Legal certified software through proof assistants: a global scientific and managerial perspective

EuroProofNet

Joost J. Joosten + Team

University of Barcelona

Valencia, Friday 11-02-2022

The Formal Verification Team

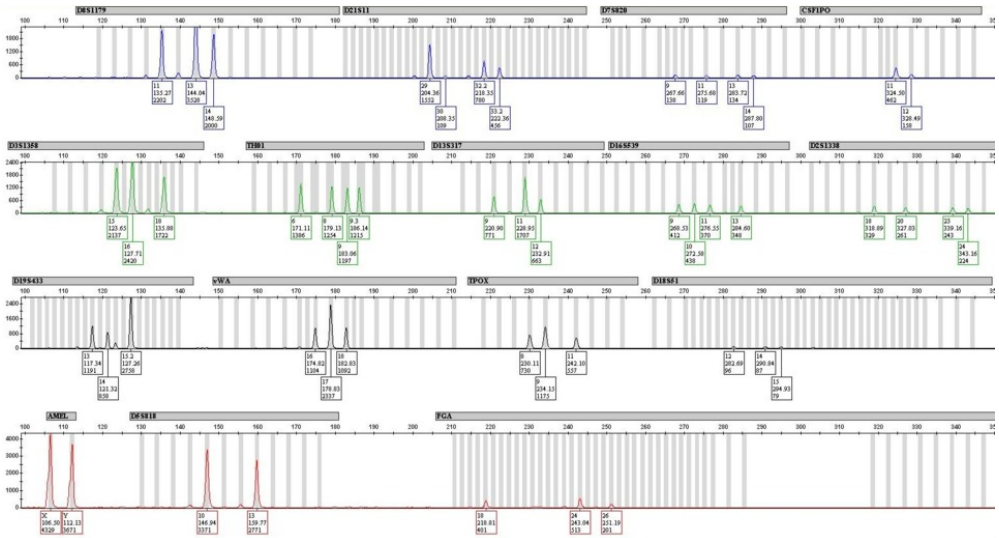
- The team is based in the University of Barcelona, and works conjointly with Formal Vindications S.L. and Guretruck S.L.
- Ana de Almeida Borges, Joaquim Buñuel Casals, Eduardo Hermo Reyes, Mireia González Bedmar, among others.

Project funded by the "Ministry of Science, Innovation and Universities", the "State Agency for Research" and the "European Regional Development Fund" (ERDF)

Computer evidence

Legal examples concerning unreliable software:

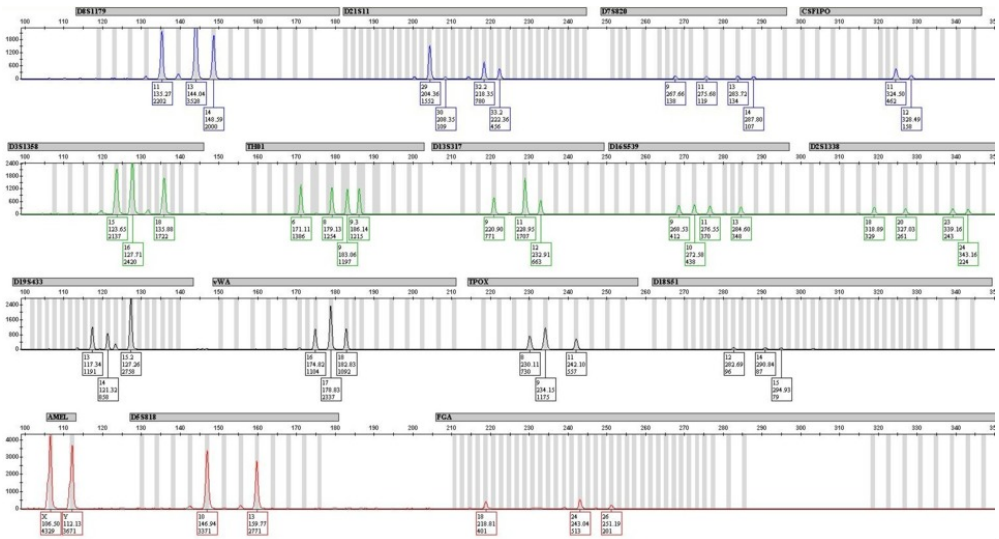
- In three cases, judges from USA have petitioned to **make the software open to the public** from proprietary DNA comparing software, due to some unreliable results:



Computer evidence

Legal examples concerning unreliable software:

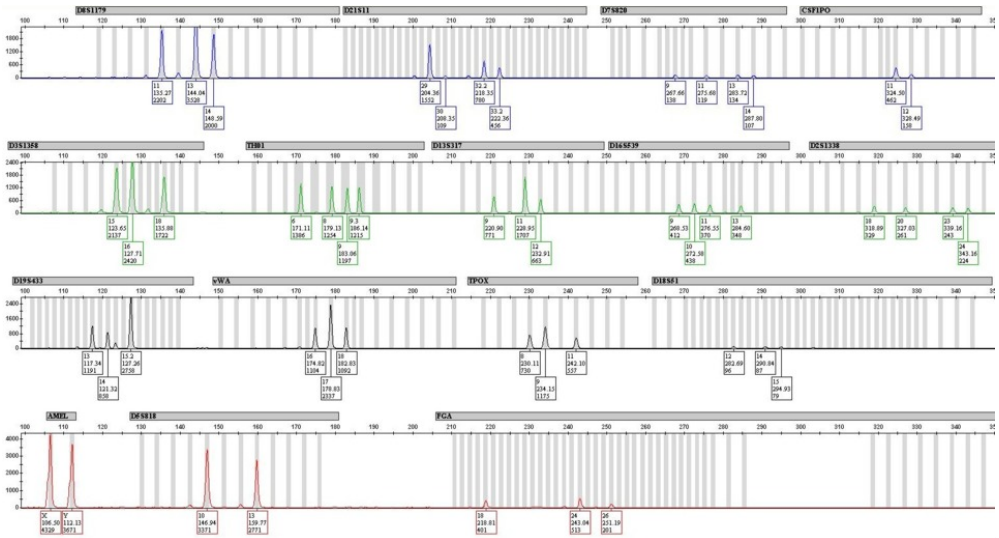
- In three cases, judges from USA have petitioned to **make the software open to the public** from proprietary DNA comparing software, due to some unreliable results:
  - STRmix



Computer evidence

Legal examples concerning unreliable software:

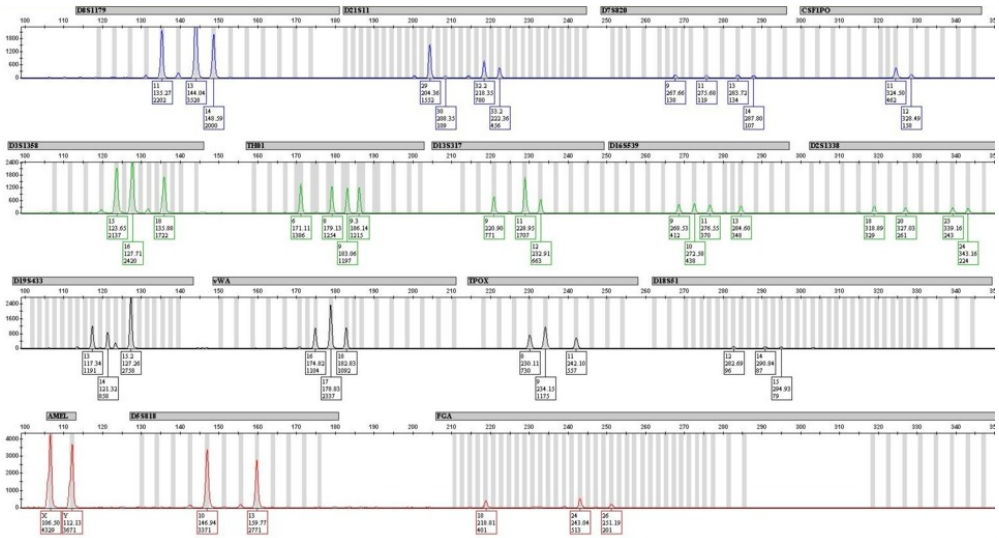
- In three cases, judges from USA have petitioned to **make the software open to the public** from proprietary DNA comparing software, due to some unreliable results:
  - STRmix
  - FST



Computer evidence

Legal examples concerning unreliable software:

- In three cases, judges from USA have petitioned to **make the software open to the public** from proprietary DNA comparing software, due to some unreliable results:
  - STRmix
  - FST
  - TrueAllele (still proprietary)



- In Valladolid, the court decided that a fine was not to be issued because the software involved **was not homologated**.

- In Valladolid, the court decided that a fine was not to be issued because the software involved **was not homologated**.
- N. Sentence: 30/2019, CONTENCIOSO/ADMTVO court. N. 4 of Valladolid (Spain)

ADMINISTRACION DE JUSTICIA

infracción imputada y sancionada en cuanto que no se han incumplido los tiempos de descanso semanales.

En segundo lugar considera que los hechos denunciados no están suficientemente probados a efectos de poderlos considerar constitutivos de la infracción sancionada. En este apartado señala que el tacógrafo del que se han obtenido datos tiene una programación o configuración de fábrica que adolece de errores y que hace que sus resultados no sean fiables ni ciertos. No se trata de una avería o de un mal funcionamiento sino de errores de fabricación, configuración y/o programación llamando la atención sobre la falta de homologación del tacógrafo y, especialmente, del software utilizado dentro del mismo. A lo anterior añade que no consta, y por lo tanto falta, la homologación del software utilizado por las autoridades para obtener y procesar los datos registrados en el tacógrafo.

Se acepta lo alegado por la parte demandante en lo que se refiere a la ausencia de prueba de cargo suficiente respecto al software utilizado por la autoridad correspondiente para obtener los datos registrados en el tacógrafo por lo que, sin necesidad de analizar el resto de la fundamentación jurídica

Our test case

- European transport regulations: Regulation (EU) 2016/799 and Regulation (EC) 561/2006

Our test case

- European transport regulations: Regulation (EU) 2016/799 and Regulation (EC) 561/2006
- Stipulate driving and resting requirements;

Our test case

- European transport regulations: Regulation (EU) 2016/799 and Regulation (EC) 561/2006
- Stipulate driving and resting requirements;
- Idea: create certified zero-error software to avoid mishaps;

Our test case

- European transport regulations: Regulation (EU) 2016/799 and Regulation (EC) 561/2006
- Stipulate driving and resting requirements;
- Idea: create certified zero-error software to avoid mishaps;
- Findings:

Our test case

- European transport regulations: Regulation (EU) 2016/799 and Regulation (EC) 561/2006
- Stipulate driving and resting requirements;
- Idea: create certified zero-error software to avoid mishaps;
- Findings:
  1. Ambiguities in texts;

Our test case

- European transport regulations: Regulation (EU) 2016/799 and Regulation (EC) 561/2006
- Stipulate driving and resting requirements;
- Idea: create certified zero-error software to avoid mishaps;
- Findings:
  1. Ambiguities in texts;
  2. Various different proprietary softwares have different implementations;

Our test case

- European transport regulations: Regulation (EU) 2016/799 and Regulation (EC) 561/2006
- Stipulate driving and resting requirements;
- Idea: create certified zero-error software to avoid mishaps;
- Findings:
  1. Ambiguities in texts;
  2. Various different proprietary softwares have different implementations;
  3. Tons of fines were erroneous and successfully rebutted;

Our test case

- European transport regulations: Regulation (EU) 2016/799 and Regulation (EC) 561/2006
- Stipulate driving and resting requirements;
- Idea: create certified zero-error software to avoid mishaps;
- Findings:
  1. Ambiguities in texts;
  2. Various different proprietary softwares have different implementations;
  3. Tons of fines were erroneous and successfully rebutted;
  4. Undefined/underspecified ontologies ('continuous driving time');

Our test case

- European transport regulations: Regulation (EU) 2016/799 and Regulation (EC) 561/2006
- Stipulate driving and resting requirements;
- Idea: create certified zero-error software to avoid mishaps;
- Findings:
  1. Ambiguities in texts;
  2. Various different proprietary softwares have different implementations;
  3. Tons of fines were erroneous and successfully rebutted;
  4. Undefined/underspecified ontologies ('continuous driving time');
  5. Differences in translations;

Our test case

- European transport regulations: Regulation (EU) 2016/799 and Regulation (EC) 561/2006
- Stipulate driving and resting requirements;
- Idea: create certified zero-error software to avoid mishaps;
- Findings:
  1. Ambiguities in texts;
  2. Various different proprietary softwares have different implementations;
  3. Tons of fines were erroneous and successfully rebutted;
  4. Undefined/underspecified ontologies ('continuous driving time');
  5. Differences in translations;
  6. Non-locality;

Our test case

- European transport regulations: Regulation (EU) 2016/799 and Regulation (EC) 561/2006
- Stipulate driving and resting requirements;
- Idea: create certified zero-error software to avoid mishaps;
- Findings:
  1. Ambiguities in texts;
  2. Various different proprietary softwares have different implementations;
  3. Tons of fines were erroneous and successfully rebutted;
  4. Undefined/underspecified ontologies ('continuous driving time');
  5. Differences in translations;
  6. Non-locality;
  7. Non shift-invariance (leap seconds count!);

Our test case

- European transport regulations: Regulation (EU) 2016/799 and Regulation (EC) 561/2006
- Stipulate driving and resting requirements;
- Idea: create certified zero-error software to avoid mishaps;
- Findings:
  1. Ambiguities in texts;
  2. Various different proprietary softwares have different implementations;
  3. Tons of fines were erroneous and successfully rebutted;
  4. Undefined/underspecified ontologies ('continuous driving time');
  5. Differences in translations;
  6. Non-locality;
  7. Non shift-invariance (leap seconds count!);
- We decided to isolate a 'non-problematic' part: Time Library

The FV Time Library

- Translation between times and timestamps

  $$\text{2022-02-11 10:20:00 UTC} \leftrightarrow 1644574827$$

- Timestamp: number of seconds since 1970-01-01 00:00:00, including leap seconds
- Time arithmetic:

  $$\text{2022-02-11 10:20:00} + 5 \text{ minutes} = \text{2022-02-11 10:25:00}$$

- Formalized in Coq
- Simple specifications, efficient implementations

Challenges

- Building a bridge (refinement) between proof-friendly types (unary `nat`) and computation and extraction-friendly types (primitive int)
- Proving bounded arithmetical statements when there is no better proof than checking every case

$$(\forall x < 400)(\forall y < 300)\, f(x, y) = 0$$

Challenges

- Building a bridge (refinement) between proof-friendly types (unary `nat`) and computation and extraction-friendly types (primitive int)
- Proving bounded arithmetical statements when there is no better proof than checking every case

$$(\forall x < 400)(\forall y < 300)\, f(x, y) = 0$$

Our solutions led to nice tools in their own right.

`http://formalvindications.com/work/`

Ingredients for software homologation

- Where (in what language) is the regulation specified?

Ingredients for software homologation

- Where (in what language) is the regulation specified?
- A formal specification should meet certain mathematical/computational requirements

Ingredients for software homologation

- Where (in what language) is the regulation specified?
- A formal specification should meet certain mathematical/computational requirements
  - Unique readability;

Ingredients for software homologation

- Where (in what language) is the regulation specified?
- A formal specification should meet certain mathematical/computational requirements
  - Unique readability;
  - Understandable to both lawyers and programmers;

Ingredients for software homologation

- Where (in what language) is the regulation specified?
- A formal specification should meet certain mathematical/computational requirements
  - Unique readability;
  - Understandable to both lawyers and programmers;
  - Consistency;

Ingredients for software homologation

- Where (in what language) is the regulation specified?
- A formal specification should meet certain mathematical/computational requirements
  - Unique readability;
  - Understandable to both lawyers and programmers;
  - Consistency;
  - Feasibility (time and space);

Ingredients for software homologation

- Where (in what language) is the regulation specified?
- A formal specification should meet certain mathematical/computational requirements
  - Unique readability;
  - Understandable to both lawyers and programmers;
  - Consistency;
  - Feasibility (time and space);
  - Locality;

Ingredients for software homologation

- Where (in what language) is the regulation specified?
- A formal specification should meet certain mathematical/computational requirements
  - Unique readability;
  - Understandable to both lawyers and programmers;
  - Consistency;
  - Feasibility (time and space);
  - Locality;
  - Domain specific ad-hoc requirements (eg., shift invariance);

Ingredients for software homologation

- Where (in what language) is the regulation specified?
- A formal specification should meet certain mathematical/computational requirements
  - Unique readability;
  - Understandable to both lawyers and programmers;
  - Consistency;
  - Feasibility (time and space);
  - Locality;
  - Domain specific ad-hoc requirements (eg., shift invariance);
  - Legal principles can be formalised and proven.

Ingredients for software homologation

- Where (in what language) is the regulation specified?
- A formal specification should meet certain mathematical/computational requirements
  - Unique readability;
  - Understandable to both lawyers and programmers;
  - Consistency;
  - Feasibility (time and space);
  - Locality;
  - Domain specific ad-hoc requirements (eg., shift invariance);
  - Legal principles can be formalised and proven.
- Extracted code that inside Coq is provably complying the formal specification

Ingredients for software homologation

- Where (in what language) is the regulation specified?
- A formal specification should meet certain mathematical/computational requirements
  - Unique readability;
  - Understandable to both lawyers and programmers;
  - Consistency;
  - Feasibility (time and space);
  - Locality;
  - Domain specific ad-hoc requirements (eg., shift invariance);
  - Legal principles can be formalised and proven.
- Extracted code that inside Coq is provably complying the formal specification
- Is the formal specification the intended specification?

Some results

- Papers mathematically/logically/computationally analysing the law, see, e.g.

  `http://www.joostjjoosten.nl/papers.html`

  most notably
  A. de Almeida Borges, J. J. Conejero Rodríguez, D. Fernández Duque, M. González Bedmar, J. J. Joosten. **To drive or not to drive: A logical and computational analysis of European transport regulations.** *Information and Computation*, 2020.
  `https://doi.org/10.1016/j.ic.2020.104636`

Some results

- Papers mathematically/logically/computationally analysing the law, see, e.g.

  `http://www.joostjjoosten.nl/papers.html`

  most notably
  A. de Almeida Borges, J. J. Conejero Rodríguez, D. Fernández Duque, M. González Bedmar, J. J. Joosten. **To drive or not to drive: A logical and computational analysis of European transport regulations.** *Information and Computation*, 2020.
  `https://doi.org/10.1016/j.ic.2020.104636`
- Analysis of erroneous applications of the law:
  `http://formalvindications.com/#document`

Some results

- Papers mathematically/logically/computationally analysing the law, see, e.g.

  `http://www.joostjjoosten.nl/papers.html`

  most notably
  A. de Almeida Borges, J. J. Conejero Rodríguez, D. Fernández Duque, M. González Bedmar, J. J. Joosten. **To drive or not to drive: A logical and computational analysis of European transport regulations.** *Information and Computation*, 2020.
  `https://doi.org/10.1016/j.ic.2020.104636`
- Analysis of erroneous applications of the law:
  `http://formalvindications.com/#document`
- Time Library;

Some results

- Model checking fragment capable of ‘feasible interval calculus’: forthcoming;

Some results

- Model checking fragment capable of ‘feasible interval calculus’: forthcoming;
- Standards for documenting (forthcoming);

Some results

- Model checking fragment capable of 'feasible interval calculus': forthcoming;
- Standards for documenting (forthcoming);
- Law simplifications to stay in feasible part (Splitter Theory, forthcoming);

Some results

- Model checking fragment capable of 'feasible interval calculus': forthcoming;
- Standards for documenting (forthcoming);
- Law simplifications to stay in feasible part (Splitter Theory, forthcoming);
- LawMaker (forthcoming);

Some results

- Model checking fragment capable of 'feasible interval calculus': forthcoming;
- Standards for documenting (forthcoming);
- Law simplifications to stay in feasible part (Splitter Theory, forthcoming);
- LawMaker (forthcoming);
- Future projects: semi-natural language with clear ontologies that is readable for lawyers yet directly translates to formal specifications.

Some results

- Model checking fragment capable of 'feasible interval calculus': forthcoming;
- Standards for documenting (forthcoming);
- Law simplifications to stay in feasible part (Splitter Theory, forthcoming);
- LawMaker (forthcoming);
- Future projects: semi-natural language with clear ontologies that is readable for lawyers yet directly translates to formal specifications.
- EPN expectations: collaborate and interact with similar efforts

Some results

- Model checking fragment capable of 'feasible interval calculus': forthcoming;
- Standards for documenting (forthcoming);
- Law simplifications to stay in feasible part (Splitter Theory, forthcoming);
- LawMaker (forthcoming);
- Future projects: semi-natural language with clear ontologies that is readable for lawyers yet directly translates to formal specifications.
- EPN expectations: collaborate and interact with similar efforts
- Forthcoming event:
  `https://www.ub.edu/prooftheory/event/lawdesign/`
  Conference on Algorithmic Law Design and Implementation