Analysis of the USA & Canadian ELD technology with "Code of Federal Regulations, 49: Transportation, Part 395 Hours of Service Drivers" & "Commercial Vehicle Drivers Hours of Service Regulations (SOR (Statutory Orders and Regulations)/2005-313)", and its consequences for North-American citizens

Guillermo Errezil Alberdi
Formal Vindications S.L.

in collaboration with:

Joaquim Casals Buñuel
Formal Vindications S.L.

Mireia González Bedmar
Formal Vindications S.L.

July 15, 2024

Contents

1 Introduction: dangerous software

EXECUTIVE SUMMARY

Industrial software always contains mistakes and errors. This can lead to disastrous consequences, huge losses and casualties. Worldwide there is no standard way to **certify software**. **America can become a leader in this field** and in this introduction we put forward a revolutionary proposal. We see how not applying our proposal leads to unworkable situations in the best and disasters in the worst case scenarios.

Next, we focus on a case study: the Electronic Logging Device. The ELD is a device to measure driving activities for road transportation. Unlike the detailed fashion in which European regulations stipulate how ELDs should behave, in the USA there exists the option for companies to self-certify their own ELD design, and thus there are a great variety of ELDs. Nevertheless, in the remainder of this document, we expose in a more general setting how some shortcomings of the transportation regulation may impact ELD designs and in turn, affect the legal security of drivers and motor carriers.

1.1 Software contains errors

Industrial software always contains mistakes and errors. Most of the times, these errors go by unnoticed. But sometimes the outcome is disastrous. A classical example is the explosion of the Ariane 5 launcher in 1996 due to a software error. Media reports indicated that the amount lost was half a billion dollars – uninsured. This is a costly software error.

Figure 1.1: On June 4, 1996, the maiden flight of the European Ariane 5 launcher crashed about 40 seconds after takeoff.

The list of severe incidents due to software errors is long and very worrying including casualties, severe damage and civil right violations.

Various professions which come with serious responsibility –like medical doctor or lawyer– are severely regulated and controlled by the state authorities. However, anybody can call him or herself a programmer.

Notwithstanding this, as an example, worldwide regulators are trying to impose some quality control on critical software. Thus, there are standards around on how to develop software. They consist of lengthy documents that listen to poetic names like DO-178B and DO-178C, ISO 9001, ISO 13485, ISO 27001...

These current standards, however, are mainly based on imposing many sanity checks in the programming process on the one hand, and on what is called *dynamic testing* on the other hand: test your program on many samples for which you know how your program should behave and compare this to how the program actually does behave. By abiding to these standards, the amount of average errors goes significantly down as we can see in the following table.

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

http://leanagilepartners.com/publications.html

Figure 1.2: This table shows the average amount of errors per Kloc, that is per thousand (Kilo) lines of code.

As we see in the table above, skilled (agile) programmers with a university background typically make around 22 errors per thousand lines of code. Applying all the involved current state-of-the-art standards to programming we see that there are relatively very few errors in the code, about one error per ten thousand lines of code. But, how do we know this error is not fatal? A typical medium-sized industrial program module can be in the order of a million lines of code. This leaves us with around a hundred errors.

We see that despite regulator's best efforts, their proposals fall significantly short.

The important message to learn is:

With the state-of-the-art quality standards for software development **there will always remain some errors in the code.**

The question begs itself:

Is there really no way to obtain zero errors? Here we mean really zero, not zero comma something. No, a round zero, **NO ERRORS AT ALL**.

In the next section we see how, in a certain sense, this can indeed be accomplished.

The techniques that we will expose are powerful enough to guarantee zero errors. Since the technique is young and much under developed America still has a chance to fully invest in it and become the undisputed world leader in what we call *software verification*.

1.2 Public Certification: zero error software

The main question is:

how can we be **a hundred percent sure** that our software does not contain errors?

In this section we present a new technique where all errors of a certain kind will be eliminated with hundred percent certainty. This is a revolutionary and abysmal change with respect to conventional programming.

The basic question: how and when can we be hundred percent sure of some knowledge? Is it possible to know something for sure? And if so, in what situations can we have hundred percent certainty that what we state is really true?

We want to know that some software always does what it should do, for any of the infinitely many possible inputs. So, we cannot check them all one by one. How can we be hundred percent sure that our software is correct?

Public Certification through Formal Verification

It may be clear that testing a software can never guarantee that the software always works well: there are simply too many possible different inputs and they cannot all be tested for.

Our standpoint is: **only mathematical proof can provide full certainty.** In particular, only mathematical proof can provide full certainty that a program always does what it should do.

In the past decades very deep mathematical theory and machinery has been developed to do what is needed to truly certify software: to mathematically prove correctness of the program. This process is called *Formal Verification*.

Formal verification is obtaining mathematical proof that the software behaves according to a formal-language specification. But we must acknowledge that this still raises some questions. The pitfall here is that a formal-language specification is not usually directly understandable for humans. This is one of the biggest holes in the mathematics of computer science: **What can be proven cannot be understood, and what can be understood cannot be proven**. The consequence is that, even if we have formally verified software, bugs can arise due to the misunderstanding of the original idea of the specification, resulting in a formalization that does not behave as expected.

When designing software, humans have in mind a intuitive specification which then is translated to a formal specification, but there is no guarantee that the translation keeps the intended meaning. Even more, depending on the level of abstraction of the language used, the same specification can be understood differently by two people, i.e., it can be ambiguous (this document included!). **In general, the equivalence between two different abstraction levels of**

language specification cannot be proven, i.e., this gap between different languages cannot be completely overcome. However, public certification can narrow it considerably.

Publicly certified specification and its software (PCSS) consists of a verified software that, moreover, comes with an interpretation of the formal specification written in some accessible language, in such a way that at least an individual with mathematical training can check that the translation from the formal specification to a human language is, up to some extent, correct.

Not everything that can be written in natural language as a computable law can be developed in software and have public certification. Natural language specifications are often ambiguous, inconsistent, or unfeasible. This is why public certification can bring unambiguity and consistency to the realm of computable laws: when something is publicly certified, it comes with guarantees that it has a translation into formal language and satisfies basic properties.

FORMALLY VERIFIED SOFTWARE consists of three components.

First, there is a Technical Specification (Σ) in a precise formal mathematical language that tells with mathematical precision and rigor what the software should do. It is completely unambiguous in the strict mathematical sense.

Second, there is a software (Π) that executes according to the specification.

The **third and key ingredient** to formally verified software is a **mathematical PROOF/DEMONSTRATION** (Δ) that the software (Π) does exactly what the specification (Σ) says it should do.

Thus, Formal Verification delivers a triple (Σ, Π, Δ). **The proof (Δ) replaces and outperforms Dynamic Testing** and is the only reasonable base for Real Certification.

This proof (Δ) guarantees that:

(i) **ZERO** bugs will appear in the resulting code and also

(ii) that the given specification is exactly reproduced by the code.

PUBLIC CERTIFICATION OF A SPECIFICATION AND ITS SOFTWARE consists of four components $(\Sigma, \Pi, \Delta, \Phi)$, where:

- (Σ, Π, Δ) is formally verified software.
- Φ is an interpretation of the specification Σ written in an accessible language for human comprehension, the closest possible to Σ, with a systematic method, to prevent the most common bug due to "misundertanding". Therefore, Φ is the guarantee that the public certification of a specific software made by a team can be reached with the same result by other teams.

In particular, the specification itself must fulfil the following requirements for *decent design*:

DECENT DESIGN REQUIREMENTS

- Type1: A specification must follow the following ***logical-mathematical principles*** and in particular should be *consistent*: no contradictions are entailed. A desirable additional property is *completeness*: the system will decide all situations.
- Type2: It must respect ***computational limits*** (not exceeding available computation time and memory).
- Type3: It must follow ***physical laws***.
- Type4: The formal specification ***represents the law accurately***.

As we shall see, many of the decent design requirements are often violated.

1.3 Illogical software specifications

Consider the following rule concerning a 20th century regulation for railway in Kansas:

`When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone.`

Figure 1.3: Railway crossings naturally give rise to critical situations. Here software better not fail. As a matter of fact, the railway industry is one of the first to apply methods of formal verification.

Clearly this regulation violates Type 1 requirements on decent design. If Kansas policy makers hired a software company to develop this law in a digital-automatic environment, the company would find the deadlock and should try to implement one of the following decisions:

1. **Follow the law** and have the trains stop forever.

2. **Self-fix**: For example by giving preference to a train on one side.

   Consider: This solution would mean that the software would be breaking the law. Moreover, the software engineers would be legislating at their will (consciously or otherwise).

3. **Engage with the policy makers to amend the law**, assuming that they understand basic mathematical properties. In this particular case, they should be able to grasp the inconsistency of the law with simple requirements on operability.

We will extrapolate and carry this case over to a more complex technology in order to show under close analysis that this undesirable situation is being experienced today within USA and Canadian regulations.

2 Europe and America

2.1 Contextual similarities and differences

As one can imagine, road transportation regulations exist all over the world. The European regulations are similar to the ones found in North America, and the use of electronic devices for recording the activities of drivers are also enforced. Therefore, the issues that arise from the use of non-certified software are similar in Europe and North America. Our team has an extensive experience treating this kind of cases in Europe, and actually a document with the same purpose of this one was developed and published for the European case.[1]

However, in Europe, ELDs (called tachographs) are manufactured only by two companies, and one of them has the 85% of the market, while in the USA there are more than 300 manufacturers with the prerogative of self-certification. This is why the European case allows for a big practical and experimental grounding, which is not possible in North America.

Here, there are two different problems. On the one hand, companies certifying their own software with no formal, rigorous and unambiguous standards imply an evident danger and also a conflict of interest. Recall, for instance, the issue with the device Samsung Galaxy Note 7, whose battery could combust or explode suddenly. This model had the CE marking, which means that Samsung had self-certified its safety[2]. Another –more malicious– example is Volkswagen emissions scandal, also known as *Dieselgate*: the company installed software to the emissions engine of the vehicles that made them respect the legal limitations on emissions only during regulatory testing.[3].

On the other hand, the fact that there are 300 versions of ELD design working at the same time and regarded as proprietary hardware and software implies not only possible incompatibilities between them, as we shall see, but also the complete impossibility to verify that their behavior corresponds to the law theoretically or practically.

In the case of Canada, the Mandate went into effect on 12 June 2021, however, no penalties will be issued until 12 June 2022. In any event, the ELD project includes only minimal differences with the American requirements for the *driver file*, as we shall see below.

2.2 Differences on USA and Canada driver files

This document analyses USA and Canadian road transportation cases at once because both settings are extremely similar. The format and content requirements for both countries is almost identical, except for a small number of variables that differ due to particularities of their respective regulations, plus the units used. This justifies a unified treatment of the problem. In this section we give a summary of the small differences that our detailed analysis of the required structure for driver files in both countries has enlightened.

One of the key differences is the unit for distance and speed. According to the USA requirements for ELDs, distances must be registered and displayed in **miles**. In the Canadian case, distances can be registered either in miles or kilometers, but it is required that the ELD is capable of displaying them in **kilometers**.

[1] *Industrial Software Homologation: Theory and case study Analysis of the European tachograph technology with EU transport Regulations 3821/85, 799/2016, and 561/06 and their consequences for Europeans citizens*, in `http://formalvindications.com/pdf/DocumentoHomologacionSoftwareEN.pdf`.

[2] `https://www.digitaltrends.com/mobile/battery-explosions-whats-going-on`

[3] `https://detroit.cbslocal.com/2015/09/21/epa-volkswagon-thwarted-pollution-regulations-for-7-years`

Regarding the structure of the driver files, the technical requirements for both the USA and Canada list over 100 fields distributed in several segments (like Header, User List, Event List...). In the Canadian requirements, a small number of these fields appear as Reserved, and hence we have no data to know if they coincide with the corresponding American fields. Moreover, there are a few fields which differ from one country to the other. Apart from those considerations, all of these fields are identical for both countries.

The table below details the few differences between both models.

Data field	Description (USA)	Description (Canada)
`Multiday-basis Used` (USA) `Cycle Used` (Canada)	This data element refers to the multiday basis (7 or 8 days) used by the motor carrier to compute cumulative duty hours. 1 character. 7 or 8.	This data element refers to the cycle 1 (7 days) or cycle 2 (14 days) used to compute cumulative duty hours Data Range: 7 or 14 Data Length: 2 characters
`Carrier's USDOT Number`	An integer number of length 1-8 assigned to the motor carrier by FMCSA (9 position numbers reserved). Minimum: 1; Maximum: 9 characters.	No data (reserved field)
`Shipping Document Number`	Any alphanumeric combination. 0-40 characters.	No data (reserved field)
`Off-Duty Time Deferral Status`	Does not exist	This data element states that the driver is deferring off-duty time and clearly indicates whether the driver is driving under day one or day two of that time. Data Range: 0 (none) or 1 (day one) or 2 (day two). Data Length: 1 character.
`Off-Duty Time Deferred`	Does not exist	This data element refers to the off-duty time deferred by the driver.
`Operating Jurisdiction`	Does not exist	This data element refers to the operating jurisdiction (south or north of latitude 60°N) used to compute cumulative duty hours.
`Motor Carrier's Address`	Does not exist	This data element refers to the motor carrier's addresses. This parameter is a placeholder for <{Home Terminal} Address>, which refers to the address of the home terminal location defined by the motor carrier, and <{Principal Business} Address>, which refers to address of the principal place of business defined by the motor carrier.

Figure 2.1: Fields presenting differences between the USA and Canada ELD requirements for driver files.

3 Doubts about USA/Canada ELDs

3.1 Driver logged into two ELDs at the same time

> Legal texts providing technical specifications for ELDs do not consider the case of a driver logging into two ELDs at the same time, using the same account or two different ones. It seems clear that the possibility of using the same account in two vehicles at the same time should be prohibited by law, but since that is not the case, hazardous consequences might be faced.
>
> This possibility shows how an incomplete or ambiguous law leaves details that are important for safety in the hands of software engineers.

A close analysis of 49 CFR Part 395 & SOR/2005-313, as well as all the supporting documentation on the topic, reveals a disturbing fact: no law in the USA or Canada forbids the possibility of a driver to log into two ELDs (of different vehicles) at the same time, using the same account or two different accounts. This might be an open door to fraud and an impossible puzzle to unify data and compute the driving time.

If different ELDs register different activities for the same driver at the same time, and taking into account that this is not explicitly forbidden by the law, how is the driving and on-duty time going to be computed for this driver? It seems intuitively clear that this should be considered illegal, but the law does not account for this case. Then, at each particular case, a debate on whether the activities that the different ELDs registered are compatible or not could happen. Of course, it seems clear that it is physically impossible for a driver to drive two different trucks at the same time, but what if one truck registered *Driving* while the other registered *On duty not driving* or *Off duty*? This might also happen with drivers that work for two different companies at the same time.

The answers to these questions could be important in some contexts, but here the relevant thing is that, **since logging from two ELDs at the same time is not forbidden by the law, the solution to the cases where this happens is arbitrary and left in the hands of software engineers or administrative staff**.

Moreover, this issue reveals that **the law is incomplete**, and hence as a specification it leaves room to free interpretation. The fact that this possibility is not considered in the law seems a slip, a forgotten detail, and it might as well be a forgotten detail for the software engineers. What if, once the data of both ELDs is transferred to the carrier's servers, it is unified with no further processing or detection that something strange happened? For example, suppose that we check the interval from 6 AM to 7 PM one day, and the first ELD registered something like

$$\begin{array}{lcl} 06{:}00 & \rightarrow & \text{Driving} \\ 18{:}00 & \rightarrow & \text{On-duty} \end{array}$$

while the second one registered

$$\begin{array}{lcl} 07{:}00 & \rightarrow & \text{Sleeper-berth} \\ 16{:}00 & \rightarrow & \text{Driving} \\ 19{:}00 & \rightarrow & \text{Off-duty} \end{array}$$

Whoever was driving the first truck was driving most likely[4] for 12 hours straight, an illegal situation. Meanwhile, the driver on the second vehicle drove for only 3 hours. However, if these

[4] we cannot be sure with only this information, see Section 3.3.

two loggings belong to the same account and are mixed with no further detection (because the law does not consider this case), then we get the following:

06:00	$\rightarrow$	Driving
07:00	$\rightarrow$	Sleeper-berth
16:00	$\rightarrow$	Driving
18:00	$\rightarrow$	On-duty
19:00	$\rightarrow$	Off-duty

which would be interpreted as a lawful scenario: 3 hours of driving time and 1 more hour on duty! Of course, a thorough check of the data, which includes for instance the identification of the vehicle, would show that the data comes from different ELDs. However, the most likely scenario is that nobody checks for the integrity of the data, since apparently everything is correct. Therefore, a violation that implies safety risks for North-American citizens can go unnoticed, depending only on the decisions that software engineers made, consciously or not.

One can think this is only a speculative example of what could happen. However, what this example shows is that **the incompleteness and ambiguity of the regulation can lead to potential hazards, and leaves them in hands of engineers**.

3.2 Incompatible ELD providers

Poor specifications for ELDs in the law yields the existence of different, incompatible ELDs in the market, in such a way that the law provides a non-digital, non-secure way of presenting the driver's loggings to law inspectors. Therefore, **a gate is open for carriers to purposedfully infringe the law and get away with it, introducing a disruption to free market and a way of violating civil rights of the drivers**.

As mentioned in the introduction, in the USA there are more than 300 manufacturers of ELDs. This means that, potentially, each manufacturer may implement a different and incompatible system for drivers to register their activities, implying that whenever a driver changes vehicle, the ELD, system and recordings may change. This poses an important challenge to carriers: How should we handle potential incompatibilites within our vehicle fleet? It is also a known and yet non-addressed issue for drivers that are employed in different carriers at the same time.

In the USA it is required that any driver shall be able to produce the required information to evaluate their activity history of the last 7 days[5]:

> Q11. What procedure should be followed if multiple, incompatible electronic logging devices (ELDs) are used to record a driver's record of duty status (RODS)?
>
> A11. The motor carrier and the driver are responsible for ensuring that all of the RODS information required by the HOS rules is available for review by authorized safety officials at the roadside. If the driver uses multiple ELDs that are not compatible (e.g., the data file from one system cannot be uploaded into the other system), the driver must either manually enter the missing duty status information or provide a printout from the other system(s) so that an accurate accounting of the duty status for the current and previous seven days is available for the authorized safety official.

The above implies that the driver has to be aware of possible incompatibilites within the trucks they drive and, if that is the case, carry with them all their recordings, an unnecessary hassle that should be avoided by automation and a standarized way to keep track of the drivers recordings.

But more importantly, this opens the gate for carriers to **willingly infringe the law** by buying incompatible ELD and forge false driver's history on paper sheets. Because it is so much easier to produce a recording in paper than it is through an ELD, this inconsistency would **incentivize unlawful agents to disrupt the market via driver's exploitation**.

Moreover, knowing that ELDs do not always record the true activities of the drivers (see Section 3.3), and knowing that printings do not contain all the information about cuts or malfunctions even if they are authentic, **printed recordings are an insufficient basis to determine whether a driver complies with the law or not**. Furthermore, **it is impossible for a law agent to verify *in situ* and with no computer assistance whether the activities of the driver are legal or illegal**.

Even if no manipulation is performed, suppose a driver drives three different vehicles, call them A, B and C. The driver is stopped for an inspection when driving the vehicle A, and he presents only the printings obtained from vehicle B. If he was driving overhours in vehicle C, **there is no way the law agents would detect it, and no manipulation of data was performed!**

[5] *FMCSA editing and annotations* `https://www.fmcsa.dot.gov/hours-service/elds/editing-and-annotations`

But there is more. The law stipulates that a carrier should be able to list all the recorded activity of any of their drivers for the last period of six months. If a driver drives incompatible vehicles (ELD-wise), how is the carrier supposed to process these recordings in order to be able to have a coherent driver history? **This process is nowhere specified and is left up to interpretation.** A process that must be performed by carriers but that is not specified by the law implies another open gate to intentional or unintentional manipulation of the data.

An automated solution would not only make life easier for drivers and carriers alike, but would also facilitate the work of the authority in checking whether a driver is following the regulations or not, and would avoid fraud and manipulation and improve the productivity of the transportation sector.

3.3 What is driving time?

One issue with the law is that it does not tell us how to compute the central notion of *driving time* in an interval. The notion seems intuitively clear but there are various essentially different possibilities.

What is driving time duration in an interval?

Close investigation of the legal texts reveals that there is no *formal* definition of **driving time duration in an interval** in SOR/2005-313 or 49 CFR Part 395. Admittedly there is a definition in section 2 of 49 CFR Part 395:

Driving time means all time spent at the driving controls of a commercial motor vehicle in operation.

But this definition has two problems:

- The definition is anything but formal, appealing to a vague notion of time with which humans can work but is not good enough for law specification, and
- it tries to specify what *driving time* is, but the most important part, in order to enforce the law, is **how driving time is computed**.

So how is it possible? Driving time is the central notion of the ELDs functionality. How has the ELD calculated the daily driving time without a proper definition? Obviously, this poor specification enables programmers to be lawmakers in disguise.

We must conclude that driving time duration in an interval is nowhere precisely defined. It rather appeals to some intuition but there are various possible ways to compute it. Yet programmers have to compute driving time duration in an interval in all kinds of legal software. Can the output of such programs then be used as legal evidence?

At this moment, this seems to be a problem of Type4, that is to say, the software Π of the ELD has a definition for driving time duration that does not exist in technical specification Σ.

We have to take into account that an ELD is not a device to put a seal on a log-file of undisputed physical measurements, **it is a device used to calculate drivers' driving time**.

3.3.1 Cut theory and the concept of UDA

A power supply interruption while a long driving period takes place can mean that the data is not properly recorded. Since the law fails to recognize periods where no information is recorded by the ELD, this is not detected unless a deep analysis of the files is performed.

Consider this hypothetical scenario:

A driver starts driving, and thus the ELD correctly registers that. After 2 hours on the road, the ELD has its power supply cut-off during 2 hours due to some malfunctioning. Coincidentally, the driver takes 1 hour break while the ELD has no power supply. When the driver gets back on the road the ELD recovers its power supply and correctly registers that the driver is driving. The scenario could look something like this:

$$\begin{array}{ll} [06{:}00,\ 08{:}00] & \rightarrow \text{ Driving} \\ [08{:}00,\ 09{:}00] & \rightarrow \text{ Off-duty} \\ [09{:}00,\ 11{:}00] & \rightarrow \text{ Driving} \end{array}$$

Now, let us see what the ELD would actually registers:

06:00	→	Driving
ELD power cuts off, so no data		
10:00	→	Driving

If this driver were to be stopped at a checkpoint, and their ELD checked, the conclusion would be that the driver has been driving, uninterrupedtly, since 6 AM which would amount to 5 hours of driving. But the truth is that the driver has been driving, not only one hour less, but also the driver has taken a one hour rest, which is a crucial piece of information in the context of road transportation regulation.

Thus, it becomes evident that we need to account for periods of *NO DATA*. In order to observe these cases, we introduce the rudiments of a general framework that aims at solving this type of problems. We name this framework *Cut Theory*.

We argue that the law should either recognize a new driver status *No-data*, to be assigned to periods with no reliable information, or state that those periods should be automatically set to *Off-duty*, to avoid unfair fines.

Cut Theory

The law should either recognize a new driver status *No-data* to be assigned to periods with no reliable information, or state that those periods should be automatically set to *Off-duty*, to avoid unfair fines. We propose that ELD log analysis software sould allow the user to automatically assign the periods with no data to *Off-duty*, since in the absence of data an *Off-duty* period is the safest assumption to protect the driver.

When there is a power supply interruption while long driving periods take place, the data is therefore not properly recorded.

The following table accounts for the whole panorama, taking into account that there are different scenarios that the law does not distinguish. In particular, **the law recognizes and registers the state of *Engine power up/shut down*, but not the state of *ELD power up/shut down***, which is a different thing that happens in reality but is not registered.

Duty status	Event registered immediately afterwards	What should be considered
Driving	"Log out"	?????*
On-duty	"Log out"	?????
Off-duty	"Log out"	?????
Sleeper-berth	"Log out"	?????
Driving	"Shut down engine/disconnected"	?????*
On-duty	"Shut down engine/disconnected"	No issue, allowed
Off-duty	"Shut down engine/disconnected"	No issue, allowed
Sleeper-berth	"Shut down engine/disconnected"	No issue, allowed
Driving	"Shut down ELD"	?????
On-duty	"Shut down ELD"	?????
Off-duty	"Shut down ELD"	?????
Sleeper-berth	"Shut down ELD"	?????

Table 3.1: UDA concept excerpt. (*) In theory these two situations cannot happen, but since the ELDs fail and are not formally verified, it actually happens.

We propose a framework that takes into account more than just the duty status registered in the ELD to determine the driver's current activity. The regulation

should differentiate between an ELD power supply cut-off and an engine cut-off, since we have seen that they might correspond to different scenarios.

Table 3.1 shows the different scenarios for which the regulation should provide an answer.

We propose what we call Unit Driver Activity (UDA) (table 3.1 is just a small sample of this framework) which is a whole framework which takes into account more than just the driving status registered in the ELD in order to

1. regulate more scenarios, and
2. avoid poorly specified regulations that might lead to inconsistent laws.

In a bit more detail, the UDA aims at contemplating more than just the basic status (i.e Driving, Off-duty, etc.) incorporating other events that might impact the driver status. The UDA framework does not provide an answer as to what the missing parts of table 3.1 should be, but it recognizes that **it is crucial for those scenarios to be thoroughly and correctly stipulated in the road regulations if one expects to achieve any kind of consistent laws**.

3.3.2 Unnoticed applications of UDA

We have presented cases where the responsibility of a lack of reliable data is a power-supply interruption of the ELD. But there are other cases where the law itself, by design, forces the software to make decisions on what to do in the periods with no data.

> "4.9. Data Transfer Capability Requirements
>
> [...]
>
> 4.9.1. Data Transfer During Roadside Safety Inspections
>
> (a) On demand during a roadside safety inspection, an ELD must produce ELD records for the current 24-hour period and the previous 7 consecutive days in electronic format, in the standard data format described in section 4.8.2.1 of this appendix."

But this means that the data produced by the ELD may very well start with an on-duty or driving period, which means that some rules cannot be checked without making assumptions on the previous data. For example:

> "(2) 14-hour period. A driver may not drive after a period of 14 consecutive hours after coming on-duty following 10 consecutive hours off-duty."

Or:

> "(ii) Interruption of driving time. Except for drivers who qualify for either of the short-haul exceptions in §395.1(e)(1) or (2), driving is not permitted if more than 8 hours of driving time have passed without at least a consecutive 30-minute interruption in driving status. A consecutive 30-minute interruption of driving status may be satisfied either by off-duty, sleeper berth or on-duty not driving time or by a combination of off-duty, sleeper berth and on-duty not driving time."

This means that the software must make a decision on how to evaluate these periods of time. Most software providers may choose to assume *Off-duty* when there is no data. The problem is that **they are making decisions that are not stipulated by the regulation**.

Moreover, if the driver is engaged in the 8-day cycle, there is not enough data to evaluate even one complete cycle, while there is enough data to evaluate a 7-day cycle, which is an arbitrary difference of treatment by the law.

3.4 Driving status according to speed is inconsistent

In Section 3.3 we have seen that regulations lack a formal, rigorous definition of driving time, and that the usual definition taken by software is not reliable. In this section we present the inconsistencies arisen from taking the speed measured by the ELD as a reliable parameter to determine alone the driving status of a driver.

There are several issues regarding ELD measurement of the speed of a vehicle and its use as parameter to determine the driving status. The measurement itself is not reliable and can be easily manipulated, even unintentionally. But even if the numbers measured are correct, the ELD might be incapable of correctly determining the driving status.

Let us reproduce the definition of the vehicle motion status in the USA, according to 49 CFR Part 395 section 4.3.1.2, and in Canada, according to the CCMTA's Technical Standard for Electronic Logging Devices, section 4.3.1.2.

> "4.3.1.2. Vehicle Motion Status
>
> (a) An ELD must automatically determine whether a CMV is in motion or stopped by comparing the vehicle speed information with respect to a set speed threshold as follows:
>
> (1) Once the vehicle speed exceeds the set speed threshold, it must be considered in motion.
>
> (2) Once in motion, the vehicle must be considered in motion until its speed falls to 0 miles per hour and stays at 0 miles per hour [0 km/h in Canada] for 3 consecutive seconds. Then, the vehicle will be considered stopped.
>
> (3) An ELD's set speed threshold for determination of the in-motion state for the purpose of this section must not be configurable to greater than 5 miles per hour [8 km/h in Canada]."

The first inconsistency arising from this definition is taking the speed of the engine as parameter. The speed measured by an ELD coming from the engine is calibrated for a certain size of the wheel, and thus it changes if the size of the wheel is changed. Assuming a wheel with 1.1 meters [43.3 inches] of diameter and a vehicle speed of 90 km/h [56 miles per hour], a variation of 100 mm [3.94 inches] in the diameter of the wheel implies a difference on the speed measurement of 8.18 km/h [5 miles per hour]. This opens the gate to accomplish **false speed measurements, and thus driving status, intentionally or not**.

Formal Vindications S.L. has conducted experimental testing of speed measurements and suggests to use as an alternative the speed measured from the GPS signal, which according to those practical tests coincides precisely with a perfectly calibrated vehicle.

There is yet another problem arising from this definition. The definition of a stop needs 3 seconds of 0 speed measured by the ELD, both in USA and in Canada. If the engine is shut down before those three seconds elapse, the ELD cannot reach the speed information and never considers the vehicle stopped (until the engine is powered up again). This is not inconsistent in and on itself: it could be a consistent definition in some context, but does not reflect correctly what this context understands as driving. As an example, think of a specification that says that the engine always shuts down after 15 minutes of operation. This could be consistent for a missile, but it would be a huge problem for a passenger aircraft.

Therefore, the reason why this is a bug is not a physical or mathematical inconsistency, the reason is that the specification written on the law does not reflect the meaning that law designers wanted to convey. The notion of bug is very wide, as has been exposed by Formal Vindications S.L.,[6] and the only method that addresses all the aspects of this notion is the PCSS method presented in the Introduction.

In 2016, our company predicted the existence of this problem in the USA, and in an internal document, the USA authorities have recognized the existence of this problem in a particular case. The Federal Motor Carrier Safety Administration (FMCSA) issued in 2020 additional permissions to the driver to manually edit ELD recording of driving time when the vehicle has been powered off immediately after becoming stationary, since this driving time "is not recorded to the specifications required by 4.3.1.2 and 4.4.1.1 and therefore may be edited to the correct duty status."

3. **May a user edit or correct driving time that has been erroneously recorded by an ELD because the driver failed to change their duty status before powering off the CMV?**
   Yes, but only if the driving time was recorded by the ELD while the vehicle was powered off and the vehicle was not in motion during the period that is being edited or corrected. The driver edit limitation found in section 4.3.2.8.2(b) prohibits the editing of automatically recorded driving time. The intent of the specification that requires automatic recording of driving time is to ensure all movement of the CMV is captured. A CMV cannot be driven while powered off. The driving time following the power off cycle of a CMV not in motion, is not recorded to the specifications required by 4.3.1.2 and 4.4.1.1 and therefore may be edited to the correct duty status.

Figure 3.1: New FMCSA bulletin recognizing that a quick powering off of the vehicle causes the ELD to log false driving time.

The amendment effectively recognizes this problem. In particular, a driver may power off the vehicle for the weekend, and the full weekend could be recorded as driving.

Therefore, we reiterate that the plain activities recorded by the ELD are not a reliable representation of the driver's true activities and that the concept of UDA presented in Subsection 3.3.1 would solve these issues.

Summary

The speed measurements coming from the engine of a CMV are not reliable, since they depend on calibration and can be changed just by changing the size of the vehicle wheels, introducing serious differences.

Moreover, a shut-down of the engine implies also a turn-off of the ELD, and hence, when the engine is shut down within the legal threshold of 3 seconds after coming to 0 speed, the ELD fails to register that information, introducing false driving time that the American authorities have already recognized.

[6] See `https://formalv.com/Docs/SoftwareBug.pdf` for a longer development of this matter.

3.5 Time standard problem: UTC versus Unix

We start by reminding the reader of some notions.

- **Coordinated Universal Time**, or **UTC**, is the main standard through which the world regulates clocks and time. It strives to agree with solar time as much as possible, which means that it is adjusted with **leap seconds** from time to time.
- A **leap second** is an extra second occasionally added to UTC. As of July 2021, 27 leap seconds have been inserted since 1970.[7]
- **Unix** time (also known as POSIX time, or UNIX Epoch time) is a system for describing a point in time. It describes each point as the number of seconds elapsed since January 1st 1970 00:00:00 UTC (the **epoch**) *not including leap seconds*. Every day is treated as if it contains exactly 86 400 seconds, including the days when a leap second is introduced. It is used widely in Unix-like and other operating systems and file formats. However, Unix time is not a true representation of UTC, as a leap second in UTC shares the same Unix time as the second which came before it.

Regulations with a dependency on time systems across the world choose to require UTC, as it is the *de facto* civil standard. This includes road transport regulations in America.

However, a vast majority of the software tools to manage time conversions and arithmetic works with Unix, even when their documentation might claim to work with UTC. Sometimes, a note to the extent that leap seconds are not considered is included, while in other occasions not even a sentence is devoted to the issue, since leap seconds are widely unknown to software engineers.

As a paradigmatic example, consider the Microsoft DateTime library, which is widely used in this kind of applications. Microsoft states in its software documentation that no leap seconds are considered (see Figure 3.2). Hence, it clearly does not follow the UTC standard.

> Time values are measured in 100-nanosecond units called ticks. A particular date is the number of ticks since 12:00 midnight, January 1, 0001 A.D. (C.E.) in the GregorianCalendar calendar. The number excludes ticks that would be added by leap seconds. For example, a ticks value of 31241376000000000L represents the date Friday, January 01, 0100 12:00:00 midnight. A DateTime value is always expressed in the context of an explicit or default calendar.

Figure 3.2: Fragment from Microsoft's time library technical specifications.

While Microsoft, Android and iOS do not work in UTC, institutions working on critical software do - for instance, NASA.

Formal Vindications S.L. has developed a Time Library to manage time conversions and arithmetic, with two great particularities:

- It includes **leap seconds**, and hence complies satisfactorily with the UTC time standard.
- It is **formally verified**, which means that the correctness of the software with respect to a formal specification is proven mathematically.

Now, one can believe that the Unix vs. UTC issue only means that there is a 26 to 27 second delay on the time measured by ELDs, which would not be so grave. However, depending on

[7] Theoretically, leap seconds can also be subtracted, but this has never happen as of the writing of this document, and physicists believe it is very unlikely to ever happen.

how the ELDs compute the driver's activities for each minute, the consequences can be vast, and actually this happens in Europe and is arguably very likely to happen also in America.

To understand the problem, it must be said that ELDs in Europe work by recording an activity (driving, rest...) each second, but the law requires that they provide an activity for each minute. The law also intends to provide an algorithm to convert the recordings in seconds to loggings in minutes – and we say "intends" because we argue that the algorithm is given in a very ambiguous manner. Now, our team provided a mathematical proof that the same recording of activities at the resolution of seconds gives totally different results when converted to minute loggings depending on whether we consider minutes in UTC or in Unix. This issue has not been tested in American ELDs yet, but our experience tells us that it is very likely that it also happens there, as any ELD needs to decide what activity to assign to each minute according to the different activities performed during the seconds of that minute.

Every administration in the USA should verify whether the time managers used by ELD providers work in UTC.

An easy way to test this is computing the time difference between 2015-1-1 00:00:00 and 2020-1-1 00:00:00.

- If the time manager works in UTC, the result will be 157766402 seconds, due to the leap seconds of 2015 and 2016.
- If the time manager works in Unix, the result will be 157766400 seconds.

After this test, it is important to check as well what is the logic for adding and subtracting durations of each time manager. Things that sound as clear as "adding a month" are defined in different ways depending on the time manager used. For instance, Microsoft adds 1 to the field month of the date (inducing exceptions in the cases where the resulting date would not exist), while others decide the duration of a month arbitrarily. The FV Time Manager defines formal time, a specification of durations, in which, e.g., a formal month corresponds to 30 formal days. Moreover the FV Time Manager has both functionalities: one that works like the Microsoft one, but including leap seconds, and one that works with the definition of formal time. Everything is formally specified and verified. To the best of our knowledge, **the FV Time Manager is the only time manager in the market that is verified and works according to UTC**, which is crucial for many applications: satisfying regulations, working as a provider for the NASA, etc.

See: *Industrial Software Homologation: Theory and case study Analysis of the European tachograph technology with EU transport Regulations 3821/85, 799/2016, and 561/06 and their consequences for Europeans citizens.* Section 8.

3.6 Time zone problem

Problem: No rule to change period from UTC to offset hours

There is no specification that stipulates how software should deal with changes between time zones. Software used by authorities to check ELD recording files and search for infractions show dates in local time. Due to this lack of specification, we need to ask the software engineer team of each ELD recording analysis program to know how this issue was solved. But if there is no specification telling us how to perform this conversion, how can we guarantee that these different programs are working according to the regulation? (Whenever something is not specifically written in the regulation, the software engineers are 'free' to do what they want.)

Usually, most electronic devices must work with the UTC Time Standard, but laws and regulations are stipulated in local time zones. Therefore, the question arises: **How should we translate from one to the other?** This question is of vital importance since we need to reconcile what the regulation is stipulating with the electronic recordings, since the latter are the primary witnesses to a hypothetical infraction.

Software engineers and lawmakers might be surprised to see this topic discussed here, but there are some issues that must be taken into account. Although ELDs register time in UTC + the local offset, there is no specification as to in what standard the calculations are performed. Is the calculation done in UTC and then translated into the local time when printing the informatin out? Or is the calculation happening in local time? Depending on the answer, we may obtain different results.

Consider the following real example of list of events:

04:30 2018/11/04	*Sleeper Berth*	UTC
05:55 2018/11/04	*Driving*	UTC
06:05 2018/11/04	*Sleeper Berth*	UTC
03:00 2018/11/05	*On-duty not driving*	UTC

If we compute driving time as agreed above, we obtain the time interval [05:55, 06:05], i.e., 10 minutes of driving in UTC.

Now if we proceed to translate this event list in UTC time into New York City timezone (taking into account the daylight saving time change that took place on the 4th November 2018, passing from UTC-4 to UTC-5, i.e. 02:00 local were 01:00 local) we obtain the results in this Table:

00:30 2018/11/04	*Sleeper Berth*	UTC-4 local
01:05 2018/11/04	*Sleeper Berth*	UTC-5 local
01:55 2018/11/04	*Driving*	UTC-4 local
22:00 2018/11/04	*On-duty not driving*	UTC-5 local

Table 3.3: Note the change in order due to the time resetting when crossing the 2:00 AM mark

Now, driving time computed as defined above gives the time interval [01:55, 22:00], i.e., 20 hours and 5 minutes of driving in local time!

We see that **the same activities lead to very different driving times depending on the time system used** (UTC or local time).

3.7 Potential contradictions from manual entries

Our goal is to devise an algorithm that automates the analysis of ELD data and, ideally, is able to decide if a driver is abiding with the law or not. Therefore, the only source of relevant information should be the ELD recordings. However, **the transportation regulation observes a number of cases where the driver and/or the carrier introduce manual data to the ELD**, which can present problems. In some cases, the manual information is **supposed to determine which rules are applied to the driver**, introducing the possibility of **inconsistencies** between what the carrier or driver decided and what they did.

As a paradigmatic example, the multiday-basis used in the USA raises concerns on the reliability and consistency of the regulation. To justify this, in this section several examples and conclusions are presented.

> "395.3 Maximum driving time for property-carrying vehicles.
>
> (a) [...]
>
> (b) No motor carrier shall permit or require a driver of a property-carrying commercial motor vehicle to drive, nor shall any driver drive a property-carrying commercial motor vehicle, regardless of the number of motor carriers using the driver's services, for any period after—
>
> (1) Having been on duty 60 hours in any period of 7 consecutive days if the employing motor carrier does not operate commercial motor vehicles every day of the week; or
>
> (2) Having been on duty 70 hours in any period of 8 consecutive days if the employing motor carrier operates commercial motor vehicles every day of the week.
>
> (c) [...]"

Depending on the *Multiday-basis used*, periods of 7 or 8 days are examined. These periods begin "on any day at the time designated by the motor carrier for a 24-hour period", i.e., the carrier designates a time where the 24-hour periods start. **But this implies that the same activities of a driver can be considered legal or illegal depending on this manual, voluntary parameter**. There is an example of this issue for the 7-day case in Table 3.4, where the same shifts done by a driver are legal or illegal depending on the choice of initial time for days. An example for the 8-day cycle can be obtained similarly, just adding one more day in between.

Monday	Tuesday	Wedn.	Thursday	Friday	Saturday	Sunday	Monday
8-13 14-19	8-13 14-19	8-13 14-19	8-13 14-19	8-13 14-19	8-13 14-19	rest	7-12 13-18

Table 3.4: Example of shifts for 8 days for a driver. If this driver is engaged in the 7-day cycle, these shifts would be illegal in case the "24-hour period starting time specified by the motor carrier" is 8 am, but legal if it is midnight.

In this example, if the 24-hour period starting time specified by the motor carrier is midnight, the activities are legal: the cycle of 7 days starting on Monday midnight has $6 \cdot 10 = 60$ hours

of driving time. However, if the starting time is 8:00, the cycle starting on the first Monday at 8:00 has $6 \cdot 10 + 1 = 61$ hours of driving time, to account for the hour from 7:00 to 8:00 on the second Monday.

This means that the carrier can declare against its best interest.

Therefore, we propose:

1. Reliably gather all the necessary information to give an assessment of each scenario as close to reality as possible, and
2. Free the driver of the unnecessary burden of feeding this information to the ELD.

In the following table we provide a summary of these cases:

Data	**Place**	**Observation**
24-hour period	USA	The carrier has to determine the starting time of the 24-hour periods for the check of the 7-day or 8-day periods. A slight change in the starting time can determine whether the driver is fulfilling the 7-day/8-day law.
`Multiday-basis Used`	USA	Carrier's input of the cycle used might be contradictory with the carrier's legal status or among different drivers employed by the same carrier.

3.8 Inconsistencies leading to ambiguity and arbitrary interpretation of the legal text

Following the thread opened at the Introduction, we can pose ourselves several questions regarding laws such as 49 CFR Part 395.

1. Is it possible to prove that **the algorithm implementing 49 CFR Part 395 satisfies 49 CFR Part 395**?
2. Is it possible to prove that 49 CFR Part 395 is **mathematically consistent**?
3. Is it possible to prove that 49 CFR Part 395 is **complete**? Meaning, an algorithm implementing 49 CFR Part 395 would not need to make any choice which is not contained in 49 CFR Part 395. Or, in other words, that the software developer cannot introduce legal choices.

As we have already argued at the Introduction, the answer to the first two questions is no, and **the only solution is the PCSS method**. Because we cannot prove anything about a natural language law if we do not go through the need to formally specify its content.

Regarding 3, 49 CFR Part 395 is incomplete. We have already presented examples of this in Section 3.3. But there are at least two more inconsistencies that we present below.

3.8.1 First critical inconsistency

The first critical inconsistency lies on the definition of an equivalent way to take daily off-duty time, split into two breaks:

> "(g) Sleeper berths—(1) Property-carrying commercial motor vehicle
>
> (i) General. A driver who operates a property-carrying commercial motor vehicle equipped with a sleeper berth, as defined in §395.2, and uses the sleeper berth to obtain the off-duty time required by §395.3(a)(1) must accumulate:
>
> (A) At least 10 consecutive hours off-duty;
>
> (B) At least 10 consecutive hours of sleeper berth time;
>
> (C) A combination of consecutive sleeper berth and off-duty time amounting to at least 10 hours;
>
> (D) A combination of sleeper berth time of at least 7 consecutive hours and up to 3 hours riding in the passenger seat of the vehicle while the vehicle is moving on the highway, either immediately before or after the sleeper berth time, amounting to at least 10 consecutive hours; or
>
> (E) The equivalent of at least 10 consecutive hours off-duty calculated under paragraphs (g)(1)(ii) and (iii) of this section.
>
> (ii) Sleeper berth. A driver may accumulate the equivalent of at least 10 consecutive hours off-duty by taking not more than two periods of either sleeper berth time or a combination of off-duty time and sleeper berth time if:
>
> (A) Neither rest period is shorter than 2 consecutive hours;
>
> (B) One rest period is at least 7 consecutive hours in the sleeper berth;

(C) The total of the two periods is at least 10 hours; an

(D) Driving time in the period immediately before and after each rest period, when added together:

(1) Does not exceed 11 hours under §395.3(a)(3); and

(2) Does not violate the 14-hour duty-period limit under §395.3(a)(2).

(iii) Calculation—(A) In general. The driving time limit and the 14-hour duty-period limit must be re-calculated from the end of the first of the two periods used to comply with paragraph (g)(1)(i)(E) of this section."

This excerpt raises inconsistency. A first concern comes when a driver satisfies at the same time (i) and (ii), i.e., there are two off-duty/sleeper berth periods complying with (ii), but one of those periods, or even both of them, complies with (i) at the same time. How is this interpreted? The law does not specify this.

On the other hand, suppose that a driver takes their off-duty time by using the two equivalent periods defined in (ii), for several consecutive days. How are days delimited? Item (iii) specifies that the starting point to count the 11-hour maximum on driving time and the 14-hour duty-period limit, is the end of the first of the two periods, but what is the ending point? Is it the beginning of the first period of the next day, or the beginning of the second period?

The law fails to specify this behavior, **leaving room for decisions made by the sofware developers that are not in the Regulation**.

3.8.2 Second critical inconsistency

Regarding the maximum on-duty time on 7 o 8-day periods, the law opens the possibility of re-starting the cycle, but it fails to specify when and how:

"(b) No motor carrier shall permit or require a driver of a property-carrying commercial motor vehicle to drive, nor shall any driver drive a property-carrying commercial motor vehicle, regardless of the number of motor carriers using the driver's services, for any period after—

(1) Having been on duty 60 hours in any period of 7 consecutive days if the employing motor carrier does not operate commercial motor vehicles every day of the week; or

(2) Having been on duty 70 hours in any period of 8 consecutive days if the employing motor carrier operates commercial motor vehicles every day of the week.

(c) (1) Any period of 7 consecutive days may end with the beginning of an off-duty period of 34 or more consecutive hours.

(2) Any period of 8 consecutive days may end with the beginning of an off-duty period of 34 or more consecutive hours."

Taking into account the following definitions given in the law:

"Eight consecutive days means the period of 8 consecutive days beginning on any day at the time designated by the motor carrier for a 24-hour period."

"Seven consecutive days means the period of 7 consecutive days beginning on any day at the time designated by the motor carrier for a 24-hour period."

There are two extra decisions to make for the algorithm to work correctly, decisions that are not reflected in the Regulation.

1. There is no rule to define what to do when there is a change in the 24-hour period starting time, in such a way that there is an overlap with the previous period with the previous 24-hour period starting time.
2. There is no rule to apply the 34-hour rest, how to combine it with the 24-hour period starting time at the same date.

4 Conclusions

4.1 Panorama

Throughout this document, we have seen that law enforcement in the road transportation sector suffers all kinds of issues, mistakes and malfunctions. The regulations fail to define in a clear manner how ELDs should record data, how this data should be interpreted, and how the drivers should behave in many situations. We have seen a number of cases that enlighten us about malfunctioning of the ELDs or the way data is interpreted.

But there is more. The regulations include ambiguous articles and many exceptions. It is practically impossible for a human reader to grasp the data recorded by the ELD and check with no mistakes whether the driver complied with the law or not. This is why **the check of the data for compliance with the law is performed by software**, and not by human readers. Like any other, this software must behave according to a specification. In this case the specification is the law, the articles about hours of service for drivers. However, **the law is written in natural language** (English, French...)**, and therefore a translation into a formal or technical language must be performed to convert it into software**. Now, a specification in a formal language cannot have ambiguities, so in the translation process many things that in the original, natural language law are open to interpretation, become interpreted by the software engineers in charge of writing the software, possibly without their own notice.

We have seen in the sections above that **there are indeed many ambiguities and inconsistencies** in the North-American laws. At the same time, there are hundreds of ELD providers and with them come hundreds of different softwares to check the legality of the data. We can assure that **dozens or even hundreds of these different software behave differently in certain situations**.

4.2 Our proposal

Facing this scenario of chaos, we propose order. **Solving all of these issues requires a common effort to transition from a system of arbitrary, ambiguous and underspecified laws to one of unambiguous and completely specified laws**, at least those laws which are to be enforced by software.

This goal can only be achieved through the PCSS method presented in Section 1.2. For the purpose of realizing the PCSS for road transportation regulations, Formal Vindications S.L. has developed a publicly certified software which implements Splitter Theory. Splitter Theory is a mathematical framework developed by our team to represent the kind of rules that are usually found in road transportation regulations such as 49 CFR Part 395 in the USA or SOR/2005-313 in Canada. The theory allows to formulate most of the articles of these laws in an unambiguous way, as it is a formal language. It comes with a certain amount of simplification to avoid many of the problems that arise from the current convoluted and ambiguous laws.

Splitter Theory presents the following advantages:

1. The rules defined using Splitter Theory are **unambiguous**.
2. Since it is a formal language, the algorithms implementing it can be **formally verified**, and actually this is the current work in Formal Vindications S.L.
3. Formal languages are popular for being "hard to understand", but actually **any person can learn to understand a formal language**, as they are defined mathematically, starting with very simple concepts.

4. Splitter Theory is equipped with notations and explanations that ease the understanding by any person, expert or not.
5. The Φ part of the PCSS for Splitter Theory makes the specification accessible for non-experts. In this way, the law can continue its natural language existence, while making sure that the software implementing the law is unambiguously determined, giving law enforcement agencies the possibility of deciding about particular cases with the knowledge of what the software does precisely.

The philosophy behind Splitter Theory is to **avoid introducing rules that are not in the regulation. Splitter Theory keeps the consistent part of the law.** The same methodology has been used in the verification of CompCert. Since C has some inconsistencies as a programming language, CompCert takes a fragment or subset of C named C Light, which does not contain inconsistencies.[8]

[8]See, for example, the work done in Princeton University at `https://dl.acm.org/doi/10.1145/2775051.2676985`.