Verified Software

Using Mireia Gonzáles Bedmar's conceptual framework from yesterday's presentation:



Catala's approach:



8

Slides Catala: Merigoux

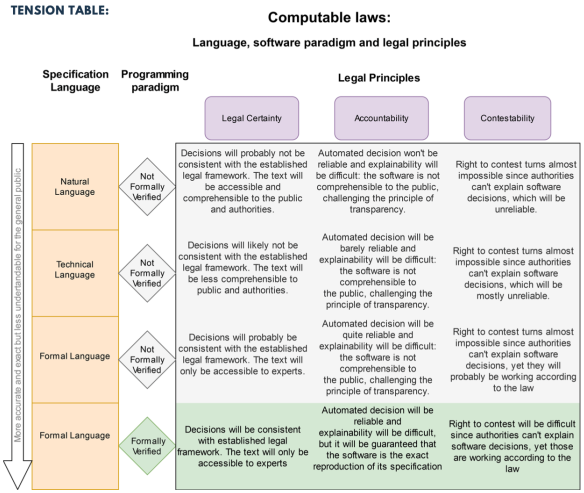
❶ Catala: A Language Reviewable by lawyers

US Tax Code, Section 132, (c)(1) Qualified employee discount

The term "qualified employee discount" means any employee discount with respect to qualified property or services to the extent such discount does not exceed—
(A) in the case of property, the gross profit percentage of the price at which the property is being offered by the employer to customers

```
scope QualifiedEmployeeDiscount :
  definition qualified_employee_discount
    under condition is_property consequence equals
      if employee_discount >$ customer_price *$ gross_profit_percentage then
        customer_price *$ gross_profit_percentage
      else employee_discount
```

Can code be the law?



TENSION TABLE:

Computable laws:

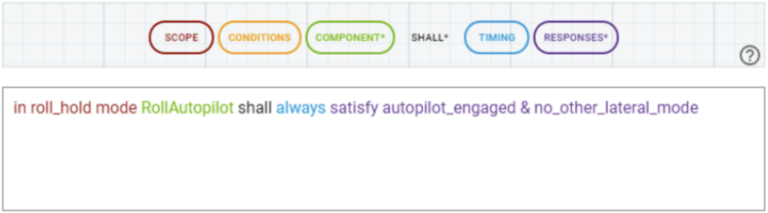
Language, software paradigm and legal principles

More accurate and exact but less undertandable for the general public

Specification Language	Programming paradigm	Legal Principles: Legal Certainty	Legal Principles: Accountability	Legal Principles: Contestability
Natural Language	Not Formally Verified	Decisions will probably not be consistent with the established legal framework. The text will be accessible and comprehensible to the public and authorities.	Automated decision won't be reliable and explainability will be difficult: the software is not comprehensible to the public, challenging the principle of transparency.	Right to contest turns almost impossible since authorities can't explain software decisions, which will be unreliable.
Technical Language	Not Formally Verified	Decisions will likely not be consistent with the established legal framework. The text will be less comprehensible to public and authorities.	Automated decision will be barely reliable and explainability will be difficult: the software is not comprehensible to the public, challenging the principle of transparency.	Right to contest turns almost impossible since authorities can't explain software decisions, which will be mostly unreliable.
Formal Language	Not Formally Verified	Decisions will probably be consistent with the established legal framework. The text will only be accessible to experts.	Automated decision will be quite reliable and explainability will be difficult: the software is not comprehensible to the public, challenging the principle of transparency.	Right to contest turns almost impossible since authorities can't explain software decisions, yet they will probably be working according to the law
Formal Language	Formally Verified	Decisions will be consistent with established legal framework. The text will only be accessible to experts	Automated decision will be reliable and explainability will be difficult, but it will be guaranteed that the software is the exact reproduction of its specification	Right to contest will be difficult since authorities can't explain software decisions, yet those are working according to the law

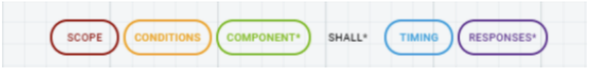
FRET: Formal Requirement Elicitation Tool

An attempt at bridging formal and natural language



in roll_hold mode RollAutopilot shall always satisfy autopilot_engaged & no_other_lateral_mode

FRET under the logic loupe: TIMING

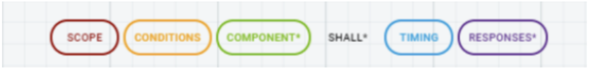


TIMING is optional and specifies when response is expected

- immediately
- never
- eventually (the default reading when Timing is omitted)
- always
- within *n* time units
- for *n* time units
- after *n* time units
- . . .

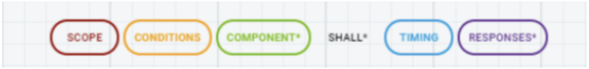
10 timing options

FRET under the logic loupe: SHALL



SHALL is mandatory

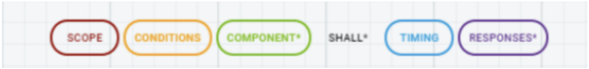
FRET under the logic loupe: COMPONENT



COMPONENT is mandatory

For example: The_Car

FRET under the logic loupe: CONDITION

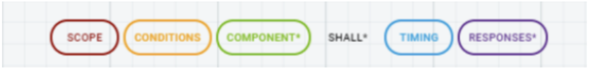


CONDITION is an optional feature: a Boolean expression

For example: When Traffic_Light_Is_Red & No_Police_Car_Nearby

Three options: void condition, trigger condition, continual

FRET under the logic loupe: SCOPE



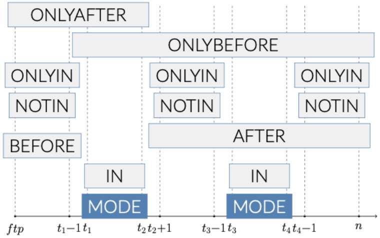
SCOPE is an optional feature: a finite collection of disjoint time intervals where the requirement is imposed

Built from a Boolean Mode M with an operator applied to it: $\mathcal{O}(M)$

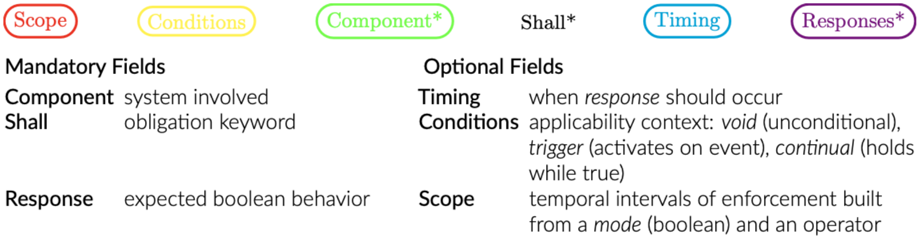
- before M
- after M
- in M
- not in M
- only before M
- only after M
- only in M
- global (default scope when omitted)

8 operators in total

FRET under the logic loupe: SCOPE

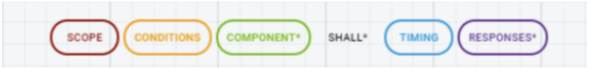


FRET under the logic loupe: summary



Mandatory Fields		Optional Fields	
Component	system involved	**Timing**	when *response* should occur
Shall	obligation keyword	**Conditions**	applicability context: *void* (unconditional), *trigger* (activates on event), *continual* (holds while true)
Response	expected boolean behavior	**Scope**	temporal intervals of enforcement built from a *mode* (boolean) and an operator

FRET under the logic loupe: Semantic templates



8 scope operators
Condition or no condition
7 Timing options
give rise to

$$8 \times 3 \times 10 = 240$$

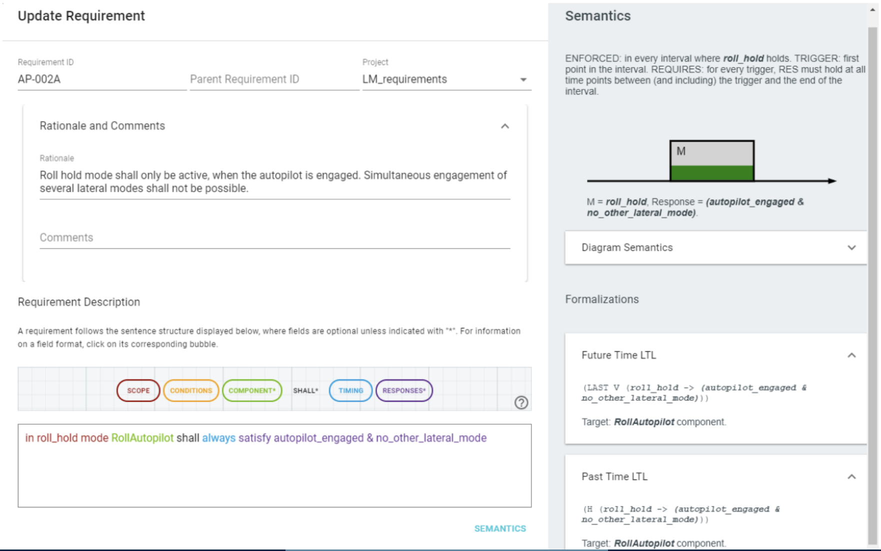
so-called semantic templates.
Each of the form

$$\mathsf{G}\Big(\mathcal{O}(M) \wedge \mathcal{C} \to \mathcal{T}(R)\Big)$$

This is a simplified representation of the fragment of LTL covered by this version of FRET*

* (there are some small letters)

FRET interface: various language levels



Update Requirement

Requirement ID: AP-002A
Parent Requirement ID
Project: LM_requirements

Rationale and Comments

Rationale: Roll hold mode shall only be active, when the autopilot is engaged. Simultaneous engagement of several lateral modes shall not be possible.

Comments

Requirement Description

A requirement follows the sentence structure displayed below, where fields are optional unless indicated with "*". For information on a field format, click on its corresponding bubble.

in roll_hold mode RollAutopilot shall always satisfy autopilot_engaged & no_other_lateral_mode

SEMANTICS

Semantics

ENFORCED: in every interval where ***roll_hold*** holds. TRIGGER: first point in the interval. REQUIRES: for every trigger, RES must hold at all time points between (and including) the trigger and the end of the interval.

M = ***roll_hold***, Response = ***(autopilot_engaged & no_other_lateral_mode)***.

Diagram Semantics

Formalizations

Future Time LTL

```
(LAST V (roll_hold -> (autopilot_engaged &
no_other_lateral_mode)))
```

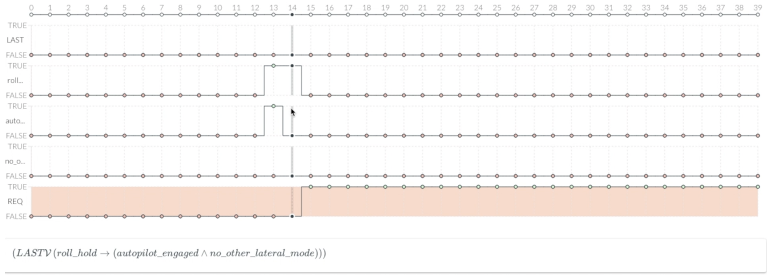
Target: ***RollAutopilot*** component.

Past Time LTL

```
(H (roll_hold -> (autopilot_engaged &
no_other_lateral_mode)))
```

Target: ***RollAutopilot*** component.

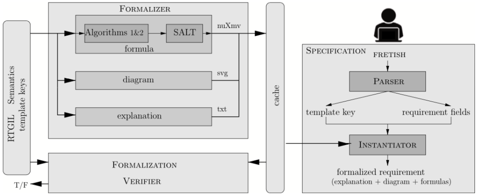
FRET: interactive sample testing



$(LASTV\,(roll\_hold \to (autopilot\_engaged \wedge no\_other\_lateral\_mode)))$

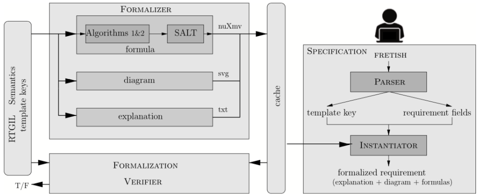
Users can test the behaviour of their FRET phrases

Serendipity has it...



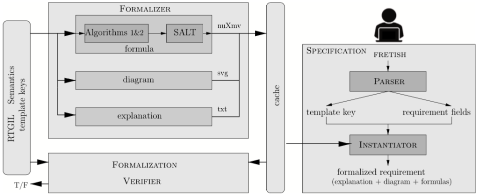
- It seems that they wanted to tap into the RTGIL tool (Real Time Graphical Interval Logic)

Serendipity has it...



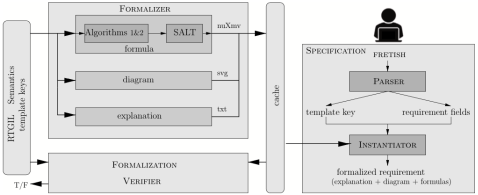
- It seems that they wanted to tap into the RTGIL tool (Real Time Graphical Interval Logic)
- which I think communicates well with SALT (Structured Assertion Language for Temporal Logics)

Serendipity has it...



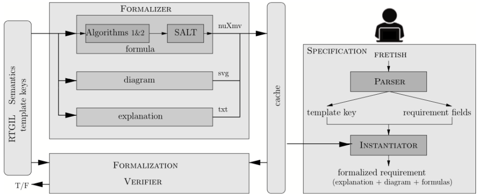
- It seems that they wanted to tap into the RTGIL tool (Real Time Graphical Interval Logic)
- which I think communicates well with SALT (Structured Assertion Language for Temporal Logics)
- and nuXmv

Serendipity has it...



- It seems that they wanted to tap into the RTGIL tool (Real Time Graphical Interval Logic)
- which I think communicates well with SALT (Structured Assertion Language for Temporal Logics)
- and nuXmv
- which is why they have curious ways of dealing with scopes (in my opinion)

Serendipity has it...



- It seems that they wanted to tap into the RTGIL tool (Real Time Graphical Interval Logic)
- which I think communicates well with SALT (Structured Assertion Language for Temporal Logics)
- and nuXmv
- which is why they have curious ways of dealing with scopes (in my opinion)
- which make their way through the LTL translation it seems

A first translation algorithm

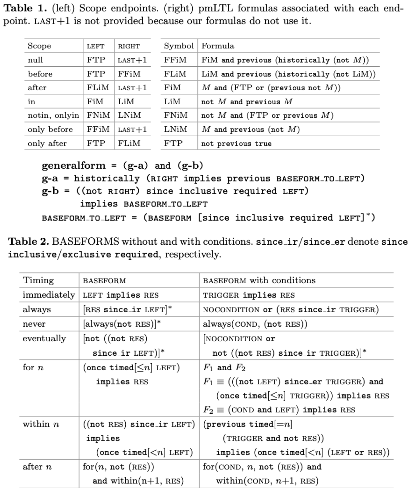


Table 1. (left) Scope endpoints. (right) pmLTL formulas associated with each endpoint. LAST+1 is not provided because our formulas do not use it.

Scope	LEFT	RIGHT
null	FTP	LAST+1
before	FTP	FFiM
after	FLiM	LAST+1
in	FiM	LiM
notin, onlyin	FNiM	LNiM
only before	FFiM	LAST+1
only after	FTP	FLiM

Symbol	Formula
FFiM	FiM and previous (historically (not M))
FLiM	LiM and previous (historically (not LiM))
FiM	M and (FTP or (previous not M))
LiM	not M and previous M
FNiM	not M and (FTP or previous M)
LNiM	M and previous (not M)
FTP	not previous true

```
generalform = (g-a) and (g-b)
g-a = historically (RIGHT implies previous BASEFORM_TO_LEFT)
g-b = ((not RIGHT) since inclusive required LEFT)
          implies BASEFORM_TO_LEFT
BASEFORM_TO_LEFT = (BASEFORM [since inclusive required LEFT]*)
```

Table 2. BASEFORMS without and with conditions. since_ir/since_er denote since inclusive/exclusive required, respectively.

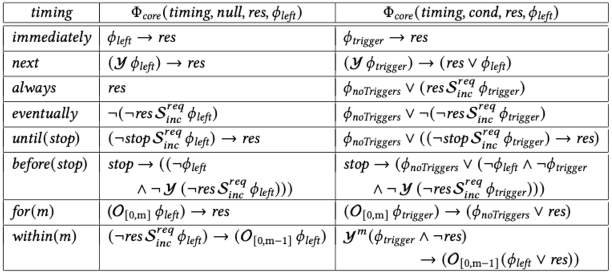
Timing	BASEFORM	BASEFORM with conditions
immediately	LEFT implies RES	TRIGGER implies RES
always	[RES since_ir LEFT]*	NOCONDITION or (RES since_ir TRIGGER)
never	[always(not RES)]*	always(COND, (not RES))
eventually	[not ((not RES) since_ir LEFT)]*	[NOCONDITION or not ((not RES) since_ir TRIGGER)]*
for n	(once timed[$\leq n$] LEFT) implies RES	F_1 and F_2 $F_1 \equiv$ (((not LEFT) since_er TRIGGER) and (once timed[$\leq n$] TRIGGER)) implies RES $F_2 \equiv$ (COND and LEFT) implies RES
within n	((not RES) since_ir LEFT) implies (once timed[$<n$] LEFT)	(previous timed[$=n$] (TRIGGER and not RES)) implies (once timed[$<n$] (LEFT or RES))

after n	for(n, not (RES))
	and within(n+1, RES)
		for(COND, n, not (RES)) and
		within(COND, n+1, RES)

$$\mathsf{G}\Big(\mathcal{O}(M)\wedge\mathcal{C}\to\mathcal{T}(R)\Big)$$

Second: slightly more promising

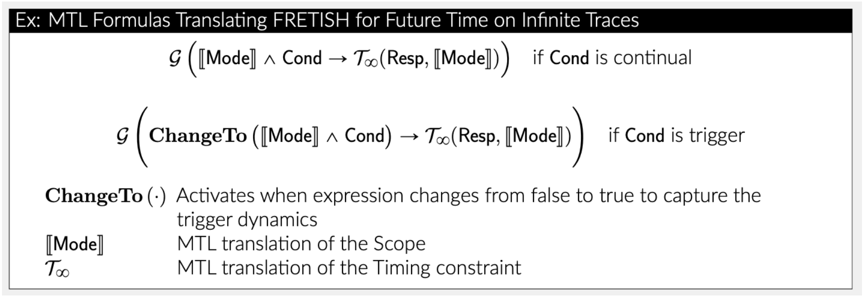
Table 2. Core formula definition for *null* condition or $cond \in \mathbb{B}$.



$timing$	$\Phi_{core}(timing, null, res, \phi_{left})$	$\Phi_{core}(timing, cond, res, \phi_{left})$
$immediately$	$\phi_{left} \rightarrow res$	$\phi_{trigger} \rightarrow res$
$next$	$(\mathcal{Y}\, \phi_{left}) \rightarrow res$	$(\mathcal{Y}\, \phi_{trigger}) \rightarrow (res \vee \phi_{left})$
$always$	res	$\phi_{noTriggers} \vee (res\, \mathcal{S}^{req}_{inc}\, \phi_{trigger})$
$eventually$	$\neg(\neg res\, \mathcal{S}^{req}_{inc}\, \phi_{left})$	$\phi_{noTriggers} \vee \neg(\neg res\, \mathcal{S}^{req}_{inc}\, \phi_{trigger})$
$until(stop)$	$(\neg stop\, \mathcal{S}^{req}_{inc}\, \phi_{left}) \rightarrow res$	$\phi_{noTriggers} \vee ((\neg stop\, \mathcal{S}^{req}_{inc}\, \phi_{trigger}) \rightarrow res)$
$before(stop)$	$stop \rightarrow ((\neg\phi_{left} \wedge \neg\, \mathcal{Y}\, (\neg res\, \mathcal{S}^{req}_{inc}\, \phi_{left})))$	$stop \rightarrow (\phi_{noTriggers} \vee (\neg\phi_{left} \wedge \neg\phi_{trigger} \wedge \neg\, \mathcal{Y}\, (\neg res\, \mathcal{S}^{req}_{inc}\, \phi_{trigger})))$
$for(m)$	$(O_{[0,m]}\, \phi_{left}) \rightarrow res$	$(O_{[0,m]}\, \phi_{trigger}) \rightarrow (\phi_{noTriggers} \vee res)$
$within(m)$	$(\neg res\, \mathcal{S}^{req}_{inc}\, \phi_{left}) \rightarrow (O_{[0,m-1]}\, \phi_{left})$	$\mathcal{Y}^m(\phi_{trigger} \wedge \neg res) \rightarrow (O_{[0,m-1]}(\phi_{left} \vee res))$

$$\mathsf{G}\Big(\mathcal{O}(M) \wedge \mathcal{C} \rightarrow \mathcal{T}(R)\Big)$$

Simplified architecture



Ex: MTL Formulas Translating FRETISH for Future Time on Infinite Traces

$$\mathcal{G}\left(\llbracket \mathsf{Mode} \rrbracket \wedge \mathsf{Cond} \rightarrow \mathcal{T}_\infty(\mathsf{Resp}, \llbracket \mathsf{Mode} \rrbracket)\right) \quad \text{if } \mathsf{Cond} \text{ is continual}$$

$$\mathcal{G}\left(\mathbf{ChangeTo}\left(\llbracket \mathsf{Mode} \rrbracket \wedge \mathsf{Cond}\right) \rightarrow \mathcal{T}_\infty(\mathsf{Resp}, \llbracket \mathsf{Mode} \rrbracket)\right) \quad \text{if } \mathsf{Cond} \text{ is trigger}$$

$\mathbf{ChangeTo}(\cdot)$	Activates when expression changes from false to true to capture the trigger dynamics
$\llbracket \mathsf{Mode} \rrbracket$	MTL translation of the Scope
$\mathcal{T}_\infty$	MTL translation of the Timing constraint

Simplified architecture: lookup tables

Ordinal	Scope operator	LTL definition
0	Global **Mode**	$\top$
1	In **Mode**	**Mode**
2	Not In **Mode**	$\neg$ **Mode**
3	Only In **Mode**	$\neg$ **Mode**
4	Before **Mode**	$\mathcal{H}\neg$ **Mode**
5	Only Before **Mode**	$\mathcal{O}$ **Mode**
6	After **Mode**	$\mathcal{O}(\neg\textbf{Mode} \wedge \mathcal{Y}\,\textbf{Mode})$
7	Only After **Mode**	$\mathcal{H}(\mathcal{Y}\,\textbf{Mode} \to \textbf{Mode})$

(a) Look-up table for FRETISH Scope

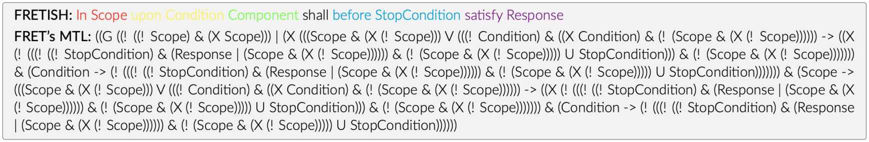
ID	Condition kind
0	void
1	continual
2	trigger

(b) Look-up table for FRETISH Condition

ID	FRETISH Timing	MTL formula
0	eventually	$\llbracket\text{Mode}\rrbracket\ \mathcal{U}\ \big(\text{Resp} \wedge \llbracket\text{Mode}\rrbracket\big)$
1	immediately	Resp
2	next	$\mathcal{X}\,\text{Resp} \vee \mathcal{X}\,\textbf{ChangeTo}\,\neg\llbracket\text{Mode}\rrbracket$
3	always	$(\mathcal{X}\,\textbf{ChangeTo}\,\neg\llbracket\text{Mode}\rrbracket)\ \mathcal{R}\ \text{Resp}$
4	never	$(\mathcal{X}\,\textbf{ChangeTo}\,\neg\llbracket\text{Mode}\rrbracket)\ \mathcal{R}\ \neg\text{Resp}$
5	within (d)	$\Diamond_{[0,d]}\text{Resp} \vee \big(\llbracket\text{Mode}\rrbracket\ \mathcal{U}_{[0,d]}\neg\llbracket\text{Mode}\rrbracket\big)$
6	for (d)	$\Box_{[0,d]}\text{Resp} \vee \big((\text{Resp} \wedge \llbracket\text{Mode}\rrbracket)\ \mathcal{U}_{[0,d]}\neg\llbracket\text{Mode}\rrbracket\big)$
7	after(d)	$\Big(\Box_{[0,d]}\neg\text{Resp} \wedge \Diamond_{[d+1,d+1]}\text{Resp}\Big) \vee \Big((\neg\text{Resp} \wedge \llbracket\text{Mode}\rrbracket)\ \mathcal{U}_{[0,d+1]}\neg\llbracket\text{Mode}\rrbracket\Big)$
8	until (stopCond)	$\Box\text{Resp} \vee \Big(\text{Resp}\ \mathcal{U}\ \big(\text{stopCond} \vee \textbf{StrictChangeTo}\,(\neg\llbracket\text{Mode}\rrbracket)\big)\Big)$
9	before (stopCond)	$\big(\text{Resp}\ \mathcal{R}\ \neg\text{stopCond}\big) \vee \Big(\big(\llbracket\text{Mode}\rrbracket \wedge \mathcal{X}\,\neg\llbracket\text{Mode}\rrbracket\big)\ \mathcal{R}\ \neg\text{stopCond}\Big)$

(c) Look-up table for FRETISH Timing for infinite trace when the Scope does not include "only".

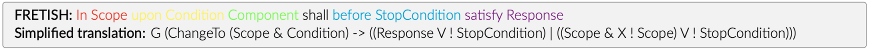
Comparison and Occam



FRETISH: In Scope upon Condition Component shall before StopCondition satisfy Response

FRET's MTL: ((G ((! ((! Scope) & (X Scope))) | (X (((Scope & (X (! Scope))) V (((! Condition) & ((X Condition) & (! (Scope & (X (! Scope)))))) -> ((X (! (((! ((! StopCondition) & (Response | (Scope & (X (! Scope)))))) & (! (Scope & (X (! Scope))))) U StopCondition))) & (! (Scope & (X (! Scope))))))) & (Condition -> (! (((! ((! StopCondition) & (Response | (Scope & (X (! Scope)))))) & (! (Scope & (X (! Scope))))) U StopCondition))))))) & (Scope -> (((Scope & (X (! Scope))) V (((! Condition) & ((X Condition) & (! (Scope & (X (! Scope)))))) -> ((X (! (((! ((! StopCondition) & (Response | (Scope & (X (! Scope)))))) & (! (Scope & (X (! Scope))))) U StopCondition))) & (! (Scope & (X (! Scope))))))) & (Condition -> (! (((! ((! StopCondition) & (Response | (Scope & (X (! Scope)))))) & (! (Scope & (X (! Scope))))) U StopCondition))))))

FRET translation



FRETISH: In Scope upon Condition Component shall before StopCondition satisfy Response
Simplified translation: G (ChangeTo (Scope & Condition) -> ((Response V ! StopCondition) | ((Scope & X ! Scope) V ! StopCondition)))

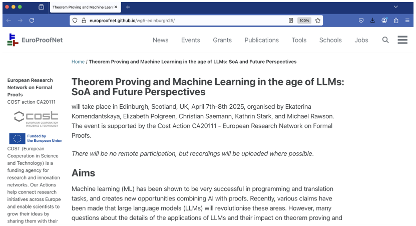
Our simplified translation (preparing Rocq implementation)

We currently compare test-case generation performance.

Spurious findings

- **Counter-intuitive constructs** whose informal interpretation diverges from their actual formal semantics, often leading to misunderstandings:
  - TheParcel shall within 1 day satisfy BeDelivered $\not\Rightarrow$ TheParcel shall eventually satisfy BeDelivered
  - **'Only In' Scopes defy logic**: Their semantics aren't logically grounded, but come from engineering intuition.
  - TheDriver shall after 3 hours of driving satisfy Rest forces that the rest cannot occur before 3 hours of driving.

Non-sharing incentives


Theorem Proving and Machine Learn
europroofnet.github.io/wg5-edinburgh25/
EuroProofNet
News
Events
Grants
Publications
Tools
Schools
Jobs

Home / Theorem Proving and Machine Learning in the age of LLMs: SoA and Future Perspectives

European Research Network on Formal Proofs
COST action CA20111

Funded by the European Union

COST (European Cooperation in Science and Technology) is a funding agency for research and innovation networks. Our Actions help connect research initiatives across Europe and enable scientists to grow their ideas by sharing them with their

Theorem Proving and Machine Learning in the age of LLMs: SoA and Future Perspectives

will take place in Edinburgh, Scotland, UK, April 7th-8th 2025, organised by Ekaterina Komendantskaya, Elizabeth Polgreen, Christian Saemann, Kathrin Stark, and Michael Rawson. The event is supported by the Cost Action CA20111 - European Research Network on Formal Proofs.

There will be no remote participation, but recordings will be uploaded where possible.

Aims

Machine learning (ML) has been shown to be very successful in programming and translation tasks, and creates new opportunities combining AI with proofs. Recently, various claims have been made that large language models (LLMs) will revolutionise these areas. However, many questions about the details of the applications of LLMs and their impact on theorem proving and

16th century protagonists: Gerolamo Cardano, Niccolò Fontana (Tartaglia (stammerer)), Scipione del Ferro, Ludovico Ferrari, etc.

21st century protagonist: ChatGTP, DeepSeek, Mistral, Claude, ...

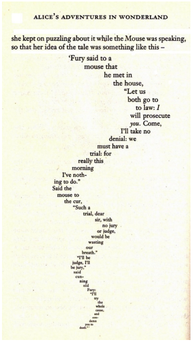
Belief revision appreciation revision

Succes from information theory: Self-attention with triple (queries, keys, values) in encoders and decoders

Controlled Natural Languages are promising to bridge reasoning and LLMs.

FRET simplification apology: The simpler they are, the better, the closer the formalisation to the CNL the better

Applied Logic in Law



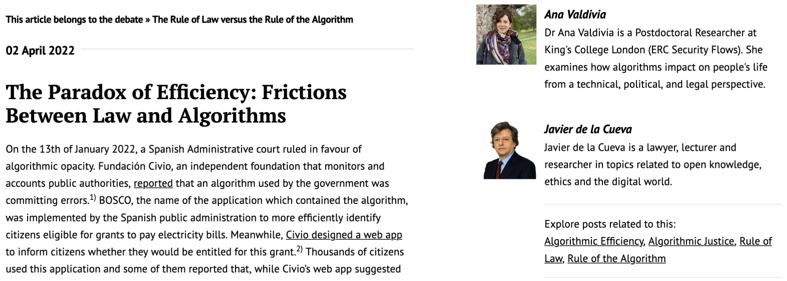
ALICE'S ADVENTURES IN WONDERLAND

she kept on puzzling about it while the Mouse was speaking, so that her idea of the tale was something like this –

'Fury said to a
mouse that
he met in
the house,
"Let us
both go to
to law: *I*
will prosecute
you. Come,
I'll take no
denial: we
must have a
trial: for
really this
morning
I've noth-
ing to do."
Said the
mouse to
the cur,
"Such a
trial, dear
sir, with
no jury
or judge,
would be
wasting
our
breath."
"I'll be
judge, I'll
be jury,"
said
cun-
ning
old
Fury:
"I'll
try
the
whole
cause,
and
con-
demn
you to
death." '

It keeps the logician off the street

Fury said to the mouse: Civio vs Bosco


This article belongs to the debate » The Rule of Law versus the Rule of the Algorithm

02 April 2022

The Paradox of Efficiency: Frictions Between Law and Algorithms

On the 13th of January 2022, a Spanish Administrative court ruled in favour of algorithmic opacity. Fundación Civio, an independent foundation that monitors and accounts public authorities, reported that an algorithm used by the government was committing errors.[1] BOSCO, the name of the application which contained the algorithm, was implemented by the Spanish public administration to more efficiently identify citizens eligible for grants to pay electricity bills. Meanwhile, Civio designed a web app to inform citizens whether they would be entitled for this grant.[2] Thousands of citizens used this application and some of them reported that, while Civio's web app suggested

Ana Valdivia
Dr Ana Valdivia is a Postdoctoral Researcher at King's College London (ERC Security Flows). She examines how algorithms impact on people's life from a technical, political, and legal perspective.

Javier de la Cueva
Javier de la Cueva is a lawyer, lecturer and researcher in topics related to open knowledge, ethics and the digital world.

Explore posts related to this:
Algorithmic Efficiency, Algorithmic Justice, Rule of Law, Rule of the Algorithm

The Bosco computer program : errors in the computation of the social welfare bonuses
Least requirement: access to source code
In France it is mandatory to publish source code of software that is used in public administration.
However, access to source code will not resolve all problems

Thanks